

AFIPS

**CONFERENCE
PROCEEDINGS**

VOLUME 33

PART ONE

1968

**FALL JOINT
COMPUTER
CONFERENCE**

**December 9-11, 1968
San Francisco, California**

**THE THOMPSON BOOK COMPANY
National Press Building
Washington, D.C. 20004**

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1968 Fall Joint Computer Conference Committee or the American Federation of Information Processing Societies.

Library of Congress Catalog Card Number 55-44701
THOMPSON BOOK COMPANY
National Press Building
Washington, D.C. 20004

©1968 by the American Federation of Information Processing Societies, New York, New York, 10017. All rights reserved. This book, or parts thereof, may not be reproduced in any form without permission of the publisher.

Printed in the United States of America

CONTENTS

PART I

TIME SHARING

The Pitt time-sharing system for the IBM systems 360.....	1	<i>G. F. Badger, Jr., E. A. Johnson, R. W. Philips</i>
Debugging in a time-sharing environment.....	7	<i>W. A. Bernstein, J. T. Owens</i>
TSS/360: A time-shared operating system.....	15	<i>A. S. Lett, W. L. Konigsford</i>

MIS VALUE ANALYSIS/JUSTIFICATION (A Panel Session—Papers not included in this volume)

RELIABILITY, MAINTENANCE AND ERROR RECOVERY IN THIRD GENERATION SYSTEMS

Considerations for software protection and recovery from hardware failures in a multiaccess, multiprogramming, single processor system.....	29	<i>G. Oppenheimer, K. P. Clancy</i>
Error recovery through programming.....	39	<i>A. Higgins</i>
OPTS-600—On line Peripheral Test System.....	45	<i>G. W. Nelson</i>
Fail-safe power and environmental facilities for a large computer installation.....	51	<i>R. C. Cheek</i>

NUMERICAL CONTROL

Methodology for computer simulation.....	57	<i>G. Chingari</i>
Subsets and modular features of standard APT.....	67	<i>C. G. Feldman</i>
A remote batch processing system for APT.....	75	<i>M. E. White</i>

THE COMPUTER FIELD: WHAT WAS PROMISED, WHAT WE HAVE, WHAT WE NEED (SOFTWARE SESSION)

Software compatibility: What was promised, what we have and what we need.....	81	<i>J. A. Gosden</i>
Interactive systems: Promises, present and future.....	89	<i>J. I. Schwartz</i>
Multiprogramming: Promises, performance and prospects.....	99	<i>T. B. Steel, Jr.</i>

APPLIED MATHEMATICS

An algorithm for finding a solution of simultaneous nonlinear equations.....	105	<i>R. Hardaway</i>
An economical method for calculating the discrete fourier transform....	115	<i>R. Yavne</i>
Nonlinear interactive stepwise regression analysis.....	127	<i>L. Edwin, A. Edwin</i>
Recursive fast fourier transforms.....	141	<i>G. Epstein</i>

THE IMPACT OF THE FCC INTERDEPENDENCE INQUIRY ON THE COMPUTER INDUSTRY (A Panel Session—Papers not included in this volume)

MIS DESIGN

Design and implementation of a general purpose management information systems data bank	145	<i>H. Liu</i>
OMNIBUS: A large data base management system	157	<i>R. P. Allen</i>
A design approach to user customized information systems	171	<i>R. G. Rangel</i>

TERMINAL LANGUAGES

Computer graphic language	179	<i>A. J. Frank</i>
Tymshare conversational language	193	<i>R. K. Moore,</i> <i>W. Main</i>
META PI—An interactive on-line compiler-compiler	201	<i>J. T. O'Neill, Jr.</i>

FRONTIER DIRECTIONS IN INTERACTIVE GRAPHICS

The organization and formatting of hierarchical displays	219	<i>G. T. Uber,</i> <i>P. E. Williams,</i> <i>B. L. Hisey,</i> <i>R. G. Wiekert</i>
Projection of multidimensional data for use in man-computer graphics	227	<i>T. W. Calvert</i>
The on-line firing squad simulator	233	<i>R. M. Balzer,</i> <i>R. W. Shirey</i>
Interactive telecommunications	243	<i>D. W. Cardwell</i>
Computer-driven display facilities for an experimental computer based library	255	<i>D. R. Haring</i>
Response time in man-computer conversational transactions	267	<i>R. Miller</i>

COMPUTER MODELS OF VISION AND SPEECH

Linguistic methods in picture processing: A survey	279	<i>W. Miller,</i> <i>A. Shaw</i>
Decomposition of a visual scene into bodies	291	<i>A. Guzman</i>
A limited speech recognition system	305	<i>D. Bobrow</i>
Computer models for speech and music appreciation	319	<i>P. Denes,</i> <i>M. V. Mathews</i>
A computer with hands, eyes, and ears	329	<i>J. McCarthy,</i> <i>D. Reddy,</i> <i>L. Earnest,</i> <i>P. J. Vicens</i>

DIGITAL SIMULATION OF CONTINUOUS DYNAMIC SYSTEMS— WHERE IS IT? WHERE IS IT GOING?

Digital simulation of continuous dynamic systems: An overview	339	<i>J. C. Strauss</i>
Mathematics of continuous system simulations	345	<i>D. H. Brandin</i>
SALEM—A programming system for the simulation of systems described by partial differential equations	353	<i>S. M. Morris,</i> <i>W. E. Schiesser</i>
TAF—A steady state frequency response and time response simulation program	359	<i>T. E. Springer,</i> <i>O. A. Farmer</i>

MEDICAL INFORMATION SYSTEMS

The automated medical history	371	<i>W. Weksel,</i> <i>P. N. Sholtz,</i> <i>J. Mayne</i>
-----------------------------------------	-----	--------------------------------------------------------------

The key to a nationwide capability for computer analysis of medical signals.....	381	<i>C. H. Caceres, G. Kishner, D. E. Winer, A. L. Wehrer R. N. Freed</i>
A legal structure for a national medical data center.....	387	<i>R. N. Freed</i>
A RESEARCH CENTER FOR AUGMENTING HUMAN INTELLECT		
A research center for augmenting human intellect.....	395	<i>D. C. Engelbart, W. K. English</i>
PLANNING MODELS FOR MANAGEMENT		
An approach to simulation model development for improved planning..	411	<i>J. McKenney</i>
Operations research in a conversational environment.....	417	<i>M. Conners</i>
MEDIAC—On-line media planning system.....	425	<i>L. Lodish, J. Little</i>
Development and use of computer models for the Southern Pacific Company.....	431	<i>A. Seelenfreund, E. P. Anderson, R. K. McAfee</i>
PLAIN TALK: MACHINES THAT SPEAK YOUR LANGUAGE		
A computational model of verbal understanding.....	441	<i>R. Simmons, J. Burger, R. Schwarcz W. A. Woods C. Kellogg</i>
Procedural semantics for a question-answering machine.....	457	<i>W. A. Woods</i>
Natural language compiler for on-line data management.....	473	<i>C. Kellogg</i>
PRICING COMPUTER SERVICES—WHAT? WHY? HOW?		
Prices and the allocation of computer time.....	493	<i>N. Singer, H. Kantor, A. Moore S. Smidt M. Marchand N. Nielsen</i>
The use of hard and soft money budgets and prices.....	499	<i>S. Smidt</i>
Priority pricing with application to time-shared computers.....	511	<i>M. Marchand</i>
Flexible pricing: An approach to the allocation of computer resources...	521	<i>N. Nielsen</i>
DATA STRUCTURES FOR COMPUTER GRAPHICS		
Data structures and techniques for remote computer graphics.....	533	<i>I. Cotton, F. S. Greatorex, Jr.</i>
Graphical systems communications: An associative memory approach.....	545	<i>E. H. Sibley, R. W. Taylor, D. G. Gordon D. L. Childs</i>
Description of a set-theoretic data structure.....	557	<i>D. L. Childs</i>
HYBRID SYSTEMS FOR PARTIAL DIFFERENTIAL EQUATIONS		
Applications of functional optimization techniques for the serial hybrid computer solution of partial differential equations.....	565	<i>H. H. Hara, W. J. Karplus</i>
Hybrid assumed mode solution of non-linear partial differential equations.....	575	<i>J. C. Strauss, D. J. Newman</i>
Hybrid computer integration of partial differential equations by use of an assumed sum separation of variables.....	585	<i>J. R. Ashley, T. E. Bullock</i>

A hybrid computational method for partial differential equations.....	593	<i>G. A. Coulman, J. F. Svetlik</i>
Preliminary investigation of a hybrid method for solving partial differential equations.....	601	<i>R. M. Howe, S. K. Hsu</i>
PROGRAMMING SYSTEMS I		
QUIP—A system for automatic program generation.....	611	<i>F. C. Bequaert</i>
The XPL compiler generator system.....	617	<i>W. McKeeman, J. Horning, E. Nelson, D. Wortman</i>
A syntax directed processor writing system.....	637	<i>E. Ferentzy, J. Gabura</i>
THE MINI-COMPUTER		
The Mini-computer: A programming challenge.....	649	<i>R. Hooper</i>
The Mini-computer: A new approach to computer design.....	655	<i>G. Ottaway; R. Shirk, D. Hitt</i>
EXECUTIVE SYSTEMS FOR HYBRID SIMULATION		
The Lockheed hybrid system (A giant step).....	663	<i>C. K. Bedient, L. L. Dike</i>
A priority interrupt oriented hybrid executive.....	683	<i>G. N. Soma, J. D. Crunkleton, R. E. Lord</i>
Growing pains in the evolution of hybrid executives.....	701	<i>M. D. Thompson</i>
The Boeing/Vertol hybrid executive system.....	709	<i>D. A. Willard</i>
Family I: Software for NASA Ames simulation systems.....	719	<i>J. S. Raby, E. A. Jacoby, D. E. Robinson</i>
SYSTEMS TECHNIQUES FOR INTERACTIVE GRAPHICS		
Stand-alone/remote graphic system.....	731	<i>M. D. Rapkin, O. M. Abu-Gheida</i>
An interactive graphics terminal for generation and display of 2D and 3D structured images.....	747	<i>T. G. Hagan, R. J. Nixon, L. J. Schaeffer</i>
A head mounted three dimensional display.....	757	<i>I. E. Sutherland</i>
A clipping divider.....	765	<i>I. E. Sutherland, R. F. Sproull</i>
A low cost computer graphic terminal.....	777	<i>M. Macaulay</i>
The Rand video based graphic communications system (Paper not included in this volume).....		<i>T. O. Ellis</i>
PROBLEMS IN THE IMPLEMENTATION OF INTELLIGENT ROBOTS (A Panel Session-Papers not included in this volume)		
COMPUTERS IN BIOMEDICAL RESEARCH		
Computer simulation of a non-linear blood flow model.....	787	<i>H. A. Crosby, M. K. Klukis</i>

A computer system for real-time monitoring and management of the critically ill.....	797	<i>D. Stewart, D. Erbech, H. Shubin</i>
Computer system for research and clinical application to medicine.....	809	<i>R. M. Gardner, T. A. Pryor</i>
The use of computers to improve biomedical image quality.....	817	<i>R. H. Selzer</i>
PROCESS CONTROL PROGRAMMING LANGUAGES (A Panel Session-Papers not included in this volume)		
LARGE SCALE INTEGRATION		
A computer system designer's view of large scale integration.....	825	<i>L. M. Spandorfer, M. E. Conway</i>
A high-speed modular multiplier and digital filter for LSI development.....	847	<i>D. F. Calhoun</i>
Efficient partitioning for the batch-fabricated fourth-generation computer.....	857	<i>B. Scheff, O. Lowenschuss, N. Cserhalmi</i>
Engineering for systems using large scale integration (LSI).....	867	<i>C. F. O'Donnell,</i>
MOS GP computer.....	877	<i>R. H. Booher</i>
OPERATING SYSTEMS I/OPERATING SYSTEMS II		
Hardware/software interaction on the Honeywell 8200.....	891	<i>T. Hatch, J. Geyer</i>
Measurement and analysis of large operating systems during system development.....	903	<i>D. J. Campbell, W. J. Heffner</i>
Thrashing: Its causes and prevention.....	915	<i>P. J. Denning</i>
<hr/>		
PART II		
PROGRAMMING SYSTEMS II		
WRITEACOURSE: An educational programming language.....	923	<i>E. Hunt, M. Zosel</i>
A table driven compiler for use with automatic test equipment.....	929	<i>R. L. Mattison, R. T. Mitchell</i>
On the basis for ELF: An extensible language facility.....	937	<i>T. E. Cheatham, A. Fisher, P. Jorrand</i>
MEMORY TECHNIQUES—HERE TODAY		
Associative processing for general purpose computers through the use of modified memories.....	949	<i>H. Stone</i>
Addressing patterns and memory handling algorithms.....	957	<i>S. Sisson, M. Flynn</i>
Design of a 100-nanosecond read cycle NDRO plated wire memory.....	969	<i>T. Ishidate</i>
High speed, high current word matrix using charge storage diodes for rail selection.....	981	<i>S. Waaben, P. Carmody</i>

AUTOMATED MAINTENANCE AND CHECKOUT OF HYBRID SIMULATION FACILITIES		
Automatic Checkout of a large hybrid computing system	987	<i>J. C. Richards</i>
Hybrid diagnostic techniques	997	<i>T. K. Seehuus, W. Maasberg, W. A. Harmon</i>
DYNAMIC RESOURCE ALLOCATION		
Demand paging in perspective	1011	<i>B. Randell, C. Kuehner</i>
Program behavior in a paging environment	1019	<i>B. Brawn, F. Gustavson</i>
JANUS: A flexible approach to real-time time-sharing	1033	<i>J. Kopf, P. Plauger</i>
A parallel process definition and control system	1043	<i>D. Cohen</i>
HUMAN AUGMENTATION THROUGH COMPUTERS AND TELEOPERATORS (A Panel Session—No papers included in this volume)		
LABORATORY AUTOMATION		
A computer system for automation of the analytical laboratory	1051	<i>P. J. Friedl, C. H. Sederholm, T. R. Lusebrink</i>
Real-time time-sharing, the desirability and economics	1061	<i>B. E. F. Macefield</i>
A modular on-line computer system for data acquisition and experimental control	1065	<i>H. P. Lie, R. W. Kerr, G. L. Miller, D. A. H. Robinson</i>
A standardised data highway for on-line computer applications	1077	<i>I. N. Hooton, R. C. M. Barnes</i>
Use of a computer in a molecular biology laboratory	1089	<i>J. F. W. Mallett, T. H. Gossling</i>
A small computer as an on-line multiparameter analyzer for a neutron spectrometer	1099	<i>M. G. Silk, S. B. Wright</i>
Applications of digital computers to the long term measurement of blood pressure and the management of patients in intensive care situations	1105	<i>J. L. Corbett</i>
HAND PRINTED CHARACTER RECOGNITION		
Some conclusions on the use of adaptive linear decision functions	1117	<i>E. R. Ide, C. E. Kiessling, C. J. Tunis</i>
Experiments in the recognition of hand-printed text: Part I—Character recognition	1125	<i>J. H. Munson</i>
Experiments in the recognition of hand printed text Part II—Context analysis	1139	<i>R. O. Duda, P. E. Hart</i>
The design of an OCR system for reading handwritten numerals	1151	<i>P. J. Hurley, W. S. Rohland, P. J. Traglia</i>

OPERATING SYSTEMS I/OPERATING SYSTEMS II		
The dynamic behavior of programs.....	1163	<i>I. F. Freibergs</i>
Resource allocation with interlock detection in a multi-task system.....	1169	<i>J. E. Murphy</i>
Cage dual processing.....	1177	<i>K. C. Smith</i>
An operating system for a central real-time data processing computer.....	1187	<i>P. Day, H. Krejci</i>
NEW MEMORY TECHNIQUES		
Holographic read-only memories accessed by light-emitting diodes.....	1197	<i>D.H.R. Vilkomer- son, R. S. Mezrich, D. I. Bostwick</i>
Semiconductor memory circuits and technology.....	1205	<i>W. B. Sander</i>
2½—D Core search memory.....	1213	<i>M. W. Rolund, P. A. Harding</i>
Design of a small multi-turn magnetic thin film memory.....	1219	<i>W. Simpson</i>
HYBRID SIMULATION TECHNIQUES		
An adaptive sampling system for hybrid computation.....	1225	<i>G. A. Rahe, W. Karplus</i>
A new solid state electronic iterative differential analyzer making maximum use of integrated circuits.....	1233	<i>B. K. Conant</i>
A general method for programming synchronous logic in analog computation.....	1251	<i>R. A. Moran, E. G. Gilbert</i>
APPLICATIONS OF COMPUTERS TO PROBLEMS OF THE ATMOSPHERE AND GEOPHYSICS		
Computer experiments in the global circulation of the earth's atmosphere.....	1259	<i>A. Kasahara</i>
Computational problems encountered in the study of the earth's normal modes.....	1273	<i>F. Gilbert, G. Backus</i>
PROGRESS IN DISPLAYS (A Panel Session—No papers in this volume)		
COMPUTER GENERATED PICTURES—PERILS, PLEASURES, PROFITS		
Computer animation and the fourth dimension.....	1279	<i>A. M. Noll</i>
Computer displays in the teaching of physics.....	1285	<i>J. L. Schwartz, E. E. Taylor</i>
Art, computers and mathematics.....	1292	<i>C. Csuri, J. Shaffer</i>
CAMP—Computer assisted movie production.....	1299	<i>J. Whitney, J. Citron</i>
What good is a baby?.....	1307	<i>N. Winkless, P. Honore</i>
A computer animation movie language.....	1317	<i>D. Weiner, S. E. Anderson</i>
NEW TRENDS IN PROGRAMMING LANGUAGES		
CABAL—Environmental design of a compiler-compiler.....	1321	<i>R. K. Dove</i>
Cage test language—An interpretive language designed for aerospace....	1329	<i>G. S. Metsker</i>
An efficient system for user extendible languages.....	1339	<i>M. C. Newey</i>

Program composition and editing with an on-line display.....	1349	<i>H. Bratman, H. G. Martin, E. C. Perstein</i>
BULK MEMORY DEVICES		
New horizons for magnetic bulk storage devices.....	1361	<i>F. D. Risko</i>
Laser recording unit for high density permanent digital data storage....	1369	<i>K. McFarland, M. Hashiguchi</i>
A magnetic random access terabit magnetic memory.....	1381	<i>S. Damron, J. Miller, E. Salbu, M. Wildman, J. Lucas</i>
Diagnostics and recovery programs for the IBM 1360 photo-digital storage system.....	1389	<i>D. P. Gustlin, D. D. Prentice</i>
SIMULATION IN THE DESIGN AND EVALUATION OF DIGITAL COMPUTER SYSTEMS		
Simulation design of a multiprocessing system.....	1399	<i>R. A. Merikallio, F. C. Holland</i>
A simulation study of resource management in a time-sharing system...	1411	<i>S. L. Rehmman, S. G. Gangwere, Jr.</i>
Performance of a simulated multiprogramming system.....	1431	<i>M. M. Lehman, J. L. Rosenfeld</i>
THE COMPUTER FIELD: WHAT WAS PROMISED, WHAT WE HAVE, WHAT WE NEED (HARDWARE SESSION)		
Hardware design reflecting software requirements.....	1443	<i>S. Rosen</i>
What was promised, what we have and what is being promised in character recognition.....	1451	<i>A. W. Holt</i>
High speed logic and memory: Past, present and future.....	1459	<i>A. W. Lo</i>
REAL-TIME INFORMATION SYSTEMS AND THE PUBLIC INTEREST		
Real-time systems and public information.....	1467	<i>C. W. Churchman</i>
National and international information networks in science and technology.....	1469	<i>H. Borko</i>
Real-time computer communications and the public interest.....	1473	<i>M. M. Gold, L. L. Selwyn</i>
Toward education in real-time.....	1479	<i>P. E. Rosove</i>
A public philosophy for real-time information systems..	1491	<i>H. Sackman</i>
COMPUTER DESIGN AUTOMATION: WHAT NOW AND WHAT NEXT?		
Introduction.....	1499	<i>J. M. Kurtzberg</i>
Functional design and evaluation.....	1500	<i>D. F. Gorman</i>
Interface between logic and hardware.....	1501	<i>R. L. Russo</i>
Hardware implementation.....	1502	<i>W. E. Donath</i>
Hardware fault detection.....	1502	<i>M. A. Breuer</i>

The Pitt time-sharing system for the IBM system 360: Two year's experience

by GEORGE F. BADGER, JR., E. ANDREW JOHNSON
and RICHARD W. PHILIPS*

University of Pittsburgh
Pittsburgh, Pennsylvania

INTRODUCTION

Overview of the system

The University of Pittsburgh has developed a console-based Time-Sharing System,¹ and has had it in service since March of 1966. This paper is to serve as a report on the utilization of such a system and on certain conclusions we have come to regarding Time-Sharing Systems.

First, we would like to describe the services available under this system. The services range from a highly conversational and interactive language called the Pitt Interpretive Language (PIL) to languages on which no substantial advantage is gained by use of a console. PIL is fully interpretive and has placed heavy emphasis on making things easy for the user. A great deal of care is placed on giving meaningful diagnostics and easy error correction. Decisions in the initial implementation between coding efficiency and ease of use were always made in favor of the user. In a second version of the interpreter some emphasis is being put on efficiency while retaining the error handling facility. The ability to service a number of programs such as PIL, with the required speed of interaction, is our justification for calling this a Time-Sharing System.

The second class of services are those which are partially interactive or in which the user gains some advantage by working from a console. These include an OS compatible Assembly Language,

and a portion of the OS macro compiler capability, a FORTRAN IV compiler with a pre-editor, a conversational context editor, and a file maintenance package. Again in the case of the shared file system, great care was exercised in preserving ease of use.

During his use of the system the user may take advantage of any or all of the facilities. He may switch processors as often as desired, and may intermix certain command language statements while working within a processor, e.g., list part of a file while preparing a program which will work with it.

This entire set of services runs on an IBM System/360 Model 50, utilizing a 2361 large capacity storage (LCS) unit. The configuration and some idea of the storage utilization are presented in Figure 1. The most important components of this system are the large capacity storage and the 2314 disk system. The system runs with the user storage being contained in LCS, both while the user is idle and while he is in execution.² The high speed storage of the machine is utilized for one processor and the operating system as can be seen from Figure 1. The primary processor resident in high speed storage is the interactive language processor. Because input to the assembler forms a large part of the service of the system, the assembler also is resident, but in LCS. Assembly is an I/O bound process compared to PIL which is compute bound.

*Currently with White, Weld & Company

¹This system was completely written by Computer Center Staff and has no dependencies on other operating systems. Parts of the system are compatible with OS/360, and several language processors are fully compatible.

²When necessary, the system will resort to memory swapping onto an IBM 2314 disk. This happens with approximately 30 users signed on, otherwise memory is allocated in contiguous blocks from a free pool when the user signs on.

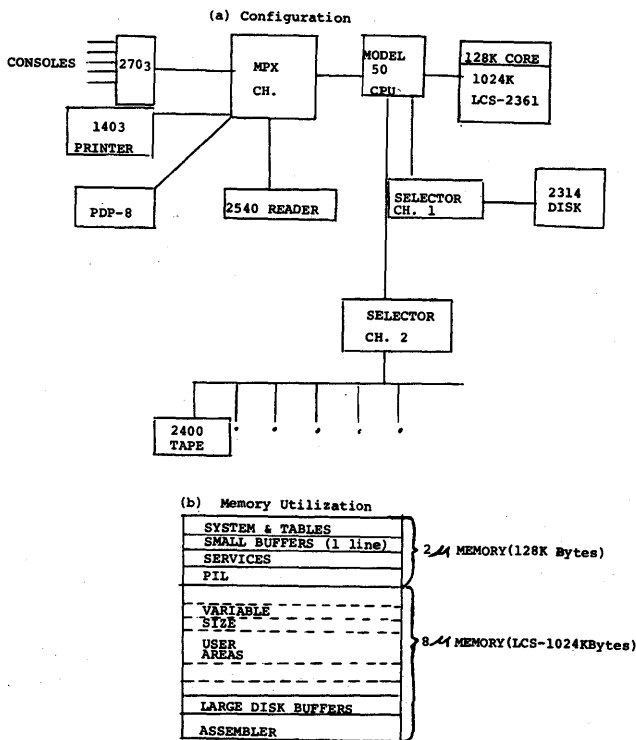


FIGURE 1

The system services something identified as a user who consists of a principal input file and a principal output file. Of the 32 users on the system, as of April, 1968, 31 have a remote console for these principal files. We currently support IBM 2741 and 1050, Teletype 33 or 35, Friden 7100 or 7102, and Sanders 720 scopes.

Because of the structure of the system, it is also possible to introduce one or several pseudo-batch background users whose principal input and output files could be any devices on the system. As of April, 1968, one such user runs in the system from a 2540 card reader to a 1403 printer. The services available to this batch user are identical to those available from the remote console. The only difference in his treatment is that a fatal error results in going to the next job rather than allowing further input which, in the case of a console, could be attempts at correction.

The system favors those who are working in a conversational mode, and has a simple queueing system to allow very rapid response to minimal requests. Because of the emphasis on interaction, the completion of any I/O results in the user being dispatched to a high priority queue which will get service as quickly as possible. Upon being put

into this queue, the user will follow all those who arrived in the queue beforehand, and then receives one-quarter second of central processing time. During the one-quarter second, he may either run out of things to do (for example, because of issuing requests for further I/O and waiting for their completion), or he may use the entire one-quarter second. In the latter case, he goes into a lower priority queue which receives service on a time available basis. The quantum of time given in this lower priority queue is increased to two seconds. This then is a general description of the system and its services.

Patterns of system utilization

The use of the system has increased very rapidly during the two years that the system has been in use. As of April, 1968, we run approximately 5000 console sessions per month, and approximately 3000 of the batch jobs. The console sessions use all of the services, but the batch user typically restricts himself to the use of either the Assembly Language or the Fortran Processor. In the conclusions of this paper we will make some comment on the choice by the user as to whether he runs in the interactive or pseudo-batch mode.

As far as the time utilization within the system, PIL accounts for approximately 23% of the Central Processor time available. That is, during a ten-hour day, $2\frac{1}{2}$ hours are actually devoted to the execution of PIL programs.³ User execution of other processors and generated code accounts for an additional 20%. The processing of supervisor calls and general system overhead accounts for 15%, and the Central Processor is in the idle state 17% of the time. The servicing of I/O interrupts, and disk unblocking with data transfer, accounts for the remaining 24%.^{4,5} (The CPU is busy 83% of the time.)

Within this pattern of utilization we are interested in the demands put upon the system in

³PIL, being fully interpretive, is very slow in execution. On a matrix inversion, for example, a PIL program will execute at $1/75$ the speed of a Fortran program excluding its compilation. A second implementation of PIL will significantly modify this ratio.

⁴All statistics in this paper are representative, but do not necessarily hold for long periods, e.g., a month.

⁵Because of the extremely heavy dependence of the System on the IBM 2314 disk with an associated file system, the CPU is devoted to blocking variable length records. This is explained later in the paper.

more basic units. In particular, there is the question of how much CPU time is required as a time slice, since this partially determines the ability of a given system to support conversational users. Table 2 and Graph 1 present this data in some detail.

These figures show an average time slot of $(20 \pm 5)/300$ seconds. The data, however, do not show the reason for termination of user execution. Since the user is primarily interested in being terminated because his work is complete, more important data are presented in Table 3. The data present the system as a finite Markov process⁶ with the transition probabilities for the set of all users (the system) being given.

The data presented in Tables 3 and 4 represent a composite of two periods, one relatively light and the other extremely busy.⁷ Values with the superscript (1) varied considerably with the probability of going from user execution to SVC varying from .7 to .3, the probability to I/O from .3 to .7, and the average time spent in user execution taking the values 3.89 and .65 respectively. All other values of interest remain relatively stable.

Since the user is frequently taken out of execution due to an I/O interrupt, we are currently attempting to study the impact of masking these interrupts for a short period of time (approximately 25/300 seconds—the level at which Graph 1 flattens out). By doing this, approximately 95% of all requests for service would be serviced in a single slice and without interruption. This should preserve the response time of the system for short requests, as well as utilizing peripherals effectively. This would make response time only slightly dependent on the system loading. On the other hand, it might allow quicker preparation of work which will finally require computation of longer duration.

We could consider the strategy above as minimizing the maximum response time for a request up to some limited size. Since a stable level of response time appears to be at least as important as fast response, we are also considering the re-

verse. That is, never respond in a period less than the period the system is capable of sustaining. Such a level, if it fell between one and two seconds would give very good performance from the conversational users point of view. Further, it would provide increased flexibility in choice of scheduling. This last might be of great importance if a swapping or paging mechanism were in use since you could reduce channel activity by choosing opportune times for the transfer.

The data presented above are representative but are not system parameters. They are intended only to give a picture of the relative activities within a time-shared environment.

System states for Tables 1-4

User execution—The system is running a user program including use of compilers, assemblers or interpreters or the file system. PIL execution plus other user executions.

SVC—A request has been made for supervisor services via the SVC instruction—mainly I/O requests.

I/O—An I/O interrupt is being processed.

Overhead—A clock trap is being processed, the scheduler is running, or certain overhead routines are in use (queueing etc.).

Idle—The CPU is in wait status, i.e., not running.

TABLE 1—CPU Utilization

STATE	PERCENT OF TIME
PIL Execution	23
Other User Exec.	20
SVC Processing	5
I/O Processing	24
System Overhead	10
Idle (CPU waiting)	17

Some conclusions

The utilization of the LCS⁸ unit has proved to be extremely successful on the Model 50. Because of the overlapping of the rewrite cycle on core with Central Processor execution, many programs run almost as fast out of LCS as they did in Cen-

⁶Treating this as a Markov chain, finding the steady state vector, and using this to predict percentages of time spent in the various states yields surprisingly accurate results. This will be studied further in a later paper.

⁷The system seems to be representable as a Markov chain, i.e., the transition matrix is constant for extended periods of time. It does seem to vary over time, but the variation is basically related to level of activity, or time in the idle state.

⁸LCS is an 8 μ memory with a 4 byte fetch when used by a Model 50. In all other respects it is identical to main memory.

TABLE 2—Frequency of time slots actually given prior to seeking another user to execute

Time in 1/60 Second	Percent of Slots Exactly This Size	Percent of Slots This Size or Smaller
1 or less	48	48
2	20	68
3	6	74
4	3	77
5	2	79
6 to 14	8	87
15	10	97
16 to 120	3	100

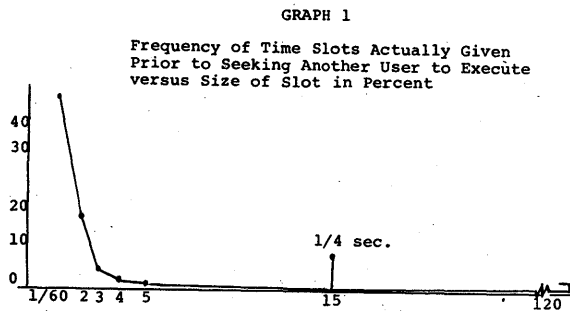


TABLE 3—Transition probabilities of the system states

	Next State				
	User Exec	SVC	I/O	Over-head	Idle
User Executing	(1)	(1)			
SVC	0	.32	.66	.01	0
Current I/O	.45	.45	0	.10	0
State Overhead	.90	0	0	.10	0
Idle	.82	.0	0	.10	.08
	0	0	95	0.1	.04

TABLE 4—Average time spent in current state before moving to next state in 1/300 second

	Next State					
	User Exec	SVC	I/O	Over-head	Idle	Mean
User Executing	(1)	(1)				
Current SVC	0	1.77	1.17	49.29	0	1.49
State I/O	.75	3.08	0	.91	0	1.81
Overhead	.92	0	0	1.25	0	97
Idle	.20	0	0	1.0	1.54	.38
	0	0	14.65	237.39	8.73	15.74

tral Memory. In the case of PIL, the interpreter itself is resident in the main storage while its data, or the text of the user's program, is resident in LCS. Timing tests have indicated that while there is some degradation of PIL's speed due to this (20% or less), it probably is not significant when compared to either swap or batch mechanisms for other Time-Sharing Systems.⁹ While the LCS proves to be very economical on the Model 50, its effectiveness would be somewhat reduced on a Model 65, and probably greatly reduced on a Model 75 because of their greatly increased internal speeds. If we were to move the system to a Model 75, it is quite likely that rather than executing out of LCS, we would use the LCS as a swap device analogous to a drum. One of the reasons that the LCS has proved very effective is that many of the requests within a Time-Sharing System are for very minimal amounts of Central Processor time. Of all the times when a user is put into execution, the majority of these times he has requested execution for the purpose of entering a single line to a processor such as the assembler or the interpreter. These requests can be serviced in a matter of several thousandths of a second, and the relative lack of speed on the large capacity storage has no significant effect on performance.

The second area in which we have drawn some conclusions with regard to the Time-Sharing System is that the file system is probably the most important single part of the system. This is true both in the sense of being a major concern of users, and of being a major potential bottleneck if handled poorly. Consistent with data presented in earlier papers by other authors, each request for service by a user generates a demand for several thousand characters of data to be transmitted from the disk system.¹⁰ Our early implementation of the file system was a very straightforward, unblocked and unbuffered tape simulation. The file system was sequentially structured. It soon became evident that the pattern of requests by users was such that a tremendous amount of head movement was necessary for this unblocked

⁹Considerable work has been done on the use of LCS as the swap/paging mechanism, and this provides great promise for future development. It is currently not possible to page on the Model 50, and this limits the value of this alternative.

¹⁰This is user owned data and does not include any "overhead" transmission.

mechanism. To improve the performance we have come to quite a different concept on the disk system. The basic unit of physical activity on the disk system is now the reading or writing of 2,048 character records. The logical records, as seen by the user, are all treated as being variable length and are blocked automatically by the system. Here also LCS is of great value since 50-100K of buffers make system performance more efficient.

A descriptor, providing a record of the blocking, is associated with each 2,048 byte record and the system can pull these records apart very rapidly upon request by the users. On a single physical read or write, the user now makes available to himself 25 card images, for example, thereby causing 1/25th of the number of seeks and interrupts to be processed. This has had a tremendous impact on the performance of the system both in the amount of time that the system has kept idle because of having to access disk records, and the amount of elapsed time while a user is doing something such as assembling, loading or compiling. The improvement was on the order of 30 to 1. Given a high speed file system such as this, and a very easy mechanism for using it, the users rapidly learn to take advantage of the facility. This is important not only to the user whose time is saved, but it is also extremely important to the system since it can accomplish much more working from disks than it could working from other devices. Further, it expands the size of jobs which are feasible under the system. Also, it greatly decreases the probability that an I/O request will require physical activity, entailing possible changing of users which may also require swapping or paging. It will also reduce the number of interrupts.

A third area in which we have drawn some conclusions regards the integrity of performance of a Time-Sharing System. We have learned over the course of two years, not always in a painless manner, that the user of a Time-Sharing System is quite different in his demands than the user of any other system. He expects the system to be on when it is scheduled to be on, and to perform in a rather stable manner while it is on. In particular, he expects the file system to perform perfectly at all times so that any work he does will never have to be redone. The user of such a system typically has a scheduled time at which he and the system are to work together and if the system is not available at the time, he cares very little why

or what kind of excuses you can present. The user is typically quite reasonable if you give him adequate warning that the system will not be available at any given time.

Maintaining the software programs of such a system is of considerable difficulty and should not be further complicated by the introduction of machine performance problems. The area in which we have incurred the most difficulty is that manufacturers do not understand systems which are to run reliably for extended periods of time. There seems to be no comfortable middle ground between extreme safety built into systems such as the FAA and other military systems, and those systems of the second generation which ran adequately for a batch processing environment. This lack of understanding is found on many levels but perhaps most significant is the lack of any facility for informing the user when malfunctions have occurred and the system is off the air. In general, there are many small rough edges in present equipment. An area of difficulty is that some part of the configuration usually is in marginal condition. The installation does not wish to terminate service completely for an extended period. The maintenance of the machine typically requires this. The machine normally runs in a partially crippled condition or not at all. An overall change in the maintenance methodology must occur soon.

A fourth area of experience relates to the use of the console system versus the use of its batch equivalent. Since the user has accessible identical facilities under each environment, we could expect him to move to that manner of operation which performed best for his job. This experience must be regarded carefully since the batch system often requires a long walk whereas, console service is available locally.

Clearly, in the area of interactive languages with emphasis on conversation, you would expect the user to remain on the console and this has been our experience. It is very unusual to find any user submitting a batch job which is a PIL or context editor run. One other use of batch is the creation of tape and disk files for use from the console.

The Assembler and Fortran are not overly conversational and do not take great advantage of the fact that the user is available. In the case of Fortran, the work is about equally divided between jobs submitted under the batch system, and jobs executed from the console. The Assembler, which

gets its main use by students in beginning programming courses, is probably used more under the batch system than under the console system.¹¹ After programs are running in either of these languages, the user typically will leave his object program on disk and will revert back to using the console for execution or for final debugging. It should be noted here that in addition to the Time-Sharing System on the Model 50, the Center makes available a batch processing system on an IBM 7090. There seems to be a rather clear dividing line rapidly discovered by the users between the jobs which should be done on the 7090, and jobs which should be done on the Model 50. The two machines are of approximately the same capabilities as far as speed and storage space are concerned. Those jobs which are heavily compute bound remain on the 7090 as do any jobs which gain little or nothing by interaction.

Because of the lack of success of other installations in using a Time-Sharing System for all of their computing and because of our own success with the two independent systems, we plan to provide a 360 batch system (OS/360) on a second Model 50 to maintain systems which are fitted to the differing demands by our users. Running the two facilities in parallel should give a clearer picture in the future of the relative advantages of time-sharing and batch processing.

Future directions

The system is still under development and we are continuing to add facilities if we find them useful as manpower permits. We are in the process of adding other processors including Algol, PL/1 and further extensions of Fortran. We also plan to implement some kind of a random access

¹¹This is at least partly due to the fact that some understanding is required to be able to effectively debug machine language at a console.

and index sequential file structure although it is not clear that these will be implemented within the shared file structure we currently use.

Another useful service that we plan to implement is the ability to submit jobs to either Model 50 system from the remote consoles. In order to implement this, we will put a channel-to-channel attachment between the two Model 50's as well as making parts of the 2314 disk available to both systems. How far we will push this effort is undetermined at present. Finally, in an effort to reduce the number of interrupts presented to the system, and in order to maintain effective utilization of unit record equipment, we are putting together a spooling package for the printer, punch and card reader. This will allow blocking to be done by the channel program with one interrupt per block. These spoolers will work in such a way as to present a job stream which runs from one disk file to another in the same manner as the batch job stream currently runs from the card reader to the printer.

BIBLIOGRAPHY

- 1 *The Pitt time-sharing system for the IBM system/360*
Computer Center University of Pittsburgh March 1968
- 2 ————*The Pitt interpretive language for the IBM System/360*
March 1968
- 3 D W FIFE
An optimization model for time-sharing
Proceedings SJCC 1966 pp 97-104
- 4 J D FOLEY
A Markovian model of the University of Michigan executive system
CACM Vol 10 No 9 September 1967
- 5 T J O'CONNOR
Analysis of a time-sharing system
Stanford University Unpublished PhD Dissertation
- 6 E PARZEN
Stochastic processes
Holden-Day San Francisco 1965 Chapter 6
- 7 A L SCHERR
An analysis of time-shared computer systems
The MIT Press 1967

Debugging in a time-sharing environment

by WILLIAM A. BERNSTEIN and JAMES T. OWENS

International Business Machines Corporation
Kingston, New York

INTRODUCTION

In the past 10 years, rapid advances have been made in computer engineering and programming. Among the significant achievements have been the development of more powerful computers, multiprocessing systems, high-level language compilers, multiprogramming operating systems, and time-sharing systems.

But there is one area in programming in which progress has been comparatively slow. This is the area of program debugging aids. The problem of finding and correcting program bugs remains perplexing and costly.

When computers were used to run stand-alone programs or to run programs under small operating systems, conventional debugging tools — such as console key procedures and system-provided dump, trace, and status information — met most debugging needs. But, as programming systems have become more complex, existing debugging tools have become barely adequate for tracking down and correcting program bugs.

With the introduction of time-sharing systems, the conventional tools have become almost worthless. This has forced a reappraisal of debugging procedures. It has become apparent that a new type of debugging tool is needed to handle the special problems created by a time-sharing system. These special problems result from the dynamic character of a time-sharing system — a system in which the program environment is continually changing, a system in which a user is unaware of the actions of other users, a system in which program segments are rolled in and out of storage locations, and a system in which one copy of code can be shared by many users. To debug in this dynamic environment, the programmer needs a *debugging support system* — a set of debugging programs that operate largely independently from the operating system they service.

This paper examines conventional debugging tools and appraises their adequacy in view of programming advances. It then discusses the characteristics of a support system that meets the debugging and program mod-

ification needs of a time-sharing system. This support system can be an important vehicle for advancing debugging technology.

Current debugging tools

Two kinds of debugging tools have been available to programmers running stand-alone programs or running programs under an operating system; *machine-level tools* and *system-level tools*.

Machine-level tools consist of procedures available at the system operator's console. These include such items as the stop key, start key, instruction step, address stop, and displaying and altering the machine state.

System-level tools are debugging tools provided as part of the programming system itself — including facilities to produce such things as storage dumps, traces, and system error recovery data.

These tools could be used to obtain the information needed to start debugging. To be useful, this information must include:

- The hardware configuration at the time the error occurred.
- The version of the operating system that was being used.
- The input parameters for the program.
- The condition that indicated the error (stop, loop, erroneous output, etc.).

With this information, the programmer begins debugging. He is likely to follow the four classical steps in the debugging procedure:

1. Attempt to duplicate the failing condition so he can obtain status information at the point of failure.
2. Initiate system-level debugging procedures, including acquisition of storage maps and collection of intermediate results from internal tables and/or data areas.

3. Initiate machine-level debugging procedures to isolate the cause of the error. This step is necessary when a peculiar condition (such as a timing dependency) produced the error but was not reflected in the system-level debugging data.
4. Correct the error.

To summarize, the solution of a program bug requires: (1) a complete report of the conditions under which the error occurred, (2) use of system-level debugging tools, and (3) if necessary, use of machine-level debugging tools.

Evolution of debugging tools

Each step forward in computer operations has produced the need for new and better debugging tools. This has been evident in the transition from stand-alone execution to operating system execution, and now in the transition to time-sharing execution.

Before the existence of operating systems, the programmer was concerned with a single program with which he was totally familiar. In most cases, he could debug that program with machine-level tools. But the process was time-consuming and many programmers started to build hooks into their programs to obtain intermediate results for later analysis. This marked the beginning of system-level debugging tools.

The introduction of operating systems — and especially multiprogramming systems — has forced programmers to be more reliant on system-level debugging tools. The use of console debugging has been discouraged and in many cases, it has been prohibited entirely. Without access to the console, programmers learned to rely on debugging aids built into the operating system.

Today, the operating system programmer assigned to the debugging task must depend upon a fixed set of system-level debugging tools. These tools and their output are predefined — often consisting of a dump in a prescribed format. The tools lack flexibility, and they provide only limited information.

These built-in tools have a major drawback. As an operating system grows, system-level tools must constantly be updated (or new tools must be created) to serve the new functions being added to the system. It is apparent that the growth of operating systems has already obsoleted the once effective and efficient debugging tools originally designed for their maintenance. The addition of more tools merely forestalls the inevitable — the appearance of one more bug, one more unanticipated situation.

It becomes necessary, therefore, to turn to a new type of debugging tool — a debugging support system. Such a system should be part of, but function separately from, the operating system it services. The support system must be flexible enough to cover the widest range of

potential debugging and maintenance needs of the “parent system,” yet it must be simple enough in structure to be bug-free within itself.

Need for a debugging support system

Impetus for the development of a debugging support system has resulted primarily from the introduction of time-sharing systems. Several major considerations have fostered design of such a system.

First, the three most important pieces of information required to start debugging (hardware configuration, description of input parameters, and description of output condition) are not available in a time-sharing system. Note the following:

- *Hardware Configuration:* Because there are many users of a time-sharing system, it is impossible for an installation to report the number of devices, central processing units, storage elements, channels, etc., that were in use at the moment the problem occurred.
- *Description of Input Parameters:* In an extreme case (such as an error in a shared program), it is important to know the input parameters introduced by all users of the system. Depending on the number of users the chance of obtaining this information ranges from the remotely possible to the literally impossible.
- *Description of Output Condition:* A description of the state of the machine at the moment the problem occurred is needed. If a user does not know what caused a bug, the contents of all storage devices for all users must be printed. Using the fastest available printing device, this action is prohibitively time-consuming and cumbersome.

A second major consideration is the need for an “on-site” debugging tool for each terminal user in a time-sharing system. When a bug is detected, a time-sharing user does not want to transmit the problem to a central location and await its solution. The tool must be immediately available for debugging at the moment and under the conditions in which the problem occurs.

Another major consideration is the need for a “conversational” tool that permits the terminal user to make repeated attempts to locate and correct his bug. If the user fails in his first attempt to locate the bug, he can try again and again and again without interfering with other users of the system.

And finally, there is a need for total flexibility. The debugging support system should provide generalized functions that the user can then employ as necessary. This circumvents the problem of predefined tools and gives the user a high degree of flexibility in defining his own debugging procedures. Provision of generalized

functions also increases the life-span of the support system by making it unnecessary to add to the support system as the parent system grows.

Definition of a time-sharing support system.

Since the authors' experience has been in writing general purpose control programs, assemblers, compilers, input/output supervisors, and similar programs, the term "system program" has taken on a particular meaning. In this discussion, the term "system program" is used to designate those programs not functioning directly to solve a specific data processing problem (e.g., data reduction, payroll processing, model simulation, etc.) but rather those general-purpose programs that perform generalized functions for many users (e.g., a supervisor, a program loader, a compiler, etc.).

The support system described in this paper need not, however, be limited to this type of system program. The function of the support system requires that it be at least relatively independent of (i.e., not utilize) those programs being supported. An axiom of the concept of support systems is that all functions upon which the support system depends must work perfectly. It is, therefore, technically simpler to construct a support system for "non-system" programs since the functions of the parent system (e.g., the I/O supervisor, the program loader) are readily available for use..

The terms "debug" and "bug" are and will be widely used among programmers and in other development environments. In this discussion, the term "bug" is used to mean any programming failure due to faulty logic, clerical errors, or timing considerations. It does *not* refer to failures of hardware.

By now it is apparent that the debugging tools that served us so well in the past are no longer adequate. They must be replaced by a meaningful debugging system — a system that can be used to solve the perplexing problems posed by the time-sharing environment.

It is feasible to produce that system today. The talent, experience, and knowledge needed to create a highly sophisticated on-line debugging system are available.

What is needed for time-sharing is a debugging support system that meets the following requirements:

- The system should permit a system programmer at a user terminal to debug system programs associated with his task. When used in this manner, the support system should operate in a time-sliced mode.
- When used to debug a separate task, the support system should provide the facility to modify a system program in relation to that task, without

affecting the program as executed in relation to other tasks.

- When a system program bug cannot be located and repaired from a user terminal, the support system should permit a skilled system programmer at a central location to suspend time-sharing activity until the error is located and repaired. The support system should then permit time-sharing activity to be resumed as though there had been no interruption.
- The support system should permit a system programmer to monitor the progress of any task from a remote terminal or from the user's terminal.
- The support system should contain the facility to remain dormant until activated by a specified condition. When activated by the condition, the system should be able to gather specified data automatically and then permit processing to continue.
- In its dormant state, the support system should not impact the performance of the parent time-sharing system.
- The support system should use a minimum of main storage and reside primarily on high-speed external storage.
- The support system should be completely independent of the time-sharing system (that is, it must use none of the facilities of the parent system), and it must be simple enough to eliminate any requirement for a support system of its own.

An effort is currently under way to produce a time-sharing support system that meets these requirements. The next section describes the characteristics of that system.

Description of a time-sharing support system.

This section describes a time-sharing support system in very general terms, concentrating more on the user interface than either the parent system or the specific implementation. While the discussion is general and applicable to the time-sharing system concept, it should be noted that the authors' experience has been with only one time-sharing system and is limited in scope.

The support system consists of a command language, a set of verb-processing programs, and supporting elements (such as an independent input/output supervisor).

The support system is interrupt-driven as are all time-sharing systems. The primary driving force is supplied by an input/output interruption that originates at a system programmer's terminal, where the system

programmer directs the debugging activity by using a conversational command language.

Any time-sharing system consists of a set of programs that dynamically reconfigure themselves to fit the demands of a variable set of terminal users. No other programming system has presented the data-capturing problems posed by a time-sharing system. The design of a debugging support system must, therefore, approach the data-capturing problem with new techniques. These techniques must be oriented to and operate with the total time-sharing system.

The inclusion of the support system requires very few additions, extensions, or changes to the basic time-sharing program. The most important change is in the technique of handling interruptions from input/output channels and passing them to the processing programs. When the support system is dormant, the time-sharing system stacks all interruptions and processes them immediately. But, when the support system is active, the support system interruptions are filtered out of the interrupt stream and passed to the support system for immediate processing. The parent system interruptions are stacked, but they are not serviced until the system programmer allows production processing to resume.

The support system consists of two major components — a Supervisor Support System which interfaces with the Supervisor of the time-sharing system and Problem Program Support System which interfaces with those portions of the system available to the user (system services, shared code, “virtual storage,” compilers, assemblers, etc.).

The “user” of the support system can be either of the following:

- *A Master System Programmer (MSP)*: This is a system-oriented user at a single terminal assigned to the Supervisor Support System. He has the privilege of examining or altering information or setting up controls anywhere in the system. This gives him the power to alter control programs that perform services for all tasks in the system.
- *A Task System Programmer (TSP)*: This is a system-oriented user at a terminal associated with a task currently active in the time-sharing system. Task debugging operations can be performed concurrently from more than one terminal. The debugging operations at each terminal are under control of the Problem Program Support System. The task system programmer may perform actions that affect only the task with which he is identified.

NOTE: In the remainder of this paper, the term system programmer is used to refer to both the master system programmer and the task system programmer.

The support system is designed to aid a system programmer in finding the cause of a failing system component. It is intended to help uncover logical flaws or clerical errors in the system software. Thus, its function goes far beyond that of a built-in error correction routine (such as an input/output error routine) that is used merely to correct a transient failure.

The system programmer uses a terminal command language to control the support system. The commands enable the system programmer to perform a variety of functions. They allow him to check on the progress of the time-sharing system, to modify the parent system during its execution, to capture static data from a fluid system for dump or display, and to localize a system programming problem in the parent system. The commands provide a set of generalized functions that the system programmer can use to perform a wide range of debugging procedures.

More specifically, the system programmer has the ability to do the following:

- Display data fields and instruction locations anywhere within the system.
- Display contents of machine registers.
- Modify variables within the system.
- Specify instruction locations within the parent system at which execution is to be stopped. The user can then intervene to alter or display data before time-sharing execution is resumed.
- Specify locations within the parent system at which data displays, alter actions, or dumps are to be executed automatically.
- Establish logical expressions to control continuation or termination of execution of a support system command string.

The most difficult of all program bugs to analyze are timing errors. This difficulty arises primarily because the exact conditions causing the error are not known or are not reproducible. The injection of a support system activity into an operating system, wherein multiple events are occurring simultaneously, will obviously alter the time relationships. The SCOPE command which would attempt to minimize this alteration is discussed under “*Future prospects for the support system.*”

By its very nature, the support system is limited to the conversational mode. Interpretation and execution of commands entered by the system programmer are performed in a manner similar to that of an interpretive compiler. First, the syntax and symbols in a command are checked. Errors are reported immediately to the system programmer’s terminal, along with the appropriate diagnostic messages to guide him in correcting the command.

Commands may be executed in immediate or delayed

modes. In the immediate mode, each command is edited and executed as it is encountered in the input command string. That string may consist of single or multiple commands.

When the system programmer specifies delayed mode, the entire command string is saved to be executed subsequently. In this mode, the system programmer designates an event that is to trigger that execution. The event is usually arrival of processor control at a given machine instruction.

The support system vocabulary contains most of our old functional friends from machine-level debugging—DISPLAY, PATCH, STOP, RUN, etc. The DISPLAY and DUMP commands specify output of data from main storage or external storage. The only difference between the commands is the destination of the output data. DISPLAY writes the data at the system programmer's terminal, while DUMP writes the data to an output device specified by the system programmer. The data format is the same in both cases, although DUMP implies printing of certain control data such as machine registers, status information, or other general data areas.

It is frequently necessary for the system programmer to suspend the parent system processing and cause it to be resumed on command. The STOP and RUN commands provide these facilities. The STOP command indicates that processing in the parent system is to be halted. The RUN command causes processing to be resumed from the point it was suspended — or from some other point. Note that a STOP command from a task system programmer means “stop my task,” while a STOP command from a master system programmer means “stop the system.” The RUN command, of course, means the inverse of STOP in each case.

In a debugging session, the system programmer frequently wants to gather system data dynamically. For example, he may want to save the status of the last 100 input/output interruptions. To do this, the system programmer uses the COLLECT command, specifying the particular event that is to trigger collection of the data. (For example, specified data could be collected each time processor control was passed to the first instruction of an input/output interruption processing routine.) Note that, while the COLLECT command is primarily a debugging tool for a failing system, it can also be used to save performance data or other surveillance data in a non-failing environment.

No matter how much a system programmer knows about a failing system, that knowledge is useless unless the instructions and data of the time-sharing system can be altered. We used to call this a “patch” function. In the support system, the PATCH function has been extended to include external direct-access storage devices as well as the conventional elements of main storage and

registers. Obviously, in a paging environment, the system programmer will want to PATCH the “official” resident copy of a program in addition to the copies currently being used.

In a time-sharing system, terms such as “storage” take on different meanings according to the current state of the system. A particular segment of code may be in one or another area of main storage, or it may be on an external storage device, depending on the particular point in execution of the program. Because of this, the system programmer must be allowed to qualify commands to specify the exact item or the exact location of the item in which he is interested. The QUALIFY command provides this facility. It allows specification of real or virtual storage, task identification, external devices, program modules, etc. The system programmer can find out the location or status of a particular item, and then use the QUALIFY command to pinpoint that item.

An IF clause is included in the support system command language to permit the system programmer to specify the condition (or conditions) under which action is to be taken. Through use of the IF clause, the system programmer can cause a statement of logical, arithmetic, and relational expressions to be evaluated to determine whether the remainder of a command string is to be executed.

The system programmer uses the DISCONNECT command to end his use of the support system. He cannot use the support system again until he re-establishes that communication link.

Two commands, DEFINE and SET, give the system programmer a quasi-programming ability in the command language when combined with the IF clause. The system programmer can DEFINE certain private symbols, manipulate them with the SET command, and make decisions based on their values.

If the system programmer were required to enter each statement when he wanted it to be executed, his ability to use the system would be severely restricted. Therefore, he is given the AT phrase to designate an event upon which a given command string is to be executed. The AT phrase empowers the system programmer to perform all the functions of the support system “on the fly.” This facility endows the language with a power only alluded to in prior systems but absolutely necessary to capture relevant data in a time-sharing system.

Just as the system programmer can establish controls using the AT phrase and alter data using the PATCH command, he is also allowed to negate those commands. To accomplish this, he uses the REMOVE command. This command erases the appropriate controls or restores the original data that have been patched.

Finally, the system programmer can invoke a pre-defined set of commands by using the CALL command.

This command makes it possible to define a standard debugging procedure, and then have that procedure executed in response to the CALL command.

From this general beginning, the system programmer can proceed to specific debugging steps guided by his experience and the support system output. The impact of CALL is to extend the language to an "automatic" level quite beyond the concepts of current debugging systems.

The commands described in the preceding paragraphs are summarized in Table 1.

TABLE 1—Summary of support system command functions

Command	Function
DISPLAY	To display specified values at the terminal being used by the system programmer.
DUMP	To dump specified values on a printer or equivalent output device specified by system programmer.
SET	To change the value associated with a symbol.
COLLECT	To collect specified values into program area.
PATCH	To change the value associated with a symbol, to save the replaced value, and to record the patching action.
STOP	To halt operation of the parent system or operation of a specific task.
RUN	To resume the parent system operation at a specified address.
QUALIFY	To specify qualifiers subsequently to be applied to implicitly defined symbols.
DEFINE	To enable the system programmer to define new symbols and, if necessary, to allocate space for the symbols.
IF	To make execution of a statement dependent upon existence of one or more specified conditions.
AT	To specify events at which a statement is to be executed.
CALL	To cause a prestored set of statements to be executed.
REMOVE	To remove previously entered AT or PATCH statements originated by the system programmer.
DISCONNECT	To specify that a terminal is to be disconnected from the support system.

In creating the support system command language, the need for flexibility was a prime consideration. Just as the designers of a machine language cannot antici-

pate every use of the finished product, neither can the designers of a support system language. The support system command language is intended to provide a general set of commands that impose no functional restrictions on the system programmer. The generality of the language makes it a highly flexible debugging tool. The commands can be used in many combinations to achieve a large number of debugging functions.

Although design of the command language is an important aspect of creating a debugging support system, the solution of internal implementation problems also offers significant challenges. To integrate a support system into a time-sharing system requires the implementors to overcome several complex problems.

A key implementation problem is that of making the support system transparent to the time-sharing system. When the support system is invoked, machine and program status must be saved so that operations can be restored at the end of support system operations. The integrity of the time-sharing system must be preserved during the support system activities.

This requisite transparency is achieved by utilizing preallocated, private work space on external storage into which is written that data necessary to restore main storage after the support systems use. In addition, the interrupt-handling modules of the parent system remain in main storage and are used to "stack" all interrupts directed to the parent system. The support system requires that the parent system be able to recognize only its activation signal, after which other surveillance tools are activated and the hardware is actually controlled by the support system.

Another example of the complexity of these problems is the implementation of a stand-alone Input/Output Supervisor (IOS). The IOS must be simple and straightforward, while providing flexibility in the fluid time-shared environment.

In retrospect, the system being implemented meets all the requirements as previously stated except one. The support system is not completely independent of the parent system, though that independence has been approximated in the most critical areas — the I/O supervisor and the user interface. Complete independence may be neither possible nor desirable except where it could justify a separate control processing unit, memory, and I/O system.

Future prospects for the support system

As mentioned previously, the flexibility of the support system prevents it from being "outgrown" by the demands of its parent system. Even as the parent system is expanded, the support system will remain capable of solving the next program bug. But even more interesting than this are implications of more sophisti-

cated applications of the system. Implied in its structure are the capabilities of forestalling the next program bug, decreasing the probability that the bug will occur, and perhaps ultimately eliminating bugs from the parent system.

Among the future prospects for the system are the following:

- Provision of the ability to have predefined debugging steps automatically executed when a problem occurs, thus enabling a system programmer to debug effectively even though he is not present at the terminal at the moment the problem occurs.
- Provision of the ability to interrogate the system at the micro-second level, thus easing the problem of capturing data at the split second a problem is encountered.
- Provision of the ability to monitor a system so closely that the system can be rapidly adjusted to achieve high-level performance.
- Provision of facilities to collect, compare, and evaluate data on program bugs, and ultimately (through selection of alternative solutions) to have a system restructure itself to avoid previously-recognized bugs.

These prospects are discussed in the following sections. It is important to realize that the basic knowledge needed to implement these functions exists today.

Provision of predefined debugging steps

The support system described in this paper gives the user an on-line real-time debugging capability. This capability is overshadowed by another more powerful capability inherent in the support system's on-site concept.

The systems programmer, in charge of solving user-detected systems problems, is impaired by the fact that he was not present to perform certain actions at the time the program bug appeared. It is indeed frustrating when an answer must be returned to the user containing a set of debugging instructions he must perform if and when the problem reappears.

The support system can be extended to include provisions for predefining the debugging steps to be performed when a problem occurs and for making those procedures a part of the total time-sharing system. An example will illustrate the extent of this capability.

There are certain "should not occur" points within a language compiler, i.e., there are points at which a compilation or assembly is aborted because of a programming or logic error. Instead of merely aborting the job, a predefined set of support commands could be executed to collect pertinent data (internal tables, transient records, etc.), to aid in debugging. The user, aware that his

compilation has been aborted, need not interfere in any way with time-sharing operation. He merely allows the system to perform the predefined debugging steps and then forwards the diagnostic data to the systems programmer.

Under this procedure, the systems programmer is sure that the data received from the user is what he needs to solve this particular type of problem since he, himself, defined the support system procedures to be performed at the error points.

The inclusion of a new command, ON, would provide this function. The following set of commands, comprising the debugging data set (DEBUG), could be used to accomplish predefined debugging function.

```
ON FORTNOGO CALL DEBUGFORT
```

```
ON ASSEMNGO CALL DEBUGASSEM
```

etc., for all language compilers/assemblers.

The phrase ON FORTNOGO or ON ASSEMNGO would cause the support system to implant AT phrases at all points within the Assembler or FORTRAN at which abort action could be initiated. Note that the user would not have to know the location of these points. The exit points and the procedures to be performed (as defined by the data sets DEBUGFORT and DEBUGASSEM) would be defined and supplied by the system programmer. Control over collecting data at these program error points would then be assumed by a central source, the systems debugging group. The total debugging efforts throughout the range of users could then be organized to yield optimum results. To use the predefined facilities, users would merely call the total debugging data set (DEBUG) before initiating time-sharing operations.

Gathering data at the micro-second level

The power of the support system could be greatly extended by introducing the SCOPE command, a specialized debugging tool. The multi-CPU environment in many time-sharing systems can be used to solve programming problems in areas that require micro-second level interrogation and response. The SCOPE command would specify that one CPU is to execute a series of operations continuously while the other CPU's function normally.

With the use of the SCOPE command, one could truly debug at a micro-second level. An example of the command would be:

```
SCOPE IF LOCKBYTE - 255 DISPLAY MODID
```

The command would force one CPU to continuously test the field LOCKBYTE for the value 255. When this

condition was satisfied, the module that changed the value to 255 would be identified in output. In this way, a systems programmer may determine which routine altered a parameter, and when the parameter was altered. More importantly, the entire system's activity could be suspended while automatic debugging routines were executed, and while an instantaneous "snapshot" of the parent system was taken for complete analysis.

Monitoring system performance

Since the support system is basically a meta-system for surveillance and data collection in the parent system, it is obvious that the support system may be used to monitor a "healthy" parent system as well as aid in debugging a "sick" one.

It is through the combined functions of system monitoring and the support system command language that the first steps can be taken to improve performance for a particular user. For example, a user could decide to monitor the number of FORTRAN compilations requested per time interval. Then he could request that, for his installation, FORTRAN modules be moved to a slower or faster input device, depending upon the previously monitored information.

Potential for the parent system to correct itself

The most far-reaching implication of the support system is that it creates the possibility of parent system self-correction. More specifically, it paves the way for a system that can learn to debug itself.

The support system provides extensive facilities for gathering information on operations in a time-sharing environment. Under current debugging procedures, this information is analyzed by the system programmer who decides on a way to correct or circumvent the program bug. The system programmer then evaluates the effect of the change and makes further modifications if necessary. Involved in this process is a series of selections and re-evaluations. Eventually, the system programmer learns the best thing to do in each of a variety of situations.

It is feasible to incorporate this human debugging process into the operating system or time-sharing system itself.

Consider the learning process involved in the gradual refinement of game-type programs (checkers, chess, etc.). These programs examine a current situation, select what appears to be the best action to take under a particular condition, take and record the action, and later evaluate it. Through extensive interplay of one

game program against another, the programs build up statistics and knowledge of the best actions to be taken under each of a multiplicity of conditions. With this accumulated knowledge, the game program "learns" to select the best alternative for a given condition.

The same type of system self-correction could be incorporated into an operating system or time-sharing system by building a selective process based on the recorded results of previous actions.

If we define the term "complex system" to mean the debugged system in its purest form, the following statement describes how to achieve the "bug-free" system:

"Selectivity through environmental feedback and previous experience as a source of selectivity (machine learning) leads to the creation of the complex system through the creation of stable intermediate forms upon which resulting more complex forms are built."—*The Architecture of Complexity*, Herbert A. Simon in *General Systems*, Vol. X, 1965.

Environmental feedback (i.e., knowledge of what has happened previously) is provided by the monitoring facility of the support system. Selectivity (i.e., the decision on the action to be taken) can be derived from the monitored information and from the recorded results of previous actions under the same or similar conditions. Whereas selectivity was previously determined by the debugging programmer's knowledge, skill, experience, and learning power, it could now be part of the system itself.

Since the basic mechanisms required to induce machine learning (feedback through system monitoring and selectivity through extended use of the support system) are available, the self-debugging system is a feasible concept.

ACKNOWLEDGMENT

The basic capabilities (with the exception of the ON and SCOPE functions) of the Time Sharing Support System described in the paper have been implemented by IBM as a Type I program under TSS/360. The authors wish to acknowledge the effort of the original designers of the Support System described herein.

Mr. M. E. Sherck

Dr. I. Rezucha

Mr. R. Heistand

The authors also wish to acknowledge the efforts of Mr. R. L. Bean in the editorial preparation of this document.

TSS /360: A time-shared operating system

by ALEXANDER S. LETT and WILLIAM L. KONIGSFORD

International Business Machines Corporation
Yorktown Heights, New York

INTRODUCTION

Experience with TSS/360 design, development, and application has been varied and interesting. For example, as we began putting the initial system together, significant performance problems were observed that had not been predicted by the earlier simulation efforts. These problems had not been anticipated because the paging characteristics assumed in the model development were significantly better than the actual system characteristics.

Measurements and analysis soon indicated that one significant problem was in the organization of the dynamic loader tables. This was overcome by splitting these tables into functional sub-tables, which greatly reduced paging during the loading process.

However, the paging problem was widespread. In most cases, the size of the actual code was two to three times the size expected by the model. In addition to the adverse effects on the available core space, this caused the paging input/output traffic to be significantly larger than expected, the level of possible multiprogramming to be smaller than expected, and the number of tasks wholly contained on the paging drum to be fewer than expected.

During the multi-terminal testing phase of TSS/360, another significant paging problem was discovered. Core-space control was based upon dynamically limiting the number of tasks active in core to the maximum that their estimated core usage would allow. A greater number of tasks would cause a high rate of unproductive paging within the system; a lesser number would not fully utilize the system facilities. The core-space control was not functioning properly due to bugs. A temporary solution was to restrict the number of active tasks in core to a fixed number. This reduced the performance problem but, when the same test cases were later run under the correctly operating dynamic algorithm for core-space control, the dynamic algorithm was found to be consistently more effective than the

static algorithm had been. This generally valid conclusion has been demonstrated in all areas of TSS/360.

Since the initial release of TSS/360 in October 1967, performance has improved significantly with each subsequent release. The initial emphasis was on building a stable system, followed by extensive measurement-and-analysis efforts to identify potential system modifications. Then, through comparatively small changes in coding and resource management algorithms, the system performance was significantly improved.

From the material presented in this paper, we feel that several conclusions—which are supported by our operating experience—can be drawn:

- Paging is a sound concept. As expected, it is a direct solution to the dynamic-core-storage-allocation problem.
- Paging also allows for a hierarchy of auxiliary storage which, in the case of TSS/360, involves high-speed drum and slower-speed but larger-capacity disk files. A larger number of users can be supported economically on a system by subdividing each user's space requirements between drum and disk storage. From experience, sound algorithms can be developed for management of this storage, which is critical to the overall performance of the system.
- A data set access method based on page-size fixed block images has the same simplicity of implementation and elegance of application as in the core-storage situation.
- In the improved command system, we have found that a highly adaptive, open-ended system is not only more valuable to terminal users, but simpler to implement.
- The best strategies for resource allocation are those that address the allocation of all system resources in an integrated way, rather than optimizing specific sub-portions. In general, the simple round-robin strategy is fairly good, but

the need to emphasize certain characteristics (such as response time to conversational requests) requires the separation of resource requests by priority.

It is the purpose of this paper to highlight the key elements of TSS/360—control system organization, user services, and task structure—in order to describe and explain the design of a time-shared operating system.

Control system organization

There are many possible resolutions to the questions concerning the division of functions within a control program, the interfaces between portions of the control program, and the decision as to which portions are to be resident in main storage and which nonresident.

In the design of TSS/360, the historical examples of the MIT Compatible Time Sharing System¹ and the IBM Time Sharing Monitor system² led towards the concept of a small, fully resident monitor whose primary function would be to create a multiprocessing, multiprogramming environment.

In TSS/360, this monitor is called the Resident Supervisor. The Resident Supervisor is interrupt driven, and is responsible for controlling the real resources of the system and for performing services in response to requests originating from tasks. A task represents the environment in which a user's processing is performed. There is one task for each conversational user of the system. A fundamental design decision was to provide within each task the facilities of a full operating system. Figure 1 depicts this overall system structure.

Task processing is always performed in relocation mode with the dynamic-address-translation feature activated. Tasks are therefore said to operate in virtual memory.

The Resident Supervisor, on the other hand, does not use dynamic address translation—that is, instructions within the Resident Supervisor have main storage addresses, not logical addresses, as operands. The decision to make the Resident Supervisor operate in the non-relocation mode was based upon the efficiency resulting from eliminating dynamic-address-translation overhead and upon the increased protection resulting from the fact that no location within the Resident Supervisor can be addressed by a program operating in virtual memory. Resident Supervisor routines, however, are capable of addressing all of main storage and of executing all of the instructions in the System/360 instruction set.

Another basic TSS/360 design decision was to have tasks be interrupt driven like the Resident Supervisor. It was felt that this structure provided the maximum of flexibility in task development. Accordingly, task-con-

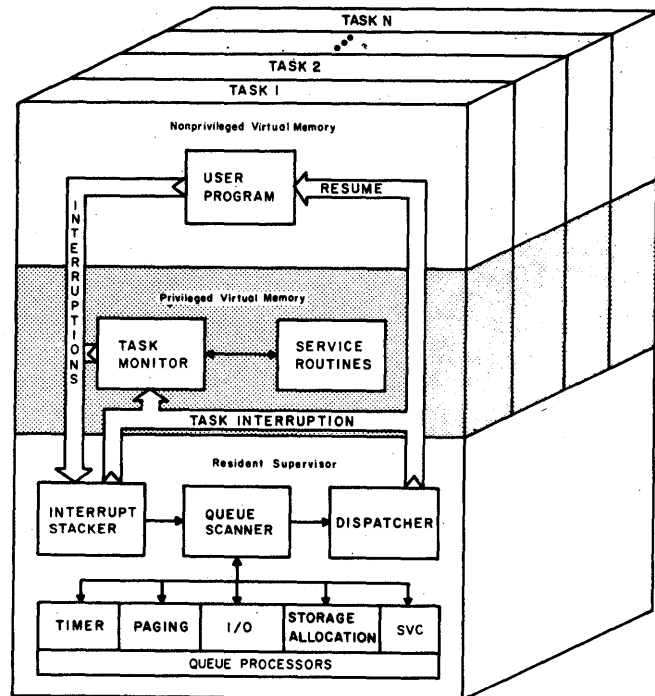


FIGURE 1—TSS/360 program structure

rol structure is in many ways analogous to the control structure of the Resident Supervisor.

In order to provide a wide variety of control program services, while at the same time protecting user tasks from each other, task virtual memory routines are divided into two classes:

- privileged routines, which operate in the privileged state;
- nonprivileged routines, which operate in the nonprivileged, or user, state.

As the term implies, routines operating in the privileged state are authorized access to many supervisor services denied to routines operating in the user state. In this way, most parameter validation and other protection checking can be eliminated from the Resident Supervisor. In addition to decreasing the overall size of the Resident Supervisor, this arrangement allows supervisor services to be more general and powerful.

The way in which the privileged state is implemented is as follows:

- In a task's virtual memory, pages that are allocated to privileged routines (and their associated tables and work areas) are assigned a storage protection key that differs from that assigned to user programs. This will cause a storage-protect interruption if the privileged part of a task's virtual memory is addressed by a user program. Priv-

ileged routines, on the other hand, can address all of the task's virtual memory.

- The dynamic loader service routine will not treat modules from a user's library as privileged routines. Thus, an ordinary user cannot cause his own version of a system routine to be loaded and executed as a privileged routine, a facility available to the systems programmer.
- A user program normally requests system services through instructions whose execution cause an interrupt to the Resident Supervisor. (In system/360 such interruptions are termed supervisor calls.) In response to the supervisor call, the Resident Supervisor, by manipulating CPU status, creates a task interruption to invoke a privileged system services routine. The privileged routine can then determine if the user's request is valid. If it is, the privileged routine may then invoke other TSS/360 supervisor calls while in the process of performing services. If the request is not valid, it will be rejected, thus preventing a nonprivileged routine from causing incorrect system operation. The reason for communicating between non-privileged and privileged state via the Resident Supervisor is that only the Resident Supervisor can execute the instruction that alters a task's protection key and, therefore, its state.

The privileged system service routines constitute the bulk of the TSS/360 operating system. These routines are either shared by all tasks or are located in independent service tasks. Printers, for example, are serially shareable and thus are serviced through an independent task. On the other hand, the dynamic loader provides service to each task and is therefore shared in parallel.

System control elements

The Resident Supervisor is primarily composed of an interrupt stacker, a queue scanner, several processors, a number of error handling and service subroutines, a dispatcher, and the tables that form the system's data base.

Entry into the Resident Supervisor is via an interruption. Some interruptions are processed immediately either because of their urgency (e.g., interruptions denoting CPU malfunctions) or for efficiency (e.g., interruptions which require a change of task state).

For most interruptions, however, the interrupt stacker builds a record called a generalized queue entry (GQE), into which a description of the interruption is placed. This GQE is then placed upon an appropriate queue. A GQE is a standard control block used throughout the Resident Supervisor to contain a description of the work to be done by a device or facility that is con-

trolled by the Resident Supervisor. Quite frequently, one control block may belong to several queues and contain forward and backward pointers to each of them. In processing these multi-threaded lists, the Resident Supervisor becomes, in effect, a list processor.

Interruptions are disabled during processing in the interrupt stacker. However, in contrast to many systems, the Resident Supervisor generally executes with interruptions enabled to facilitate processing of interruption queues, on a priority basis, without regard to sequence of arrival. When the interrupt stacker completes processing, it generally exits to the queue scanner.

Every system needs some facility for sequencing the work to be performed by the control program. In systems which operate with interruptions disabled, the hardware priority-interruption system provides this function for the interrupt-handling routines, and some other control-program routine provides a similar function for the system's resource-allocation routines. Within TSS/360, these two functions have been combined into one centralized queue scanner and a scan table. Each system queue is anchored in the scan table.

Because the queue scanner is a central facility within the Resident Supervisor, it must operate efficiently if the Resident Supervisor is to operate efficiently. To achieve this efficiency, the queue entries in the scan table are organized to minimize the number of entries that must be inspected when the scanner is searching for work. Moreover, the organization of scan-table entries reflects an awareness of the possible interactions among queues so that, for example, an exit is not made to a processor only to find that a needed facility (such as an I/O path) has been allocated to some other request.

When the queue scanner finds work that can be done, it passes control to the appropriate processor; when it determines that there is no currently available supervisor work, control is transferred to the scheduler and dispatcher.

TSS/360 was designed for a generalized multiprocessing environment in which multiple CPUs may be simultaneously executing the single copy of the Resident Supervisor. To facilitate multiprocessing, it was necessary to define a number of programmed interlock flags to prevent unwanted recursion and logical race conditions. In general, TSS/360 used the approach of defining a small number of interlocks, each covering a wide scope. These interlocks generally guard entrance to the queue processors and to the major system data bases.

The purpose in minimizing the number of interlocks is two fold:

- First, placing interlocks at the entrance to the queue processors tends to prevent a CPU from

entering a path of logic only to soon be forced to await the resetting of an interlock. When the queue scanner finds an interlocked queue processor, it simply bypasses inspecting that queue and proceeds to the next entry in the scan table.

- Second, in a multiprocessing situation, it is desirable to permit one CPU to perform error-recovery procedures whenever another CPU encounters a processor or storage unit error. Because all processors use the single copy of the Resident Supervisor, it may be necessary for the recovery CPU to reset programmed interlocks initially set by the malfunctioning CPU. This means that the recovery CPU must be aware of the reason why the interlock was set. The fewer the system interlocks, the simpler the recovery procedures can be.

In general, a queue processor locks its associated queue upon entry and unlocks the queue as soon as the processor has dequeued a GQE for processing. In certain cases a queue processor may lock a queue until some specific future event or condition has occurred. Each scan table entry has indicators reserved for such use.

TSS/360 has adopted a policy of concentrating the physical locations of the interlock flags in an orderly fashion within a very few key system tables. This has proved to be a valuable aid to the development programmers, who can determine the status of the Resident Supervisor by inspecting or displaying these system tables.

The following is a brief description of the major processors within the Resident Supervisor:

- *Task Core Allocation:* Controls the overall core storage space; it processes requests for space allocation and responds with the location assignments
- *Auxiliary Storage Allocation:* Controls auxiliary-storage space; it processes requests for drum and disk page-space allocation.
- *Page Drum Request:* Processes input or output requests for the auxiliary paging drum. Because of the unique mechanical characteristics of the drum (several pages per track with instantaneous switching), the requests are sorted by angular position to maximize throughput.
- *Page Drum Interrupt:* Processes interruptions that are the result of paging drum input/output operations. This processor will attempt to keep the drum I/O channel busy by adding drum requests to an active drum I/O channel program. It calls the page-posting routine to process the results and releases core space when appropriate.
- *I/O Device Request:* Processes requests for I/O operations to devices *other* than the auxiliary storage drum; it first determines, by calling the path-

finding subroutine, if a free path to the requested I/O device is available and, if possible, reserves a path.

I/O device requests are either disk paging requests or other I/O requests. For disk paging requests, a subprocessor is called to convert the request into an I/O channel program. For other I/O requests, a request control block chained from the queue entry already contains the I/O channel program. This I/O program is normally created by the task requesting the I/O. The I/O operation is started by the request processor, which returns to the queue scanner.

- *Channel Interrupt:* Processes input/output interruptions that originate in other than the paging drum. It determines if the interruption is synchronous or asynchronous by verifying if a request on the corresponding device-request queue had initiated the operation. If the interrupt is synchronous, various processing is performed. If the interrupt is asynchronous, an interrupt entry is queued for the task currently associated with the device. If no task is currently associated with the device, and it is a terminal, the channel interrupt processor will call a routine to create a new task that will then be dispatched. The newly created task will begin execution of the appropriate task-initialization routines in response to its initial interrupt.
- *Timer Interrupt:* Processes timer interruptions; it determines if a task has reached the end of its time-slice or whether a task-specified time interval has elapsed. At time-slice end, various processing is performed. For task-specified intervals, a task-simulated timer-interrupt entry is queued for the task.

TSS/360 error recovery and retry procedures are designed to dynamically correct errors or to minimize the effect of errors on the system as a whole. Although the specific recovery procedures differ for each type of error, the general approach to recovery is the same. Failing operations are retried where possible, failing hardware devices (e.g., a CPU or I/O device) are checked and intermittent failures retried. Where an operation cannot be retried at all or is retried without success, a "hard" failure is recognized and fault localization, to the component level, is invoked. The failing element or device is removed from the system in an orderly manner, so that only the affected tasks are disrupted. An environment record is generated for later analysis by service personnel and the **system** continues operation. It is only as a last resort, when recovery is not possible and when removal of the failing component would

render the system inoperative, that the system is shut down.

In addition to the queue scanner's scan table, the Resident Supervisor contains data bases to describe task status and to describe I/O path status.

Each task has associated with it a control table that is separated into portions. The first portion is needed for scheduling and control purposes, so it is kept continuously resident in main storage. The second portion contains the task's relocation tables that must be in main storage during a task's time-slice, but not necessarily between a task's time-slices.

To allow a user's program to be highly device-independent and to allow the Resident Supervisor to remain relatively insensitive to dynamic changes in system configuration, TSS/360 users normally employ device-class codes that describe a device as a member of a class of like devices. Furthermore, the TSS/360 access methods employ symbolic addresses to designate devices.

The Resident Supervisor uses a group of tables, called pathfinding tables, to translate a symbolic device address into a hardware address that specifies a path through a channel control unit, channel, and device control unit to the device. The supervisor-maintained pathfinding tables are used to determine if a device is busy instead of attempting to physically address the device. In a typical environment, it is expected that there will be multiple paths to most devices. In such a situation, the efficiency of I/O processing will be increased by reducing the number of "busy" or "unavailable" conditions encountered during an attempt to initiate an I/O operation. The use of common pathfinding tables also assists in synchronizing I/O processing in a multiprocessing environment, because an I/O interruption may be accepted by any available CPU, not just the CPU that initiated that operation.

In retrospect, the design of the Resident Supervisor has proved to be sound and remains, in outline, essentially as initially described in 1965.³ Experience has shown that it is nearly impossible to predefine an optimal overall system. A significant amount of tuning of resource-control algorithms and processing procedures must be expected. We have found that the best method to do this tuning is by modification and measurement of the running system.

Task control elements

TSS/360 includes a scheduling algorithm for determining the sequence of allocation of CPU time to competing tasks. As implemented initially, the scheduling algorithm divided tasks into conversational and non-conversational groups.

The original algorithm followed a round-robin schedule for the active tasks (those not waiting for the com-

pletion of some event, such as terminal input). Conversational tasks were scheduled for dispatch in consecutive order to the end of the list. At this point, a test was made to determine if an installation-specified real-time interval had elapsed. If not, the system devoted the remainder of the interval to the round-robin execution of the nonconversational tasks. If the interval had been exceeded, the system went back to redispatch the first active conversational task.

As a result of system experience, this algorithm was modified. All active conversational tasks are now dispatched in round-robin fashion until no further active conversational tasks are available. Then the system begins to dispatch active nonconversational tasks, but with provision for pre-emption whenever a conversational task becomes active. Instead of round-robin execution of the nonconversational tasks, the system tends to run to completion as many nonconversational tasks as can be effectively multiprogrammed within the available core resource. This modification was incorporated because round-robin scheduling for the non-conversational tasks served no useful purpose and reduced system throughput by causing the system to do additional paging in switching resources.

The scheduling algorithm outlined above is not considered to be the optimum for general time-sharing operation in any specific customer's installation. Experience with scheduling algorithms and their effect upon the system dictated the need to provide a flexible facility for modifying the task-scheduling algorithm. TSS/360 is adding this facility, called the table-driven scheduler, in which table entries are made to define sequences of states and attributes that a task can assume. When created, each task is assigned an initial table entry in which specific parameters explicitly state:

- the relative priority of every task associated with that table entry
- whether such tasks may be interrupted by a higher priority task
- the time-slice quantum to be allocated to the task
- the maximum core space to be allocated to the task
- other parameters concerned with the action to be taken when execution of a task is suspended.

Execution of a task can be suspended for reasons such as time-slice end, terminal-wait condition, or excessive paging. Associated with each of these conditions is a value specifying the table entry to be assumed by the task on the occurrence of that condition.

The collection of schedule table entries, which can be prepared at each installation, specify the scheduling algorithm to be followed by the system. The table entries can range from extremely simple ones that

simulate a round-robin queue, through exponentially related algorithms, to complex time-and-priority algorithms.

The allocation of the CPU resources to tasks, to best carry out the sequence selected by the scheduling algorithm, is controlled by the dispatcher. The dispatcher first determines if a new task can be placed into execution. This is determined by comparing an estimate of the core pages a task is expected to require during its next time-slice with the number of unreserved and available core pages. The estimate of a task's page requirements is based on its activity in the preceding time-slice. If enough core pages are available, the count of available core pages is reduced by the estimated number and the task is prepared for execution. This dynamic control of the number of tasks allowed to concurrently execute in core storage is vital to avoid overloading a paging system such as TSS/360.

A modification has been made to the dispatcher to dynamically detect CPU-bound tasks. When more than one task is ready for immediate execution, non-CPU-bound tasks are dispatched before CPU-bound tasks. Through this strategy, the system dynamically maximizes its probability of multiprogramming (overlapping I/O with computing).

When a task is selected for immediate CPU execution, a task-interrupt-control routine is entered. The need for a task-interrupt mechanism arises because the Resident Supervisor processes requests for system services in a logically independent fashion, that is, the Resident Supervisor may be concurrently performing several services for a task. There is no way to forecast the order or time of completion of processing of each of these services.

Therefore, for a task to operate asynchronously with respect to the completion of system services, a task-interrupt mechanism has been created that is analogous to the hardware-interrupt mechanism that allows the Resident Supervisor to operate asynchronously with respect to the real computer system. Operation of task interruptions is similar to hardware interruptions. The major difference is that the hardware interruptions convey a change in the status of the entire system to the Resident Supervisor, while the task interruptions represent a change in status of only that portion of the system currently allocated to the task being interrupted.

A task interruption is requested by a Resident Supervisor routine when it discovers an event, such as I/O completion, whose further processing is a task's responsibility. However, a task is not always prepared to receive an interruption; further, the task for which the interruption is destined may not be the next task to be

dispatched. So there is a software queueing-and-masking facility that is analogous to the hardware facility.

Before control is given to a task, the dispatcher transfers control to the task-interrupt-control routine, which checks the task's interruption queues for unmasked pending interruptions. If none is found, control is given to the task at the location saved in its control table.

If pending interruption is found, the task-interrupt-control routine changes the location pointer to point to an appropriate interruption processor of the Task Monitor. Now, control will go to the interruption processor. This action of influencing the dispatcher's transfer of control is called a task interruption.

The Task Monitor consists of a group of privileged service programs that receive and process task interruptions on a priority basis via queueing, scanning, and dispatching mechanisms analogous to those of the Resident Supervisor. The Task Monitor may thus be considered a task-interrupt handler, whereas the Resident Supervisor is a hardware-interrupt handler.

The Task Monitor performs these major functions:

- Provides an interface with the Resident Supervisor for receiving and analyzing task-oriented interruptions.
- Provides linkage to required service routines or user routines, either by immediate dispatching or by queueing the interruption for later dispatching in a priority sequence.
- Maintains the integrity of the task and service routines that are dispatched, primarily through save-area management.

The Task Monitor is designed to provide for flexible handling across a wide range of interruptions. Thus, it provides an ability to dynamically specify task-interrupt-handling routines and to dynamically arm, disarm, and change the relative priority of these routines. As with the queue scanner of the Resident Supervisor, provision has been made to use this generalized process in an efficient manner.

User services

Because TSS/360 is a comprehensive operating system, it offers a wide variety of user services⁴, such as:

- Command system
- Program control subsystem
- System programmer support system
- Catalog management
- Page-oriented data management
- Magnetic-tape and unit-record data management
- Dynamic program loading
- Virtual memory allocation
- External storage allocation

- Resource control and accounting
- Task-interruption control
- Language processors

From this list, we have chosen to describe in this section the command system, the page-oriented data-management services, and the dynamic-program-loading services. Not only does each of these represent a key aspect of TSS/360, but each has relevance to problems of general interest.

Command system

The command system is the principal interface between a time-sharing system and its users. Therefore, it has a position of special importance in TSS/360.

Initially, the TSS/360 project attempted to define a set of commands that would be satisfactory for all users. The result was a rigid set of commands that completely satisfied no one. This experience led to the conclusion that it is better to implement a command system than a command language.

As a result, TSS/360 now contains a flexible command system that is delivered with a set of simple commands that can either be employed as is or be completely replaced and expanded in a straightforward fashion. This approach allows each installation and, more important, each user at an installation to customize the system-user interface to his own needs.

In TSS/360, the syntax of the command system has been separated from the semantic content of command statements. This regularization of syntax and structure has resulted in a simpler implementation utilizing a single, centralized command analyzer and execution facility.

The command-system syntax is simple and natural. Each command consists of an operation name, which is usually followed by one or more operands. As supplied with the system, the delimiting character for the operation name is a blank or tab; the delimiter between operands is a comma; the delimiter between commands is either a semicolon or the end of a line of input; and the line-continuation flag is a hyphen entered as the last nonblank character of a line.

When an individual enters his commands conversationally, he is told of the actions taken by the system in response to each command and, when necessary, he is prompted for additional non-defaultable information needed to complete an action, is informed of errors (if his command entry is either incomplete or incorrect), and is told of the options he may exercise in response to an error. Special care has been taken to make the types of options consistent for all commands. Nothing, for example, could be more frustrating to a user than to be required to resubmit an operand with delimiters in one situation and without delimiters in another.

Each user can establish his own spellings, abbreviations, or operation names for commands through a SYNONYM facility. Use of this facility sets up one or more equivalences for the original name but does not destroy it.

Any command operand may be entered either by position or by keyword. Keywords may appear in any order and have the general form `KEYWORD = value`, where `KEYWORD` is the name of the operand and "value" is the actual value of the operand. For each command operand, the user may select the form that is most convenient for him. A keyword has a global meaning since it is associated with the value to be passed, not with the particular command invoked. Therefore, the SYNONYM facility, available for command operation names, is also available for keywords. In contrast to many other systems, almost every command operand has a default value. Moreover, the user need not accept rigid default values for operands, for he can easily override those supplied with the system. For example, a standard default for the FORTRAN compiler might be to produce an object code listing. Any TSS/360 user can individually change this default so that, in his case, the language processor will not produce an object listing unless he specifically requests it.

TSS/360 maintains a special prototype data set that is copied into the user's library when he is initially joined to the system. This data set, called a user profile, contains three tables: the first specifies the initial default values for command operands; the second contains his character-translation list (to allow redefinition of printing characters and control characters); and the third contains command operation names and equivalences. The user can modify any of the entries in these three dictionaries, which, in conjunction with the command system, define his command language.

The command system includes as a fundamental feature a command procedure facility, which permits the user to create a stored procedure comprising commands and logical statements that control the flow of command execution. Invocation of a command procedure is identical to invocation of a system-supplied command. The command statement consists of the procedure name followed by a series of parameters, whose values are inserted by the command system at the proper points in the procedure. The resultant statements will be interpreted as though they had originated in the input stream. For maximum power, command procedures can be nested and/or recursive. When defining a procedure, a user can utilize the facilities of the TSS/360 text editor. Once defined, a procedure may be edited, shared, copied, etc., as with any other file.

Another interesting feature of the command system is the use of "null commands." For example, immedi-

ately after a user has signed on the system but before control is returned to the terminal, TSS/360 automatically invokes a command procedure called ZLOGON. As initially supplied with TSS/360, ZLOGON is a "null" command—it does nothing. However, the individual may redefine the ZLOGON command procedure to perform functions to augment the initialization of his task. Thus, "null" commands are conceptually similar to the "user exits" frequently associated with general-purpose programs.

The command system also provides a facility for defining new command primitives. Efficiency can be enhanced through use of this "Built-in" facility as the command system can directly bypass much of the interpretive processing required in the expansion of command procedures.

Still another feature of the command system available to the user is the ability to augment system-message handling:

- He can request explanation of system messages or of key words in such a message; word explanations may continue to a number of levels.
- He can dynamically specify the classification of messages he is to receive; this filtering, or masking, capability provides different message-severity levels and message lengths.
- He can construct a personal message file that will be issued in lieu of the corresponding system-supplied messages.

The command system also provides a flexible system for handling attention interruptions that is quite useful. For instance, suppose a user has forgotten to identify a library that contains a subroutine required by his mainline program. When he receives a system diagnostic message, he can use the attention button to re-enter the command mode, define the library, and then resume processing at the point where the message was issued.

The program control subsystem of the command system is a powerful facility that permits a user to inspect and modify programs during execution. These dynamic control facilities eliminate the need for user-written debugging and control instructions that must be pre-planned, coded into the user's programs, and then later removed.

The output from the TSS/360 language processors may optionally include a dictionary containing the values and other attributes associated with the symbols or variables used in the source program. Through the use of this dictionary, the program control subsystem can properly interpret debugging statements utilizing source program symbols and can properly format its input and output.

Even during the initial shakedown of TSS/360, there

were many users who insisted upon using the system only because of the power associated with a dynamic execution-control system. This has made clear that an essential element of any interactive system must be a dynamic symbolic debugging and control facility.

Page-oriented data management

The access methods that support page-oriented data management in TSS/360 are called virtual access methods. The name "virtual" was given to these access methods to reflect the fact that they utilize only one physical block size—that of a page. The virtual access methods were specifically designed for a time-sharing environment and present a clear division between data set management and physical device management. Each of the three virtual access methods provides access and processing capability for a specific type of data set organization:

- Virtual sequential access method (VSAM)
- Virtual index sequential access method (VISAM)
- Virtual partitioned access method (VPAM)

In all three of these access methods, only data set management is performed in virtual memory; the construction and execution of channel programs and error recovery (i.e., physical-device management is performed by the Resident Supervisor. The direct-access volumes, on which TSS/360 virtual organization data sets are stored, are entirely formatted into fixed-length, page-sized data blocks. No key field is required. The record-overflow feature is utilized to allow data blocks to span tracks as required.

The page-sized block for data storage was selected for a number of reasons. For example, rotational delay is a significant factor in direct-access throughput, since it cannot be overlapped as mechanical-seek time can. Any block size significantly smaller than a page would be extremely wasteful of total direct-access capacity unless elaborate strategies were utilized to avoid rotational delay.

The need for a large block size is also apparent when the simultaneous direct-access activities of multiple users are considered. Due to conflicts in demands for access arms, a mechanical seek may frequently be required before accessing a data block. A larger block size makes better use of the total access cycle while, at the same time, reducing the frequency of access requests by each user.

The direct-access volume-packing efficiency is also quite high for page-sized blocks. First, the data-recording space is utilized at better than 90% of the theoretical capacity that could be obtained by the use of cylinder-length blocks. Second, the smallest external-storage

allocation unit is a single page; hence, a large number of small data sets can be kept on one volume. Furthermore, large data sets need not be allocated physically contiguous external storage space. This contributes to higher volume packing efficiency by reducing external-storage space fragmentation.

The physical representation of a typical virtual sequential organization is shown in Figure 2. The specification of any virtual data set is contained within the data set's external page map, which is stored on the direct-access volume together with the data pages. There is one entry in the external page map for each page-sized block occupied by the data set. The content of an entry specifies the location of a block in external storage. The position of the entry within the external page map signifies the relationship of the associated block relative to the other blocks in the data set.

For the three-page data set shown in Figure 2, the external page map shows that the first data block is between the other two pages of the data set. This example emphasizes that block relationships in the data set are determined by the contents of the external page map rather than by their physical position within the volume. This concept allows the virtual access methods true device independence across the range of direct-ac-

cess devices. That is, it is perfectly feasible for a data set to have physical records recorded on, say, the IBM 2311 Disk Storage Drive and the IBM 2314 Direct Access Storage Facility in any mixture. Furthermore, because information is referenced relative to the beginning of the data set and not by its location with respect to an external-storage device, it is entirely practical to move data sets (or portions of data sets) among a hierarchy of devices.

In a typical virtual index sequential organization, three classes of blocks can be specified within the external page map: directory pages, data pages, and overflow pages. One entry, corresponding to the lowest record key in each data page, is placed in the directory. Records are maintained in collating sequence within the the data set by key value. To find a given record, the directory is searched and then the data page containing the record is searched. Locator entries, corresponding to each record within a data page are stored in the back of the data page. Space in overflow pages will be assigned when record insertions exceed the capacity of a data page. The record locators in the primary data page will point to secondary locators within the overflow page. The placement of data and locators within the same block is a significant convenience associated with choosing a fixed block size, and is in contrast to many contemporary systems.

In a typical virtual partitioned organization, two classes of page blocks can be specified within the external page map: directory pages and member pages. The partitioned organization directory contains an entry describing each member, which is specified as a contiguous group of entries within the member-data portions of the external page map. Members are subsidiary data groups that may have sequential or index sequential organizations (or any combination of the two). Members can be expanded or contracted by simply adding or deleting entries within the external page map. The partitioned organization allows a user to manipulate individual members or to conveniently treat a group of data sets as a single entity for purposes such as creating libraries or sharing data sets through the system catalog.

Two types of interlocks are provided to coordinate simultaneous access to shared data sets by more than one user:

- Read interlock: prevents another user from writing into the interlocked data space; other users may have read-only access at the same time.
- Write interlock: prevents another user from reading or writing the interlocked data space; can be set only when no other interlock is set.

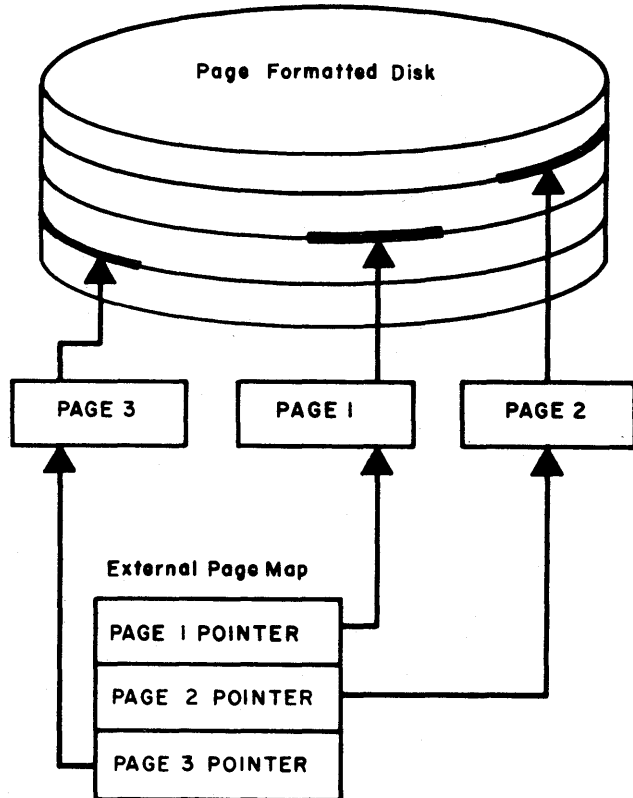


FIGURE 2—Typical virtual sequential organization

Interlocks are established at various data space

intervals, depending on the data set organization. Virtual sequential organizations are interlocked at the entire data set level. Virtual partitioned organizations are interlocked at the individual-member level. Virtual index sequential organizations, however, are interlocked only at the individual data-page (block) level; this allows a much finer level of sharing than is available in most other systems. The control mechanism for sharing has been simplified significantly by the choice of placing interlocks at the level of the physical block, rather than at the level of the individual record.

When a logical record is wanted (in a straightforward case), the flow of control is as follows. The appropriate external-storage address of the record's page is obtained from the external page map. This address and the virtual memory address of a buffer are passed to the Resident Supervisor in a request list.

The Resident Supervisor places the symbolic device address and relative block number in the relocation-table entry associated with the buffer's virtual address. However, the page itself is not yet read into main storage. It is only when a user addresses a record in his virtual memory buffer that a paging-relocation-exception interruption occurs, causing the Resident Supervisor paging processors to bring the page into main storage.

The virtual access methods write onto external storage only those pages of the buffer that have been modified. When it is necessary to write a buffer page onto external storage, the appropriate virtual access method routine obtains an external-storage address for the page from the external page map and passes the virtual memory address of the buffer, together with this external-storage address, to the Resident Supervisor. The appropriate Resident Supervisor routines then write the buffer page into the data set on external storage.

The external page table maps the external-storage locations of a given portion of the data set into a virtual memory buffer. The size of the buffer controls the extent of virtual memory allocated to the data set. This second level of mapping allows the user to process a page-oriented data set that can be as large as 65,000 pages, which is a great deal larger than the 4096 pages available in a 24-bit-addressed virtual memory.

TSS/360 brings into the buffer only those pages of the data set that are currently needed. The size of this buffer need not be limited to one page; it may be as large as a segment (256 pages), thereby allowing a user to address all or a portion of a data set in the same manner as main storage.

The TSS/360 user thus has a choice that allows him to treat a properly organized data set as a file or as one-level storage. There are several advantages, however, involved in the use of traditional data management macro instructions, such as GET and PUT. For ex-

ample, while information within auxiliary storage is vulnerable to a system failure, information that is maintained through macro instructions is updated directly on external storage, and is thus preserved across system failures. In addition, macro instructions directly signal when buffer contents are no longer required and thus enhance efficient auxiliary space management.

As described, the virtual access methods perform a programmed search of data set indexes in virtual storage. Conceptually, this amounts to combining the benefits of paging large indexes with the benefits of substituting high-speed auxiliary drum storage for slower speed disk storage.

This concept of programmed searches can be extended by user programs to secondary indexes for data sets. For example, the TSS/360 Assembler macro library is maintained as a line data set for maintenance purposes. However, the library must frequently be accessed alphabetically on the basis of macro instruction name. A list of such names combined with the line numbers locating the macro instruction is maintained, alphabetically sorted, in a separate sequential data set. When it is desired to locate a particular macro instruction, the entire alphabetically arranged name list is brought into virtual memory and a programmed search is performed to locate the appropriate index (i.e., line number) to the macro library.

We have found that implementation of the virtual access methods required significantly fewer lines of code than were required for a corresponding set of TSS/360 access methods used to support physical-sequential devices, such as printers and magnetic-tape units. It is apparent that the removal of device-dependent operations (with complex channel programs), the standardization of block size, and the elimination of exceptional procedures (such as end-of-volume operations) simplified the actual coding for the virtual access methods. Furthermore, the separation of data set management from physical device management simplified debugging.

Program loading services

In TSS/360, program loading is dynamic; that is, during execution one program may reference another program that has not been previously processed by the dynamic loader. Although not unique to TSS/360, this is another of the means by which a user is given flexibility during his terminal sessions.

In most conventional systems, there are a number of difficult design trade offs associated with dynamic loading. For example, the available memory space must be apportioned in some way between the storage requirements of the link-loader and the option to leave the program to be loaded in main storage. As another

example, the cost of performing basic linking and un-linking functions during program execution must be traded off against the potential inefficiencies of passing inter-module parameters by value.

In TSS/360, the loading process is performed in virtual memory. The large virtual memory environment of TSS/360 permits a disassociation of claims on address space from claims on main storage, and thus allows the allocation of storage to be optimized on a system-wide basis. Moreover, because of the large virtual store environment, it is seldom necessary to unlink program modules. This makes it unnecessary to place system restrictions upon the form of intermodule references.

A program module generated by a language processor resides in the system as a member of a partitioned data set before being loaded and, in this state, consists of at least two parts: text and module dictionary. A third part, an internal symbol dictionary, used by the program control subsystem, is optional.

The text of the program module is divided into control sections. This division is determined by source language statements for output generated by the Assembler, and automatically for output generated by the FORTRAN compiler.

From a system standpoint, the purpose of control sections is to allow a program to be divided into portions whose virtual memory locations can be adjusted (independently of other sections) by the dynamic loader without altering or impairing the operating logic of the program.

For the user, a control section is a segment of coding or data that can usually be replaced or modified without reassembling an entire program. A control section also represents a segment of coding or data to which attributes can be assigned independently.

At the time the user creates a control section, he may assign a variety of attributes to it, such as:

- fixed length
- variable length
- read-only
- privileged
- shared

The module dictionary consists mainly of a group of control-section dictionaries, one for each control section of the program module. A module dictionary describes the text: its length, attributes, external-symbol references and definitions, and information to be used in relocating address constants. Collecting all linkage data into one module dictionary allows the TSS/360 dynamic loader to calculate linkage addresses without bringing the larger text portion of the module into main storage.

In TSS/360, the dynamic loader resides in virtual

memory. The basic functions of the loader are to load programs into virtual memory—not into main storage—and to relocate only those address constants that are in pages of text actually referenced during execution of the program.

The process of loading a program into virtual memory does not involve the movement of any text and is performed in the allocation phase of the dynamic loader. Loading a program into virtual memory consists, in large part, of establishing the addressability of the program within the virtual store.

When the dynamic loader's allocation phase is invoked, it utilizes the virtual access methods to locate the program library containing the requested object program module.

Utilizing information from the module dictionary, the loader requests the allocation of a virtual memory for the object module text. Virtual memory allocation involves the creation of relocation table entries for the text and the assignment of protection keys according to the attributes of each control section. The loader next places the external-storage addresses of the module's text pages into the relocation table entries just created. Locations within a program are addressed through base registers, index registers, and displacements. Base registers generally contain values obtained from address constants. For each text page that contains address constants, an "unprocessed by loader" flag will be set in the appropriate relocation-table entry.

Among other functions during this phase, the dynamic loader examines all external references of the module, and obtains and processes the module dictionaries for any additional object modules required to satisfy these external references. This process results in the dynamic loader recursively invoking itself as long as additional dictionaries must be obtained.

When the allocation phase is complete, the dynamic loader exits, supplying location values that correspond to entry points in the loaded program.

The second phase of the dynamic loader is invoked when a page containing address constants is referenced and consequently brought into main storage during program execution. Address constants on the page are adjusted to reflect the values calculated during the allocation phase of the loader.

A secondary function of the dynamic loader is to enforce the TSS/360 protection rules concerning the loading and referencing of program modules.

During the allocation phase of the dynamic loader, the content of each module dictionary is placed in a private task table. Called a task dictionary, this table contains the information needed to load (and unload) modules for particular task. A task dictionary consists of a header containing three hash tables, and a body con-

taining one module dictionary for each module loaded for the task.

To link programs dynamically, the dynamic loader must be able to look up all external-symbol definitions in an efficient manner; hash tables, consisting of headers and a number of hash chains, are used for this purpose.

To reduce the number of pages referenced during the loading process and to prevent a nonprivileged user from accidentally linking to a system routine or a system routine from erroneously linking to a nonprivileged user routine, three symbol tables are defined: privileged-system, nonprivileged-system, and user.

The privileged-system table contains external symbols defined in control sections with the privileged attribute.

The nonprivileged-system table contains nonprivileged external symbols defined in control sections with the system attribute. A further convention has been adopted: the initial entry points of nonprivileged system routines directly invoked by a nonprivileged user (such as a language processor) may begin with certain reserved characters. This has the effect of making these routines "execute-only" to the user.

With the two system symbol tables, instead of just one, the dynamic loader does not need to search a hash chain containing a large number of privileged symbols when looking up nonprivileged symbols. As will be shown, the loader does not normally reference the privileged system symbol table during system operation.

The third symbol table, constructed for the user, is primarily protective. It provides close control over the interface between the user and system routines by separating the user's symbols from system symbols.

Although the loading and protection facilities just described are quite powerful, it has already become apparent that future computer systems might require extensions to these facilities. This is currently a subject of study within IBM and elsewhere.^{5,6}

Task structure

Within TSS/360, tasks function in the environment of a large, segmented virtual store. Our knowledge of the proper way to utilize this environment evolved as the system was built and used.

Because of the large size of this address space, the need for specifically declared overlays is eliminated. This does not remove the need to plan program organization when efficient execution is desired; it merely makes it possible to minimize planning. In a time-sharing environment, where there is a premium placed upon solving a problem quickly, this added flexibility is significant and frequently desirable.

Initial virtual memory

During the initial stages of development, it was realized that certain system service routines must reside in each task's virtual store when the task is initiated (e.g., the dynamic loader). This virtual-store image would be created during system startup. As the system developed, it became apparent that efficiency could be enhanced by including a large number of other system routines in this initial virtual memory.

The TSS/360 routines that currently make up initial virtual memory include all privileged system service routines and many nonprivileged system programs, such as the FORTRAN compiler.

By tightly pre-loading most system programs at system startup, the overhead usually associated with library searches, binding, and unbinding is significantly decreased. The trade off here is time versus the auxiliary-storage space needed to hold the fully bound copy of those routines included in initial virtual memory.

Still another advantage is obtained by binding at system startup. Efficiency in a paging system is closely associated with the degree of locality of reference over a time-slice. In a highly modular system, it frequently occurs that there are groups of routines that follow a pattern such that all members of the group tend to be referenced within a short period of time whenever any one of them is referenced. Page-reference patterns associated with system programs can be significantly improved by ordering routines with an affinity for each other so that they are packed, as a group, into a minimum number of pages.

In TSS/360, this ordering is based upon a control section name list that can be altered easily to optimize the packing of system programs to minimize paging. This is especially significant in TSS/360 because many control sections are much less than a page in length.

Sharing

Virtual memory sharing in TSS/360 is utilized in three ways:

- When users share programs, they share the pure-procedure sections of the program. Each user receives a private copy of any modifiable data contained in the program.
- When users share data sets, they share a common external page map control table.
- All tasks share certain common control tables (such as the I/O device allocation table).

Program modules designed for simultaneous sharing by more than one task are called re-entrant. Such modules are characterized by their division into a shareable

control section that does not change in any way during execution and a private control section (PSECT) that contains modifiable data and address constants.

While most system programs are re-entrant modules with PSECTs, it is not necessary to use a PSECT when composing a TSS/360 program. With greater effort for special cases, it is possible to write re-entrant programs where all parameters are held in CPU registers or where working space is dynamically acquired.

When a re-entrant program is composed, all modifiable data, work areas, and address constants may be placed within a PSECT. Allowing the composer of a program to create the PSECT relieves the caller of that program of the requirement to know precisely what address constants the called program requires.

The use of PSECTs has effects upon the structure of programs within TSS/360. Whenever a user loads a shared re-entrant module, a private copy of its PSECT is placed into the user's private virtual memory, while shared access is established to a single copy of the program's re-entrant control sections. Programs are shared in such a way that the PSECTs and the re-entrant portions of the called routines are separately mapped into the task's virtual memory. Moreover, because each user's virtual memory is allocated dynamically and independently, the single physical copy of a re-entrant control section may be mapped into different virtual memory locations for each concurrent user (see Figure 3). Therefore, to perform linkage to a re-entrant routine, two virtual memory addresses must be supplied

The first address specifies the location at which execution of the program module will begin when control is transferred. This is the conventional external reference value.

The second address can be used to specify where the PSECT of the linked module has been mapped within the task's virtual memory. If this pointer were not supplied, the re-entrant module would have no way of knowing, for instance, where the appropriate private modifiable data are located since the PSECT may be placed in different virtual memory locations in each concurrent task.

Putting all address constants and modifiable instruction sequences into one or more PSECTs does not guarantee that the resulting routine will be re-entrant under all conditions. While this provides for intertask re-entrability (i.e., sharing by a number of tasks), intratask re-entrability must be considered.

A single task can re-enter the same program when it receives a task interruption while executing a system routine or when a routine is called recursively. In such a situation, the PSECT will not protect task integrity, since within a single task there is only one copy of the PSECT. This is why the Task Monitor provides either

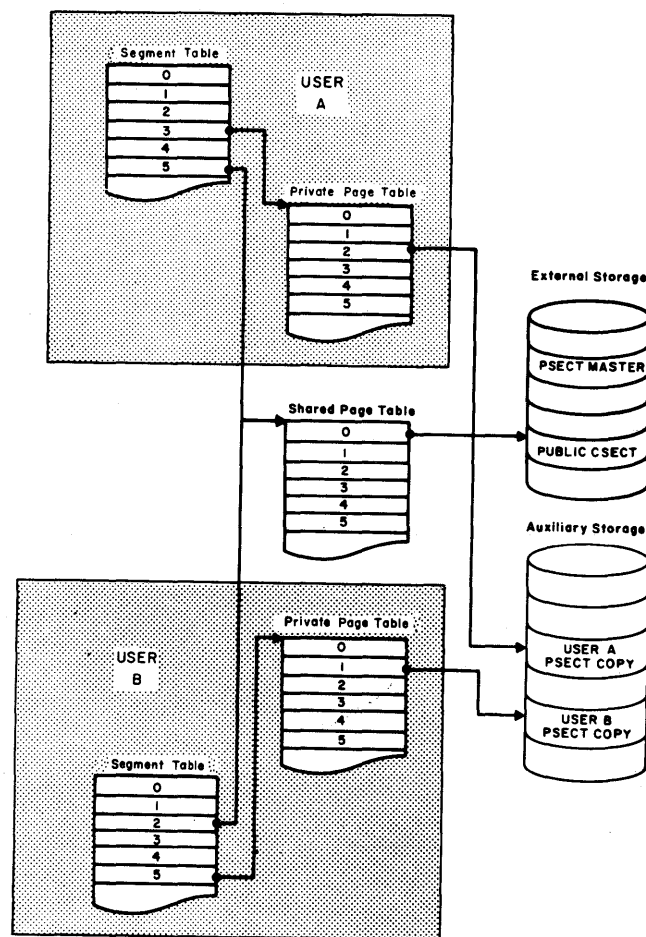


FIGURE 3—Sharing of programs in TSS/360

a push-down save-area or a means by which a routine can protect itself from unwanted intratask re-entrancy.

A PSECT is generally used to hold the register save-area for a re-entrant routine. Placing a save-area within a PSECT, rather than into a push-down stack, reduces overhead and facilitates tracing linkages during debugging.

The sharing of programs in virtual memory is based on many users actively using pure-procedure sections of the same program (such as the FORTRAN compiler) with resultant decreases in the paging overhead and utilization of main storage. Because of the amount of shared code in TSS/360, the probability that shared pages will be simultaneously used is high only for a few system routines. The primary value of shared code thus lies in its read-only attribute, which allows only one copy of a page of code to be on auxiliary storage. During the lifetime of the average task, there is a high probability that a number of users will invoke, say, the FORTRAN compiler. Thus, instead of many copies of

the compiler existing on auxiliary storage, there is only one copy of its pure-procedure sections.

When users share a data set, they share the external storage map table. They do not share buffers because there is a low probability of two or more users frequently accessing the same data at nearly the same time. Further, each user can independently modify his copy in the buffer without affecting other users.

When users share a program or a data set control table, they share a common page table (see Figure 3). This leads to a great deal of flexibility. For example, a common page-table entry can be pointed to by segment-table entries for several different tasks. However, sharing of each such page table can be restricted to specific groups of users. The way in which this sharing is accomplished is as follows:

Virtual memory service routines cannot directly address shared page tables. Therefore, the Resident Supervisor must provide a method of symbolically associating the shared item with the page table that maps it. A control table, located in shared virtual memory, serves as the repository of sharing information. Whenever a user invokes a program from a shared library or opens a shared data set, the system searches the shared data table.

Because each system user can catalog a shared program or data set using any name he wishes, the search of the shared data table is, by convention, based upon the name established by the item's owner. If the entity has not been previously referenced during the session, then an entry for this name will be created in the table.

Next, shared virtual memory is obtained for the entity. The Resident Supervisor creates the required number of shared-page-table entries and sends back the symbolic identification number of the shared page table and the location of the requested allocation within the segment. This information is stored into the shared data table. Thus, there is now an association between the name of the entity to be shared and the page table that maps the entity.

When another user invokes the shared module or opts the shared data set, a search of the shared data will yield a match on the name. The symbolic page table number can then be used in a supervisor call to request that a segment-table entry for this user be made to point to the proper shared page table.

Virtual memory sharing requires the use of programmed interlocks to prevent destructive intertask interference. The use of interlocks for sharing, however, requires careful control. For instance, system operation can be severely affected if one task sets an interlock in a system table and then becomes inactive for a long time. Furthermore, substantial system overhead is incurred if tasks waiting for the interlock to be reset are continually being dispatched only to find that the interlock is still set. This type of problem is representative of the many subtle considerations involved in the control of extensive sharing among tasks in a time-sharing environment. We are still gaining experience and insight into this aspect of the TSS/360 time-shared operating system.

REFERENCES

- 1 *The compatible time-sharing system: a programmers guide*
P A Crisman ed MIT Computation Center Cambridge Mass
MIT Press 2nd ed 1965
- 2 H A KINSLOW
The time-sharing monitor system
Proc AFIPS 1964 FJCC Vol 26 Washington DC Spartan Books
1964 pp 443-454
- 3 W T COMFORT
A computing system design for user service
Proc AFIPS 1965 FJCC Vol 27 Part I Washington DC Spartan
Books 1965 pp 365-369
- 4 *IBM system/360 time sharing system: concepts and facilities*
Form C28-2003 IBM 1968
- 5 R M GRAHAM
Protection in an information processing utility
Comm ACM Vol 11 No 5 May 1968 pp 365-369
- 6 E B VAN HORN J B DENNIS
Programming semantics for multiprogrammed computations
Comm ACM Vol 9 No 3 March 1966 pp 143-155

Considerations for software protection and recovery from hardware failures in a multiaccess, multiprogramming, single processor system

by G. OPPENHEIMER

Radio Corporation of America
Riverton, New Jersey

and

K. P. CLANCY

Keystone Computer Associates, Inc.
Willow Grove, Pennsylvania

INTRODUCTION

Data processing systems are liable to both hardware and system software failure. In first and second generation systems the impact of such failures was typically limited by the scope of the system itself to the one or limited few programs operating at the time. Resumption from the beginning of the program or preplanned checkpoint typically constituted complete recovery.

Third generation systems have increased in complexity. They have assumed responsibility for the management and retention of user files, and they support and automatically schedule the concurrent operation of increasing numbers of batch and interactive tasks. Information maintained both on direct access devices and in memory is critical to continuing system operation. Much of this information is dynamically updated in place and as a consequence even partial loss may discontinue and delay system operation. Rapid and smooth restoration of service subsequent to hardware and software failure has become a necessity. Considerations for system protection and recovery are, therefore, critical aspects of third generation system design.

The types of failure that a system must contend with include processor or memory malfunction and the unreadability of data on a direct access volume. No complete solution to all the potential problems exists. Cures for each type of

sickness are apparent, though widely variant in scope and cost. Tradeoffs exist in terms of dollars invested and overheads incurred in hardware and software procedures that might be employed. Increasing such costs can lessen the probability or localize the impact of a failure.

A processor or memory malfunction may terminate the operation of the system abruptly. Salvaging the information in memory under such abortive circumstances tends to be extremely complicated and time consuming, if feasible at all. Processing cannot be resumed from the point of failure. All tasks in process must generally be prematurely terminated and restarted, and some data in files that were being modified may not be retrievable. The second section of this paper is devoted to a discussion of this topic.

A major volume related failure, such as a head scoring the recording surface, could destroy a significant amount of data. Without some means of protection, a loss of this magnitude could effectively cause total loss of the system. This subject is pursued in section 3.

A minor volume related failure such as the unreadability of a single track of data could cause the loss of one or more files, or could prevent the initiation of accepted tasks or prevent the completion of one or more initiated tasks. Preventative software techniques are helpful, but limited. Online procedures to monitor and identify mar-

ginal performance of the processor, memory and I/O operations may warn of imminent failure or permit avoiding the use of marginal elements. For example, a routine which vigorously exercises the tracks of a volume and inhibits the use of marginal tracks will significantly reduce the probability of track unreadability.

Such techniques reduce the probability of failure. They do not prevent failure. Consequently, provisions for recovery from failure are still essential. In sections 4 and 5 this area is considered in terms of system files and user files, respectively.

At the current state of the art it is not feasible to consider system recovery procedures independently of the structure and organization of the system to be protected. Consequently, the orientation herein is directed toward techniques considered for the protection of the time sharing operating system on the Spectra 70/46. Particular emphasis is placed upon the system and user information retained on online direct access volumes.

The relevant system characteristics which bound the areas of discussion are:

- . a single central processor, employing a single memory bank.
- . direct concurrent access to the system by multiple interactive users.
- . user program and data files, as well as system information, maintained on mass storage. Allocating of file space as well as file access and retrieval are fully managed by the system. File size is essentially unrestricted.
- . task (workload) control fully managed by the system. Tasks submitted to the system are maintained on online devices and scheduled by the system for multiprogrammed operation concurrent with interactive processing. Output is directed to printers and punches by the system based upon resource availability.

System restart after a memory loss

In systems employing hierarchical control memories and micro programmed processor logics, memory failure tends to produce less than an orderly system failure. Recovery of memory information under such circumstances may, in fact, be impossible. Similarly, in systems without motor generator sets, the abruptness of power failure typically makes it impossible to recover the information in memory. Recovery under these circumstances tends to reduce to making the system usable again. Automatic restart of the work in

progress at the time of failure is not feasible. Terminal users must reconnect to the system and background tasks must be resubmitted after update-in-place files that were being modified have been reset to an original or checkpoint state.

One approach to system restart after a memory loss is to periodically record the status of the system (e.g., all the information on all the direct access volumes) when no one is using the system. While this permits re-establishing the system at the chronological point at which the status was recorded, it requires repetition of all file modification subsequent to the reset point and re-entry of all workload awaiting initiation at the time of the failure. Such an approach places a major burden of recovery on the users of the system.

An alternative method is to re-establish the utility of the system by correcting system status information. This information reflects status for tasks in process at the time of failure. It is invalid at restart time in view of the necessity to initiate all processing anew. The need for such a corrective procedure is dependent upon the probability of memory loss. Its complexity is determined by the extent of the dynamic control information that is maintained in memory as contrasted with that maintained on direct access volumes.

Information necessarily in memory includes the precise status of tasks actually initiated, and file content and file structure modifications actually in process.

Information not subject to such dynamic updating and consequently, logically allocatable to either the memory system or direct access volumes includes:

- . file accessibility information.
- . direct access space allocation information.
- . system workload information.

Considerations for the protection and recovery of the above three categories of information are presented in the following discussions.

File accessibility information

The files in the system may be accessed by multiple users on a password restricted basis. Input access may be granted to any number of concurrent users. Output and inplace update access is restricted to one user at a time.

To control access to sharable files, the system maintains the status of file usage for each opened file. Indications of granted output or update access and of the number of input users are employed to

prevent interference among simultaneously incompatible accesses. If this information is maintained on a direct access volume, system restart procedures must reset these indications. Otherwise files being accessed at the time of failure would be incorrectly restricted. If this information is maintained in memory, the reset procedure is unnecessary. Moreover, fewer updating accesses to the volume would be required during normal system operation.

In addition to the tradeoffs of processing efficiency and memory space usage, the restart consideration is involved in the placement of the indicator information. It appears highly desirable that the system identify files that were being created or modified at the time of failure. These files may require special processing before a valid program restart may be initiated. File content may be imprecise. Records modified or created and in the output process at the time of failure may not be reflected completely or correctly on the volume. Moreover, the loss of memory contained file structure information could prevent retrieval of the entire file. Advising subsequent users of the file would appear to be a minimum requirement upon the system.

Hence, if usage indications are maintained on a direct access volume, the system restart procedure should search file directories, reset the indicators and establish flags which will notify subsequent users of the potential need for specialized file correction procedures. If the usage indications are maintained in memory, some auxiliary method of identifying the files in the process of modification at failure must be devised.

Direct access space allocation

Space allocation is typically maintained on direct access volumes to avoid memory loss and, in some systems, to permit full utilization of removable disc packs. Such information may not accurately reflect the allocation if failure occurs while it is being modified. Moreover, the actual file bounds within an allocated space may not accurately reflect the additions which were being created at failure. To attempt to reconstruct such files and re-establish the space allocation properly is extremely complicated, and appreciable delay in restarting the system may be anticipated. However, since it is reasonable, per the previous section, to identify all files possibly requiring special processing, it also seems reasonable that the allocation information be assumed correct. Thus,

questionable file marking may be employed to warn of potential allocation problems as well as advising users of potential data loss within a file.

System workload information

Tasks queued on direct access volumes and awaiting system initiation may be processed just as though the system was being restarted after a normal shutdown. Tasks which had been disrupted by the failure, however, cannot be re-initiated. Before such tasks may be restarted, update files must be returned to the status they possessed at a previous checkpoint. Since it is not possible to economically retain knowledge of which tasks had actually performed such file modifications, all tasks which had been initiated must be purged from the system by the system restart procedure. Appropriate notification must also be given to the operator so that file resetting and restart may be properly initiated.

If in controlling output print and punch operations, the system retains all data on direct access volumes until the entire operation is completed, these operations may readily be restarted subsequent to a failure. Restart of an operation from an intermediate point may be desirable in certain instances, such as printing. Such restart requires special operator communication.

Summary of memory loss considerations

In guarding against memory loss, the appropriate use of direct access storage for control information and data provides an economical method for protection. It limits the impact of failure to the actual tasks and files in process. Execution of such file correction procedures, as described subsequently, and restarting of the related tasks, appears to provide an adequate level of protection.

A minimal protection scheme would therefore involve purging of in process tasks, warning file users of suspect file structures, restart of print and punch output, and normal processing of tasks awaiting initiation.

Major volume loss

The preceding section described procedures which a system could employ to resume operation subsequent to the loss of information in memory. Such procedures are heavily dependent on the availability in direct access store of pertinent control fields. Failures of the direct access store

therefore are of critical concern in the design of system recovery procedures. This section addresses protection and recovery from a major direct access volume loss.

Head scoring of the recording surface, extraneous output signals generated by device control electronics, or major software errors could destroy a significant portion of the information on a volume. Recovery from such a catastrophe is dependent largely on how direct access space is allocated. If critical system information is restricted to a single volume, it is possible to reconstruct the volume. This requires making a copy of the volume contents at the close of each processing period and maintaining (on a different volume) records reflecting all significant changes to the volume. Similarly, if each user file is allocated to a single volume and a volume catalog is maintained to identify all files on the volume, it is possible to retrieve backup copies of the files to re-establish the destroyed volume. Alternatively, in this instance, a full volume copy of the information on the volume could also be used to recover from a volume loss.

Many systems permit file information to cross volume boundaries in order to obtain more flexible system performance and better overall utilization of the available space. As a result, individual volume copies and retrieval of backup files becomes less practical. A full volume copy tends to be useless as it will, if used, result in inconsistencies within the files that reside on more than one volume. Moreover, the retrieval of backup copies for all the files allocated on the volume may involve a large number of files and require significant space allocation and deallocation on unaffected volumes.

Fortunately, full volume loss tends to be a rare occurrence. Therefore, a reasonable approach, for systems that permit files to cross volume boundaries, seems to be to use a copy of the full set of direct access volumes as a system reset point.

System file considerations for minor volume related failures

The unreadability of information stored on direct access devices is significantly more probable than entire volume loss. To protect against such loss, the software may perform preventative marginal track tests. To accomplish this requires a routine which vigorously exercises the tracks of the volume and inhibits further use of marginal

tracks. Forming a temporary copy of the volume and reloading the copy data completion of the exerciser may significantly reduce the number of unreadable tracks encountered during system operation.

Regardless of the use of such preventative functions, consideration must be given to the protection and reconstruction of critical system files. In this section protection and recovery techniques for the following system files are discussed:

- . user identification information.
- . resource usage information.
- . file related information.
- . volume space allocation information.
- . system workload information.

User identification information

To identify legitimate users of the system the administrator enters pertinent information into a file. Before processing any task submitted, the system interrogates this file to verify that the user has been authorized access. Typically, records of this file include user identification, an access password, permissible account numbers, the highest priority he may use, and the amount of file space he is permitted to use and has used to date.

With the one exception of the permanent space usage field, every change made to this file is initiated by the administrator. Each such change is also recorded in hard copy form at the administrator's terminal. Therefore, protection of this file may be readily achieved by periodically recording a backup copy. In the event of file loss, the administrator may retrieve the backup copy and reapply the few changes he has made since the copy was made. The changes to the space usage field, which are not directly available to him in hard copy form, may be retrieved from the resource usage file and applied to the re-created version.

Resource usage information

An accounting file is normally maintained to serve as the basis for charging for the use of system resources. Dynamic posting of usage to this file is performed as tasks terminate processing.

The types of information included in this file are the user identification number, the applicable account number, a task sequence number, time stamps for task entry into the system and for task initiation and termination, priority level

exercized, processor time used, memory space used, increment of permanent file space used, maximum temporary file space used, number of I/O orders issued, and number of types of private devices required.

Since this information is both critical and low in volume, it may be dually recorded on separate volumes and the two copies may be retained on-line. Alternatively, some reduction in system overhead may be effected by maintaining a single base copy and recording a supplementary file. In systems where accounting transactions are merely appended to the base file, the supplementary file will contain the identical records posted to the base copy. Where accounting transactions are used to update entries in the base copy, the transactions are typically placed in the supplementary file. If the base file is copied periodically to create a backup, the supplementary copy will occupy less space and need be maintained only between the periods in which copies are made. To achieve still greater protection, each supplementary file may be printed before it is destroyed.

File related information

Systems employing removable direct access volumes typically maintain file related information in a file directory or catalog, and also in volume related labels. Catalog entries include the filename, creation date, access passwords, access restrictions, and possibly, current access (usage) indicators. Also included are fields indicating the number of volumes and the actual volumes on which the file resides, and optionally, the location of the backup copy. Volume related labels describe areas of the volume allocated to the file and, for index sequential type files, the related control indices.

A degree of protection for the catalog may be achieved by periodically producing a backup copy. However, reconstruction of a catalog from such a copy may prove to be incomplete and erroneous. If a file is modified or moved subsequent to the creation of the backup, the volume information within the copy may be undetectably erroneous. Subsequent allocation and deallocation could present a different set of volumes or a different ordering among the same volumes. Verification checks might be employed to minimize, but not eliminate possible ambiguities. Moreover, changes to certain information such as access limitations and file type could not be reconstructed.

A more satisfactory approach requires maintaining a log of changes made to the catalog. The use of such a log in conjunction with a prior copy of the catalog permits accurate and complete reconstruction. The system user is affected only if he should attempt to access the file before reconstruction is accomplished. In general, this means that he may not be able to complete some processing, but that he is able to resubmit and resume his processing at a later time with a minimal potential loss and with no responsibility for catalog reconstruction.

Protection of the volume related labels presents a somewhat different issue. Loss of a volume label may make it impossible to determine the areas on the volume in which the file is located. In general, this means that it is impossible to read the file. Information relative to a backup copy should still be available and it may be retrieved in lieu of the copy on the direct access volume. Alternatively, an analysis of the volume labels of the other files on the volume and of the unallocated volume space information could be used to determine the space that was allocated to the file in question. This information could then be used to read the appropriate areas of the volume and to retrieve the file. The sequence of the data would then have to be verified and reorganized by the owner. Both logging procedures and dual recording of the labels would provide for more comprehensive protection, but at a significant increase in overhead.

Volume space allocation information

Unallocated volume space information is typically maintained on each volume. Loss of this information makes it impossible to safely allocate additional file space on the volume, but does not affect the retrieval of existent files. Protection of this information does not seem to be warranted. The file space allocated to each file on the volume may be determined, using the file labels on the volume, and the unallocated space may be deduced. A more basic approach would copy all files allocated to the volume and reinitialize the volume. However, in systems where files cross volume boundaries significant space allocation and deallocation will be required on otherwise unaffected volumes.

System workload information

The tasks representing the system workload are generally identified by entries in various queues.

Normally, a single system routine maintains the queues and the file space in which the necessary information is stored. Proper maintenance of these queues is so critical to continuing successful operation of the system that dual recording is typically justified. If appropriately considered in designing the system it is possible to accomplish the required level of redundancy without the cost of total duality.

An approach to the appropriate level of redundancy is achieved by dividing the relevant file space into index and task areas. In the index area, there is one entry for each task, each entry containing a task sequence number, an indicator identifying current status, a priority indicator, resource requirement information and a pointer to a task header block. Similarly, in the task area, there is a task header block for each task containing the detailed information necessary to initiate a task, interim resource and time stamp information and a pointer to the file which contains the information required to process the task.

The loss of a track containing indices could result in the loss of information necessary to initiate or account for a significant number of tasks. The loss of a track containing task header blocks could result in a similar loss for the tasks involved.

The inclusion in the task header block of a backward pointer to an index entry makes it possible to identify the task header blocks associated with an index. With this information an index which is unreadable may be reconstructed. The backward pointer identifies the task header blocks associated with the lost index and the information in the header blocks and the file to which it points is sufficient to reconstruct the index.

An additional pointer in the index, identifying the file to which the header block points would similarly make it possible to reconstruct lost header blocks. However, the addition of a pointer to a file, which may be a rather lengthy filename, significantly reduces the number of header blocks that may be stored and consequently affects the efficiency of the system.

The absence of the file pointer in the index block implies that loss of a track of header blocks requires these tasks to be resubmitted, if they are to be run. The operator can be apprised of the tasks involved to facilitate this action, since the sequence numbers are available in the index. The choice here, as so often, is one of efficiency versus full protection by the system.

Summary of minor volume related failure considerations for system files

System files contain information relating to the entire user community. Loss of a single track of data from one of these files could result in impaired service and significant inconvenience to many users. A loss of user identification information could result in denial of service to all users affected. Loss of resource information might be reflected in lost revenues, loss of file related information could cause the loss of many files stored within the system, while loss of system workload information might impose appreciable delays and rework of tasks accepted by the system. Mechanisms to minimize the impact of these losses have been discussed above. The techniques cited range from manual administrator adjustment to full redundancy.

In the following section the subject of protection and recovery is developed from the point of view of individual user files.

User file protection

Increased utilization of direct access devices have enhanced the operation of third generation systems in a number of ways. Direct access volumes and files may be concurrently shared by more than one user providing the user community as a whole with greater availability and utility of the system. Files, and low activity files in particular, may be updated on a record basis and the need for recopying of entire files is reduced. Such updating in place is in many situations the only feasible approach to updating large files with the frequency required by some applications.

File shareability has introduced the need for accessibility procedures alluded to in preceding sections. Update-in-place has similarly increased the need for more sophisticated protection procedures. File update procedures employing third generation access methods affect both the data content of the file and, in the case of non-serial organizations, the very structure of the file. Should a memory failure occur while a file is being updated, consideration must be given to reconstruction of both file content and file structure. Reinitiation of an aborted task or resumption of it from a checkpoint established prior to the failure will otherwise prove unsatisfactory. Additionally, the loss of any system control information identifying the file or the inability to read a track of data within the file may cause the loss of the entire file.

In this section protection and recovery of user files is developed in terms of consideration for:

- . file content
- . file structure
- . protection by file copy
- . file reconstruction procedures

Certain of these considerations fall within the province of system responsibility, while others may be achieved effectively only by the application programmer.

File content

Typical magnetic tape system recovery procedures entailed either reinitiation of a task with a new set of output or "update" tapes or repositioning the tapes to the position they held at the time of a prior checkpoint and resumption of execution from that checkpoint. This is not sufficient in direct access update-in-place operations since processing performed subsequent to a checkpoint and prior to a failure will have updated records. Resumption from a checkpoint will involve "double-updating" of these modified records if corrective procedures are not employed. To illustrate, if the file processing involved updating the accounts receivable, activity such as subtracting a payment or adding the value of a new shipment must be undone before the transaction may be reapplied. Alternatively, procedures may be developed to avoid redundant application of the transaction.

File structure

Direct access device file organizations have become more sophisticated. Typical magnetic tape systems possessed only a serial organization wherein the primary structural consideration was the end of recorded information on tape. Since typical procedures merely repositioned the tapes to their location at a prior checkpoint, the difficulty of detecting the true end of data was avoided by subsequently executed recovery procedures. No greater difficulty is encountered in reconstituting a serially organized direct access file.

Linked file structures, and partially linked structures such as index sequential, are often considered to be immune to the effects of failure since allocation information and file sequence are embedded within the file itself. However, the manner in which the next available record position is maintained, for example, is extremely critical where insertion or deletion, or size modification of records is permitted. Typically, the

next available record position is maintained in memory for efficiency purposes. Subsequent to a memory failure, the only value available is that from the volume. This value may be significantly out-of-date. Utilization of an obsolete value may cause the improper reuse of an allocated file section, destroying records and aborting link sequences. Similarly use of an obsolete value from a checkpoint can cause the file to be impacted by the pre-restart file correction procedure. The seriousness of the situation is increased in environments where records completely unrelated to the user's file transactions may be placed in an overflow area to make space available for an insertion or space modification. Use of an obsolete value of the next available record position in this case may result in the undetected loss of inactive records.

Protection by file copy

A variety of protection procedures are possible, depending on file size, organization and frequency of modification. In a number of instances, the file to be processed may be small enough to fit into the memory system. This is particularly the case when the file is to be manipulated by an interactive language processor or file-editing processor. Indeed, some systems specifically constrain files to such a size. In this environment, copying the file into the memory system prior to processing provides file protection against memory loss or software failure during the update process. At worst, the processing which immediately preceded the failure must be repeated. In addition, since the updated file copy is returned to the device only when a process is successfully terminated, the application programmer is afforded the luxury of experimenting with a file without invalidating its contents or structure.

In the general case, protection against file loss still implies maintaining a second copy of the file, regardless of its size. In the extreme, this could be taken to mean that the file should be dually recorded on line. The cost in space and processing time does not normally warrant such a procedure, and for the case of memory loss, it is reasonable to expect that two copies could be lost as readily as one. A safer and less costly procedure is to copy the file after each processing session in which the file is modified. This provides two current copies of the file, one of which may be used as a current backup in the event any loss should occur in the other.

The space required to maintain backup file copies presents a different set of considerations. The lower access time, higher transfer rate direct access devices are best used for frequently accessed files. File copies generally do not fall into this category and typically need not be on-line at all. If a system configuration includes mass storage equipment of sufficient capacity to contain such backup copies, the space problem is essentially eliminated.

For such a configuration, systems should be equipped to readily locate and maintain files in the mass store for retrieval to the faster, normal working store. System commands and procedures for such file transfer operations provide essential support to file recovery procedures.

In systems without mass on-line storage, the capability to copy a file to tape becomes an equivalently important system feature. A standard copy function, however, tends to underutilize the capacity of magnetic tape reels and to burden the application programmer with the management of multiple, unrelated tape reels or single tape reels containing sets of unrelated files. To offset this, a system function may be provided to manage backup tapes for all users. Commands may also be provided to request creation or retrieval of a backup copy and the system should manage the backup library while ensuring file privacy by prohibiting direct user access to the library.

File reconstruction procedures

To make a copy of a file each time it is modified reduces the value of updating the file in place. Certainly the reduction in I/O operations achieved by only reading and writing records which are to be modified is totally negated. However, a gain may still be achieved in this regard since it is not necessary to copy a file every time it is modified. Rather, it is possible to copy it after some time period or after some number of modifications have been made. Periodic copying of a file is almost a necessity on larger, low-activity files. In situations where such an approach is taken, consideration must be given to file reconstruction procedures, more specific to the file and application than the general file copy procedures previously discussed. Procedures specifically concerned with file content may typically be planned best by the application programmer. File structure considerations are more specifically within the province of the system.

Application program techniques

Subsequent to a failure resulting in the loss of information in memory, typical recovery procedures entail a resumption of processing from a checkpoint. Difficulties arising from redundant pre-failure and post-failure application of transactions may be avoided either by reconstituting relevant records to their status at the time of checkpoint or by eliminating the redundant application of the appropriate transactions. The feasibility of the particular approach is heavily dependent upon the actual application. Accordingly, although the system may provide the application programmer with tools which will facilitate his development of appropriate procedures, it is generally not feasible for the system to undertake the full procedure.

To re-establish a file so that it contains precisely the same data as when a checkpoint was taken, it is necessary to maintain an auxiliary file which is synchronized with the checkpoint. Prior to updating a record, the unmodified input image of the record is recorded in the auxiliary file. A file of such "pre-images" is thus constituted. When a new checkpoint is established, the auxiliary file may be erased and a new series of pre-images supplementing the current checkpoint may be initiated. Subsequent to a failure, the pre-image file may be read in reverse order and used to return every record to its status as of the related checkpoint.

Alternatively, procedures to avoid redundant application of transactions, would either mark effected transactions or store effected transactions in an auxiliary file analagous to that above. Unfortunately, such procedures are not perfect. If transactions are marked before the primary record is updated, a failure before update will cause the transaction to be eliminated; while if the primary record is updated prior to marking the transaction, an intervening failure will cause the transaction to be applied a second time.

Subsequent to the loss of data in a file or the inability of the system to retrieve the file because of the loss of some vital control-information, the user must generally rely on a backup copy to recreate his file. If a new backup copy is not made each time the primary file is modified, it is necessary to augment the backup copy with an auxiliary file. This auxiliary file could contain either the transactions which were used to modify the primary file or the after image of the records that were modified subsequent to recording the backup copy. With an auxiliary transaction file,

the reconstruction must essentially repeat the update processing for each transaction to produce a current file. With an auxiliary file of after images, the processing required entails only the application of the after images to the records of the retrieved backup copy.

System reconstruction considerations

File content recovery considerations are peculiar to the application and, accordingly, they are left to the application programmer for solution. File structure considerations require the retention of information typically unavailable to him. As a result, procedures, if any, for this latter purpose require system functions. To illustrate, a system function could be employed to reconstitute the structure of the linked file discussed previously. With full knowledge of how the space is allocated to the file, a special procedure could be employed to retrieve records from the suspect structure and create a new linked file. Such a procedure would utilize the link addresses to retrieve all data. Since the suspect file would be employed for input, concern for unavailable file structure information is irrelevant.

Procedures such as the above are highly dependent on actual file organization and typically require reprocessing of the entire file. Therefore, it appears that file copy techniques are both more feasible and more basic.

Summary of user file protection considerations

The need to safeguard both the data within files as well as the information relating to the organization of the files has added to the complexities of file protection in third generation systems. File update-in-place has transformed the creation of backup-files from an automatic by-product of

the update process to a planned design requirement. These additional responsibilities must be shared by both the system and the application programmer.

SUMMARY

Modern data processing systems require software protection and recovery. Nevertheless, the virtually limitless possibilities of malfunction cannot be completely managed by a software system simultaneously required to provide efficient performance. Necessarily, the uncertain probability of the occurrence of malfunctions and their impact on continuing system operation are most significant parameters in the decision that must be made to include or exclude each specific protection mechanism.

In addressing itself to the protection of information on direct access volumes, this paper has considered the impact of memory loss, volume loss and track loss and described factors weighed in the design of a given system. Preventative tools such as track and memory exerciser routines have been mentioned. Dual recording, backup file procedures, transaction logs, dual recording of updated records prior to and after update, and redundant recording of selective information have all been

The impact of recovery and reconstruction procedures on the system has been reviewed and the often ignored necessity of sharing the burden of recovery with the system administrator, the system operator and the system user indicated.

There is no single solution to the myriad problems that exist. Nor will the problems vanish until the Utopia of hardware and software perfection is reached. In the meanwhile, progress can only result from a solid facing of the problems and a judicious weighing of the practical solutions.

Error recovery through programming

by ALAN N. HIGGINS

International Business Machines Corporation
Kingston, New York

INTRODUCTION

The requirement for error recovery procedures has existed as long as computers themselves. Since the earliest computers, one of the goals of design has been to increase the reliability and availability of the computer to the user. While great strides have been made in this direction, the need of error recovery is still as present today as ever and at this time, the need is actually amplified and more pressing than ever before.

With the many advanced techniques in programming such as multiprogramming and multiprocessing, the cost of an error has increased dramatically so that no longer are the consequences of an error limited "merely" to the loss of a job and the imposition of the need for a subsequent rerun.

Error today can:

- Cause the termination of concurrently executing tasks.
- Cause an environmental control system to go down.
- Cause the loss of teleprocessing messages.
- Cause the generation of a report to be delayed

No longer can rerunning the job be accepted as a prime means of "error recovery. The situation existing when running under an Operating System, and executing a number of jobs in the computer at the same time, makes improved error recovery procedures mandatory.

It is recognized that the Engineering Community is diligently striving to improve the hardware itself and thus for a complete solution it is necessary to look at the other half of the question of error recovery—what can be done to improve reliability, to improve availability, to improve error recovery through programming?

In order to do this, we have to first consider in a general way error recovery procedures or Recovery Management Support. The next step is to look specifically at some of the work which has been done in Operating System/60 with the Recovery Management Support for the Model 65.

System incidents

An examination of system incidents reveals that such incidents are due to a number of sources. Among these are Hard Core errors (including errors in the CPU, memory and channels), errors from Input/Output devices and control units, procedural and operational errors. Each of these is made up of a number of different errors but from a gross point of view, it seems reasonable to state that there are three general types of system interruptions:

- Hardware malfunctions.
- Design errors (both hardware and software).
- Operator or user injected errors.

Systems planning must therefore be influenced by the facts that machines will malfunction, neither hardware nor software is perfect and that operators are still likely to make as many mistakes as they have in the past.

Recovery management

The primary objective in any error recovery procedure or Recovery Management Support should be to alleviate the burden of system interruptions to the user. In order to accomplish this we must:

1. Reduce the number of interruptions to which the user is exposed and,
2. Minimize the impact of these interruptions when they do occur.

Recovery Management therefore should provide the user with a higher degree of system availability (more time for more jobs) by minimizing the impact of system malfunctions upon his operations.

With this objective as the target, error recovery takes on a broader meaning and scope than has been applied to the concept in the past. In an environment of multiprogramming, the system becomes all important and it is most necessary that no matter what happens, the system must continue to function. It often becomes a situ-

ation of sacrificing a part so that the "whole" may survive. In order to accomplish this, Recovery Management facilities may follow a pattern similar to one where the support attempts to reduce the number of system interruptions by retrying the operation which was interrupted by the malfunction or it may terminate the task affected and continue system operation. If this is not possible, then the second step toward accomplishing the primary objective of error recovery becomes of paramount importance—to minimize the impact of the interruption. This is done by preparing the system for a simple restart or it may indicate that repair by maintenance personnel is required.

Instruction retry.

This pattern, which has just been outlined, suggests a number of functions which can be performed to achieve the objectives of Recovery Management. The first of these functions is instruction retry. The concept of instruction retry is not really new. It is something which IBM has been doing for years, particularly in the I/O area. Instruction retry has been standard procedure whenever an error was encountered in reading or writing a tape. But it is possible to extend this retry capability and to employ it when a CPU or memory malfunction occurs.

A relatively large number of malfunctions are intermittent in nature rather than solid failures and therefore, there is a high probability of success of execution and recovery if an instruction retry can be attempted. The first thing which must be determined then is whether instruction retry is feasible and then if feasible, to execute the retry.

The determination of instruction retry feasibility is usually quite dependent upon the characteristics of the particular machine. Ordinarily for feasibility to exist, the "environment" of the computer must be valid or free from error. Dependent upon the specific machine, this may include the data contained in general purpose registers, floating point registers, machine log-out areas, permanent storage areas, etc. Arbitrarily, the criteria of validity can be keyed on parity. If the parity of the data is good, the environment is assumed valid and therefore retry is feasible. If parity is bad, then no further retry action can be taken.

Having ascertained that instruction retry is feasible, it is necessary to continue the analysis and determine if a specific instruction is retryable. To do this, it is first necessary to locate the failing instruction. The procedure involved here is again dependent upon the particular machine and what type of fetch or pre-fetch logic is employed and whether or not the instruction counter is accurate. In one case, a comparison of the internal registers in the machine log-out can provide the clue as to

whether the instruction counter is accurate; in another it may be a function of when the machine check occurred and what updating cycles the instruction counter was executing at the time.

It is obvious, therefore, that it is not always easy or possible to locate the failing instruction but if the instruction counter is accurate and it is possible to locate the failing instruction, an analysis can be performed to ascertain whether the retry threshold of the interrupted instruction has been exceeded. (The retry threshold is that point in the instruction cycle after which retry cannot be attempted and is usually indicated by a bit set by the hardware.) The retry threshold has been exceeded when during the normal instruction cycle one or more of the original operands has been changed. If the threshold has not been exceeded, it is possible to cause another attempt at executing the failing instructions. If, however, the threshold is exceeded, it may be possible to extend the threshold by examining the instruction type to determine whether a copy of the original operand might still be intact in some internal register and if it is, by restoring it. This is accomplished by rebuilding (in a special execution area) the instruction from the contents of the log-out or the internal registers or main storage.

Therefore, from an analysis, it is possible to determine that an instruction is either:

- 1-Retryable, that is the retry threshold has not been exceeded or if it has been exceeded, the damaged operand can be restored and therefore instruction retry can be attempted or
- 2-Non-retryable, that is instruction retry is not possible because either the threshold has been exceeded or the damaged operand cannot be restored, an invalid environment exists because of incorrect parity or the value of the instruction counter is indeterminate. If the second condition is the case, then it is necessary to look for another way to handle the error recovery.

Refresh main storage

The occurrence of a parity error in main storage obviates instruction retry therefore, one function which could be of value would be the ability to "Refresh" main storage. By this is meant to repair the damage which either caused or was caused by a malfunction by loading a new copy of the affected module into main storage. (A module is a program unit that is discrete and identifiable with respect to compiling, combining with other units and loading.) The use of refreshable code requires a good deal of foresight in coding since in order to be refreshable, a module must not modify itself or be modified by another module; for example, it must not set switches,

contain dynamic storage areas, or store registers or address pointers within the body of its code. The foresight is well rewarded, however, when it is possible to load this refreshable code and then continue execution without changing either the sequence or the results of the processing.

The attribute "refreshable" is similar to "reentrant". Most reentrant modules meet the requirements specified above and in addition, a reentrant module is one that may be utilized by more than one task at a time (some modules classified as reentrant deviate from these requirements by operating in a pseudo disabled manner, thus actually allowing modifications during a short period of time). The difference between the two is that "reentrant" is based on the operational characteristics of the module within the system while "refreshable" is based only on the fact that the code is not modified in any manner.

Selective termination

The functions of instruction retry and refreshable code are most desirable since they render the error recovery procedure transparent to the user and require no intervention on his part. Unfortunately, it is not always possible to attain this level of recovery. When this is the case, it is necessary to accept some degradation in order to keep the system operational. One way to accomplish this is to implement a function of Selective Termination. Such a function would enable the system to examine the failing environment, determine what problem program was executing and then proceed to terminate this program while continuing all other jobs which were executing at the time of the malfunction. This is really a type of job-abort which frees the resources of the system allocated to the job and makes them available for future use. If a problem program was utilizing system code when the malfunction occurred, selective termination could be effective if the system code was transient rather than resident in nature. This process results in the loss of a specific job but it does enable the system to continue without interruption.

Another function which would aid in the error recovery process when a memory malfunction occurs is the ability to logically carve out or remove that portion of the memory in which the malfunction occurred. Since this type of error recovery would result in job termination and might not return resources (Storage, I/O devices, etc.) to the system, such a procedure would obviously introduce undesirable side effects, such as loss of availability of I/O devices, loss of part of core and, loss of the terminated job, but it would preserve the system and operation would continue until an orderly correction could be made.

I/O Recovery

The functions which have been discussed so far have been directed mainly to errors which occur in the CPU or memory. From an examination of system incidents, it is evident that a significant portion of errors occur in the I/O area. Is there anything which can be done to improve error recovery procedures for I/O?

In the first place, there is I/O retry which is available through the ERPs (Error Recovery Procedures) for the different I/O devices. As indicated earlier, it has been standard procedure to retry I/O instructions when errors occur. A number of errors (unit check, unit exception, wrong-length indication, protection check and some chaining checks) can be corrected by this means. An I/O Supervisor performs an analysis and selects, according to device, the proper ERP to attempt recovery. After retry is attempted, the ERP regains control to determine whether or not the retry has been successful. If it was successful, the I/O retry is transparent to the user.

There is another group of I/O errors—channel checks (channel control check, channel data check and interface control check)—which need not be disastrous but which after analysis of the conditions causing the error, it may be possible to recover. Such an analysis would determine the type of operation that failed, the type of device affected, the sequences which occurred across the I/O interface following the error and whether a retry can be attempted.

The I/O device or medium can malfunction and if a retry is not successful, there may be other ways to continue the execution of the job. One such way would be to have the ability to switch data sets (devices), that is to change a tape or disk pack from one drive to another and then to retry the operation with the new drive. Another possibility (if the malfunction was really related to the Channel or Control Unit) would be to try another route to the same device. In this circumstance it would be an attempt to use the device by accessing it through a different route, that is by addressing it through a different channel or control unit.

Other system incidents

Another group of system incidents is due to procedural and operator errors. Several things can be done to decrease this and as such, it certainly deserves concentrated attention. The first is, of course, better trained personnel but from a programming point of view, several possibilities exist. It is most desirable to require a minimum of user intervention and interaction in order to accomplish execution. Control information should be minimal. When interaction is required, messages should be clear and concise — to the point of outlining

possible choices. A conversation mode could be optional which would permit correction or confirmation of operator action. All these points are generally grouped under a concept of Operator Awareness and have a very definite place in the planning of any error recovery support.

All of these functions are aimed at continuing the operation of the system but unfortunately this is not always possible to accomplish. Therefore, the next best thing is to minimize the effect of the malfunction. This can be done by attempting to preserve information concerning the malfunction and to make it available to assist knowledgeable personnel to determine what caused the error and what can be done to correct it. This will have the most desirable effect of shortening the Duration of the Unexpected Interrupt and get the system back in operation as quickly as possible

RMS/65

The Recovery Management for the System/360 Model 65 (RMS/65) has provided a number of these functions in the operating system. These functions are contained in two programs which make up RMS/65. These are the Machine Check Handler (MCH) which is directed at CPU and memory malfunction and Channel Check Handler (CCH) which is oriented to I/O problems. The RMS/65 has provided a hierarchy of recovery which involves four levels:

- I. Functional Recovery
- II. System Recovery
- III. System-Supported Restart
- IV. System Repair

Functional Recovery is the successful retry of an interrupted instruction. MCH handles the operation for the CPU and main storage through its Machine Analysis and Instruction Retry (MAIR) facilities. The MAIR facilities perform an analysis of the machine environment at the time of the machine check interruption to determine the feasibility of retrying the interrupted instruction. MAIR then retries the interrupted instruction when retry is feasible. The CCH performs the analysis function for the channel checks discussed earlier. This is accomplished by intercepting I/O interruptions before the I/O Supervisor receives them and performing an analysis of the existing conditions. If feasible, the status bits are manipulated to make the channel check look like a failure for which ERP exists and then control is transferred to the appropriate ERP for action. Functional recovery is of course the desired goal because in this case the malfunction is transparent to the user.

System Recovery is the second level of recovery and is required when functional recovery is either not feasible or fails. The objective is to preserve the system and

to continue processing all unaffected jobs. This is done by means of a Program Damage Assessment and Repair feature which attempts to analyze the malfunction environment, to isolate and repair the program damage if possible and to report permanent failures to the program and operator. This feature also incorporates the mechanism to provide the capability of selective termination of a task.

The function of System-Supported Restart is called on when both Functional and System Recovery have failed but a stop for repair is not required. The operator is informed that such a condition exists and that it is necessary to restart the system.

The fourth level of recovery support provided by RMS/65 is System Repair. In a way, this is perhaps one of its most important functions since the detailed error analysis information which is provided can be of great assistance in the determination of the cause of failure and in suggesting the proper correction for the problem. Once the repair is completed, initialization is required to restart the system.

Figure 1 shows the relationship of these levels of recovery to one another and to the main objective of Recovery Management Support which is to keep the system in operation.

Each level of recovery performs the important func-

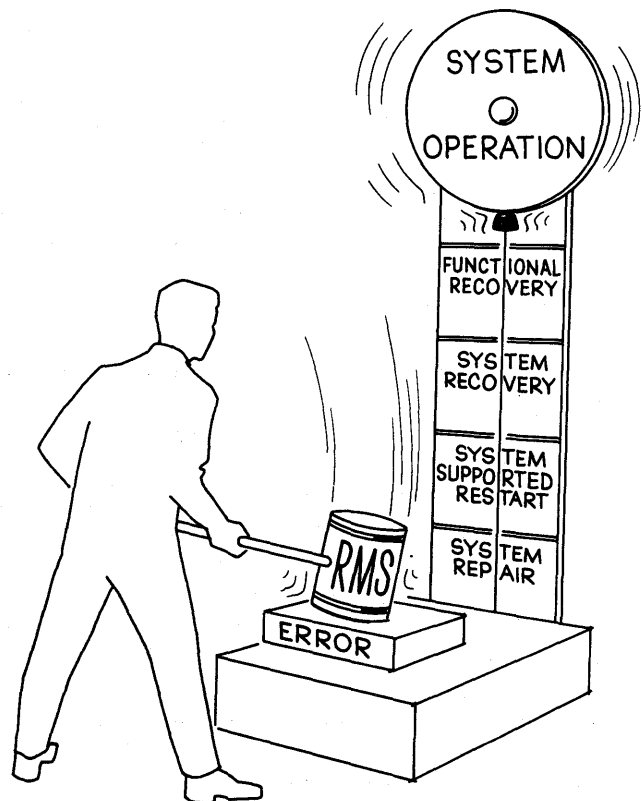


FIGURE 1

tion of recording information concerning what happened, the status of the computer at the time of the incident, what action was taken and the results of such an action. This information which is recorded on a special data set SYS1.LOGREC, is then available through execution of the Environment Record Editing and Printing utility (EREP) which runs under the control of the Operating System/360. This program edits and prints the records generated by MCH and CCH (as well as by several other recording functions) and provides the information for interpretation by the experienced Customer Engineer. A Standard Operating Procedure in a Computer Center using MCH and/or CCH should be to execute EREP on a regular basis and then the information should be available to the CE as an aid or indicator to anticipate serious trouble. For example, if a particular pattern appears indicating possible degrada-

tion, preventative maintenance can be performed before the occurrence of a serious incident.

CONCLUSION

RMS/65 is a step in the direction which error recovery must take if the requirements of computer technology are to be met in this area. More and more the question of error recovery cannot be relegated to hardware or programming alone but rather these two must form an effective partnership and attack the problem together in order to provide a satisfactory solution. Every sign indicates that this is being accomplished and it appears that some meaningful steps such as RMS/65 are being taken toward the goal of reducing the number of interruptions to which a user is exposed and to minimizing the impact of these interruptions when they do occur.

OPTS-600—On-line Peripheral Test System *

by G. W. NELSON

Graphic Corporation
Phoenix, Arizona

INTRODUCTION

Computer Test & Diagnostic (T&D) Programming Development and usage in the past ten years have undergone very few changes. As a result of this, the philosophy and attitude towards T&D's are passé. From basically simple systems, to the very complex, the T&D's for the early systems were completely off-line—either the customer work was run, or T&D, but not both.

Being a dynamic industry, hardware and customer software systems grew in complexity, sophistication, and customer dependence. The third generation systems brought with them multi-programming, compute and input/output overlap, multi-processors, and the heavy customer on-line work loads.

The failure of a peripheral device such as a magnetic tape handler, card reader, etc., does not always necessitate the loss of the total system, due to the ability of reallocation of a replacement device to the job. However, the system throughput is decreased by an amount proportional to the work capacity of the lost device. Should the failure be of such a nature, the customer is faced with two choices: (1) give the entire system to the maintenance personnel so that off-line T&D's can be run to determine the specific malfunction, or (2) keep on running at a reduced capacity. In only the most extreme circumstances will the customer choose the first alternative.

If the customer continues to run, then the failed device cannot be examined or repaired until the normal maintenance period. This turns out to be

a double edged sword which hurts the customer, due to reduced capacity and the maintenance staff, as they must wait in order to start taking corrective action.

The General Electric Company recognized that in many cases this awkward situation could be greatly reduced or possibly eliminated if on-line T&D's were available. Active development of on-line T&D's for the GE 625/635 computer systems started approximately four years ago. This has evolved from the early peripheral exercisers to the present comprehensive On-line Peripheral Test System—OPTS-600.

The OPTS-600 System is an integral part of the total General Electric Comprehensive Operating Supervisory (GECOS) System (see Figure 1). The test executive, OPTS-600, operates within the frame work of GECOS-III, but provides for all test dispatching, peripheral device allocation, lan-

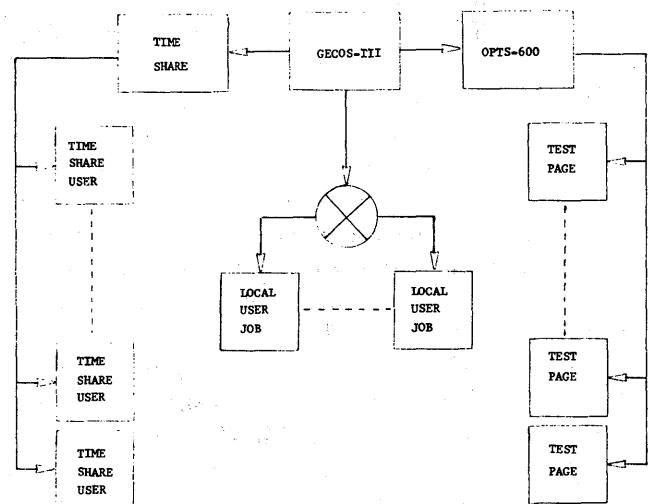


FIGURE 1

*The work performed herein was accomplished at the General Electric Co., Phoenix, Arizona.

**Formerly with General Electric Co., Phoenix, Arizona

guage processing, issuing of test input/output, data transfers, memory space management and error output. The test executive will control from one to eight individual peripheral tests in a multi-program environment, *concurrent* with normal customer operation.

The use of an on-line test system is relatively low when compared to the utilization of GECOS subroutines, libraries, compilers, and assemblies. For this reason the OPTS System is not maintained permanently in memory, but called in from system file storage on demand by customer or maintenance engineer.

Console request verbs describe the exact type of test to be executed on the desired peripheral. Requesting a test on a specific device results in a test page being called in for execution. Test pages represent tests explicitly designed for a particular peripheral device. The OPTS-600 system contains a complete library of comprehensive test pages which are available for selection by the maintenance engineer. The test pages are written in a test and diagnostic English type language, which is also available to the maintenance engineer via conversational console entries. In this manner the maintenance engineer can write and execute tests of his own design concurrent with customer operation.

OPTS-600—On-line peripheral test system

The purpose of the On-Line Peripheral Test System (OPTS-600) is to provide the on-line capability, under GECOS-III, of comprehensive testing and on-line trouble-shooting of malfunctioned equipment. This is accomplished without unduly interfering with the overall system capability to continue the processing of customer jobs.

A secondary function is that during slack periods of customer operation, or idle peripheral availability, equipment testing can be accomplished. This allows the normal schedule preventative maintenance time to be utilized more effectively in the actual corrective maintenance of equipment rather than the running of tests.

The OPTS-600 System is an integral part of GECOS-III, similar to the Time-Sharing System which is also a part of GECOS-III.

It is not necessary for all segments of the GECOS-III System to operate in the unrestricted memory access mode (master mode) at all times. Most functions can be executed in the restricted

memory access mode (slave mode) and would require only occasional excursions into Master Mode. Systems programs such as OPTS-600 are allowed this privilege of going into master mode when necessary. It is under this Privileged Slave Mode (going into master mode when the need arises) that the OPTS-600 system operates. By operating in the Privileged Slave Mode instead of directly in master mode, GECOS-III System's programs are afforded greater fail soft protection against malfunctions causing systems disasters. Should a malfunction occur in a systems program, only that program will be eliminated. This is particularly useful to the OPTS-600 System because of the probability of testing malfunctioned equipment. OPTS-600 is called into memory from system file storage via console demand.

The OPTS-600 Executive requires 5000 words of memory and can control up to eight different test pages at once in a multi-test program environment. The OPTS-600 Executive consists of four main modules. (See Figure 2).

1. Master Mode Service
2. Language Processor/Dispatcher
3. Error Output
4. Test Page

The Master Mode Service module is the only portion of the OPTS System that operates in the unrestricted memory access mode. Five main service functions are provided by this module:

1. Peripheral allocation and request buffering.
2. Test Page loading, initialization, and memory space management. Memory is allocated dynamically as required, so as to keep requirements at a minimum.
3. System Termination.
4. Fault protection and processing.

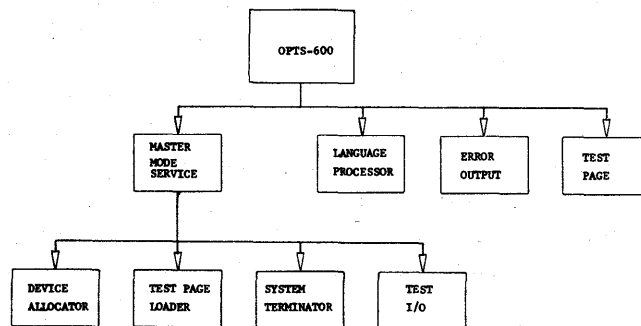


FIGURE 2

5. Issuing of Input/Output data transfers.

The Language Processor/Dispatcher is the main operating module within the OPTS system. The major portion of this module contains the Test and Diagnostic language processing function. Processing of the T&D Language entails the following functions:

1. Interpret source language code.
2. Set up Input/Output (I/O) test sequences as directed by the source language.
3. Perform error checking on behalf of previous test I/O.
4. Execute operator selected operating options.

The dispatcher controls from one to eight test pages. Input/Output is maximized in the dispatcher by dispatching control to the Test Page that has had its I/O completed for the longest period of time.

The error output module formats and transmits error messages for the test page to the console that originally initiated the request, the systems accounting file for historical error development, or both depending upon the selected operating options.

The OPTS System is capable of directing error output and control input requests for a test page to any one of four local consoles. This provides the maintenance engineer the ability to initiate a test request on a subordinate console, thus reducing the output and interference on the main system console which receives the bulk of the system message traffic (e.g., a maintenance console).

For the systems that have Real-time capability, OPTS-600 provides the ability to use a remote teletype (TTY) as if it were a local console. A small percentage of device malfunctions will require that the local maintenance engineer obtain

the assistance of centrally located device specialist, in order to correct the problem. By using the TTY, the specialist can dial into the computer system and be automatically connected to the OPTS system. He now has available to him the full range of operating features of OPTS-600 Test Pages and programs of his own design. The TTY now becomes in reality, a local console on a long extension cord. All error messages for the device under test will be directed to the local console, the remote TTY, with the additional ability to transmit copies of the error message to still other TTY's for monitoring. In getting first hand knowledge about the malfunction via OPTS-600, the specialist may be able to instruct the local engineer as to the type of corrective action to take. By resolving the problem in this manner, down time will be considerably reduced due to the device being out of service for a shorter period of time.

The OPTS system provides the maintenance engineer the capability of accumulating all of the OPTS-600 error messages on a Systems Accounting File which is dedicated file for all 600 Systems. In the case of routine device testing, the accumulate mode of error collection will bypass console error message typeouts. Then, at a more convenient time the maintenance engineer can dump all accumulated error messages to the printer. These messages are identical to what would have been typed out on the console, plus the engineer now has a permanent and detailed historical error report.

Test programs may be called in three ways. The operator may initiate a test program by typing the verb "TEST." The request verb is followed by the descriptors which exactly define which peripheral, the type of test desired, and the desired operating options. See Figure 3.

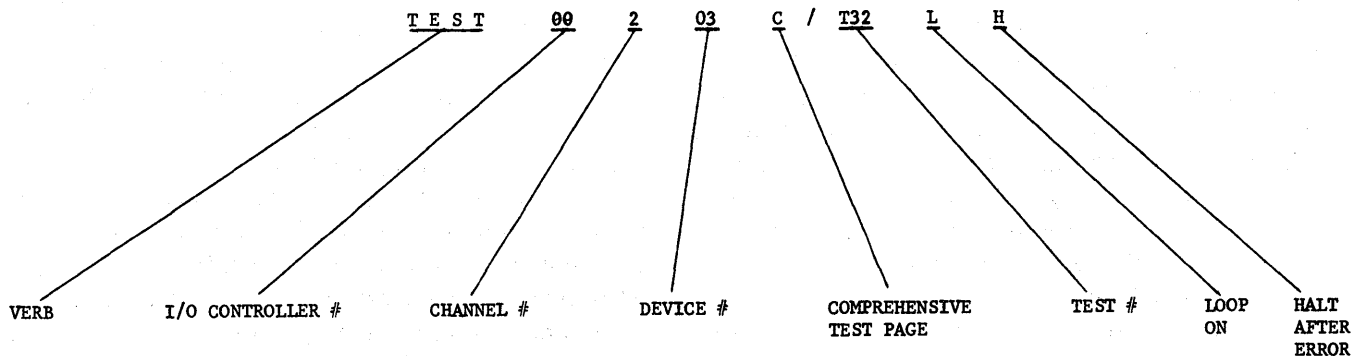


FIGURE 3

This would request a comprehensive test page (C), to be executed on Input/Output controller 00 (IOC), Channel 02 of the IOC, and Peripheral Device 03 on that channel. The operating options specify that execution is to start at test 32 (T32), loop on that test (L), and halt after each error (H).

It is recognized that a specific peripheral device may not be available for test at the time the maintenance engineer enters a request. So, to prevent him from having to stand guard at the console, and attempt to use the "TEST" request before the GECOS system allocates the device to another job, he may enter the "TSTQ" request. This request is identical to the "TEST" request except that it says the device is obviously not available right now, so queue the test request and periodically check the availability of the device. Once the device is found to be available, start a test as specified by the request parameters.

During error recovery by the GE 625/635 supervisory system on behalf of a customer job, the machine operator may be interrogated as to the desired recovery action to be taken. The operator has a varied combination of responses that he may evoke. The "T" option can be used at this point in conjunction with other responses. For example: he could reply "RT," this would cause the error recovery to be *retried* and the specific device *marked for testing* before it is allocated to another user. The "T" response is the machine operator's counterpart to the "TSTQ" request.

To minimize the number of operator messages and responses required, operating options can be used to control the testing of a device. Any combination of up to five option characters may be entered with the "TEST" or "TESTQ" requests. After a Test Page has gone into execution, any combination of twelve option characters may be used. Figure 4 is a list of the operating options that may be applied to a specific Test Page.

Peripheral Test Pages are written in a macro Test and Diagnostic Language (TDL). Each Test Page requires 2000 words of memory, and is divided into a series of tests. The tests are escalating in complexity, starting with very basic communications, and advancing to the more complex events of sequencing and data sensitivity. Each test performs a particular testing function and is capable of being looped on indefinitely, or of being executed without requiring the previous execution of any other test. When errors occur, part of the

A = Accumulate on Accounting file for Historical Error Development
 B = Bypass console error messages
 C = No transient error recoveries
 H = Halt on error for new operating options
 L = Loop on current test
 N = Negate next option character
 O = Go to enter options for more operating options
 R = Recycle entire Test Page
 S = Skip to next subtest
 Txx = Start execution of test xx
 (eg: ABL = Accumulate errors on accounting file, Bypass console error messages, Loop on error.)

FIGURE 4

error message contains a direct reference to the test that failed, the line of code that was in execution, and the exact peripheral instruction being attempted. This error reference is also the paragraph number in the documentation that contains a prose description of the purpose and method for the test and operation involved.

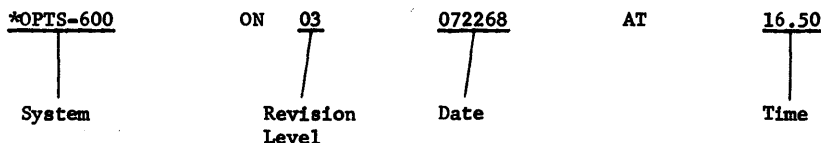
When the OPTS System is called into memory via an auxiliary console, the master console is notified of the request by a log-on message, giving the System Identification, Revision Level, Date and Time. See Figure 5.

Once the requested peripheral device is allocated and the test page called in, a start message is issued to the originating console. See Figure 5.

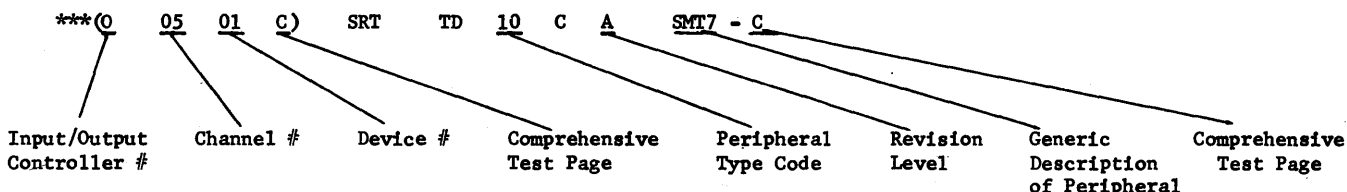
Within the parentheses is the identification of input/output controller, channel, device number, and the type of test page that will be executing. This information prefixes all messages related to a device under test. The message further states that it is a start of Test & Diagnostics for Device Type 10, Comprehensive Testing Program, Revision Level A. The device type is further explained as being Standard Magnetic Tape, Seven Channel, Comprehensive Test Page. Unless the operator has requested specific control over the test page, or the device becomes inoperable due to an attention condition, no further operator intervention is required. Upon completion of the test page, a termination message is typed.

In addition to the more detailed error messages which accompany each device error as they occur, a summary is typed with the termination message. This termination message will summarize all errors encountered during execution, plus the amount of channel time in milliseconds that the

LOG-ON MESSAGE:



START MESSAGE:



TERMINATION MESSAGE:

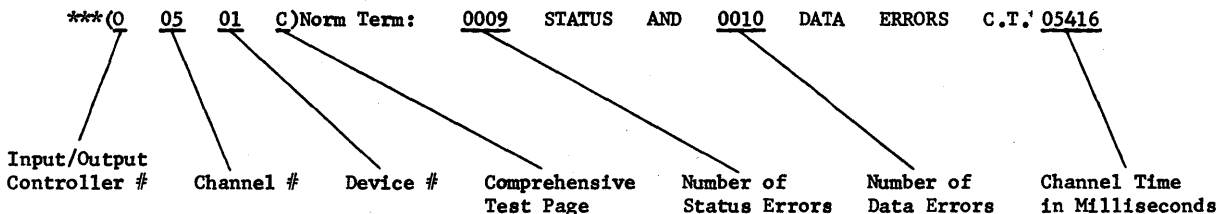


FIGURE 5

test ran. See Figure 5.

Upon the completion of all test pages, the OPTS System will type a log-off message to the master console. This message will be the same as the log-on format, with the exception that the amount of processor time that the entire OPTS-600 System was in execution is given in milliseconds. All memory will then be released back to the main level operating system (GECOS).

The OPTS-600 System also provides the unique ability to the maintenance engineer to manually write and execute special tests on-line, concurrent with customer operation. This is provided by the "Manual Test Pages" which are available for all peripherals. The "Manual Test Page" contains no coding: the maintenance engineer provides the coding from the console, or remote teletype. By specifying a manual test page he may then enter the Test and Diagnostic Language coding of his own design. This provides the needed flexibility for custom designed T&D to cover as needed situations, or more convenient on-line trouble-shooting. The maintenance engineer has the same instruction repertoire available as the diagnostic pro-

grammer. The OPTS Executive will monitor the instructions and inform him of any illegal instructions or sequences that could cause system problems.

All test pages are written in TDL (*Test & Diagnostic Language*) which is a Macro Language that causes a complete sequence of events to be executed for each Macro encountered.

The "TDL" Instruction Set consists of Macros for all normal test and diagnostic functions with provisions for adding Marcos to perform specialized functions when necessary.

There is basic information that must be used when issuing I/O commands to a peripheral subsystem. This is true regardless of what language the test is written in. The information normally required is given in Figure 6.

Obviously, if all this information has to be defined for each command issued, any test program, even a fairly simple one, would be somewhat complex to write and understand. "TDL" combines these items into two groups.

The first six items are combined into each peripheral operation instruction.

1. Peripheral Op-Code---Operation code directed to a peripheral such as read magnetic tape, punch card binary, etc.
2. IOC Command-----The command directed to the Input/Output controller, such as transfer data, skip over data. This is used in conjunction with the op-code.
3. Record Count-----The number of blocks of data to transfer.
4. Major Status Expected---The major classification of the state of peripheral. This is reflected back to the computer at the completion of each peripheral operation.
5. Substatus Expected----A further break down in the classification of a peripherals condition.
6. Interrupts Expected---A signal sent by the peripheral at the completion of an operation.
7. IOC Number-----There can be up to four Input/Output controllers on one system.
8. Channel Number-----The specific channel on an IOC, as there can be sixteen.
9. Device Number-----The specific device connected to a channel, this also may be one of sixteen.
10. Record Length-----The number of words to transfer
11. DCW List-----Data Control Word. This word specifies where in memory the data will be transferred to or from.
12. Type of Data-----Octal, decimal, random.

FIGURE 6

For example:

“WTB”—Sets up Peripheral Op-Code 15 (Write Tape Binary), IOC Command 00 (Data Transfer), Record Count of 1 (one block to be transferred), Major and Substatus Expected 00 (Ready), and Termination Interrupt Expected.

“BKF”—Sets up Peripheral Op-Code 47 (Backspace File) IOC Command 01 (Non-Data Transfer), Record Count of 1, (one block to be shipped), Major status 04 (End of file), Substatus of 17, and expects a termination interrupt.

Because they can apply to many peripheral operations within a program, the last six items in the list may be stored in “Standards” and applied to each subsequent peripheral operation. Standards may be overridden by defining any of these items immediately following the peripheral operation mnemonic.

“TDL” has instructions for defining different types of test data, e.g.:

D7070707070 .Data in octal

AD010101010101	Add to existing data pattern in standards
DRAN	Generate random data pattern
DROT	Rotate data one character position left
DLNXX	Use data from “TDL” Line XX
DREAD	Write data from read area

Instructions for defining any or all information necessary for I/O execution are provided as well as utility instructions for looping, branching, outputting special messages, e.g.:

LPO5.49	Loop to line 05 a total of 50 times before going to next instruction.
LP85.SV1	Loop to Line 85 and save return address in return register one.
RET1 + 1	Return via save return register one plus one field.

SUMMARY

Realizing that computer equipment does fail, and of all failures peripheral devices, such as card readers, card punches, and magnetic tape handlers, represent the largest proportional share of these failures, On-line Comprehensive Test and Diagnostic Systems can be very beneficial in improving total system availability, and customer satisfaction.

To this end, the OPTS-600 System is currently being used by the maintenance engineers on the GE 625/635 computer systems. Through the utilization of the on-line test and diagnostic system, the engineers preventative maintenance (P.M.) time is used to actually do P.M. or corrective maintenance (C.M.), rather than running tests in order to determine if additional maintenance is warranted.

An additional advantage is achieved by the T&D System, in that tests are executed in the same environment that the customer experiences. By being on-line and an integral part of the total operating system, the customer is able to establish a higher equipment confidence level due to the OPTS-600 System being subjected to the same loading and interference situations.

Fail-safe power and environmental facilities for a large computer installation

by R. C. CHEEK

Westinghouse Electric Corporation
Pittsburgh, Pennsylvania

INTRODUCTION

For a modern large-scale computer installation, the reliability of the power supply and environmental conditioning systems is as important a consideration as the reliability of the computing equipment itself. This is especially true of installations which involve on-line real-time and time-sharing operations, in which no convenient re-start point is available for resumption of operations after an outage. It is becoming more and more important in batch operations, many of which have grown in scale to the point where the expense of lost time due to re-starts can be a major economic factor.

Reliability of power supply, as the term will be used in this paper, refers not only to the continuity of power but also to its quality in terms of constancy of voltage and frequency and its freedom from momentary transients and surges. From the standpoint of general continuity, the typical electric utility power supply is excellent. However, this continuity is achieved by the provision on the power system of redundant lines and circuits equipped with fault-detecting relays, so that a line on which a fault occurs can be automatically and quickly removed from the system. Inevitably, the fault itself creates a transient voltage surge which is propagated generally over the system, and the switching operations required to isolate the faulty line create additional transients. These transient voltages often find their way into users' circuits. They go relatively unnoticed by the average user, but they may induce a delicate flip-flop circuit in a computer to flip when it should flop and play havoc with a program in the process of being executed. As will be pointed out later in this paper, such transients may be transferred by induction between circuits which upon casual study appear to be completely isolated electrically as far as metallic connections are concerned. For example, they may appear in the output of a motor-generator set by induction due to the proximity of the generator output wiring to the motor input wiring.

The continuity of service required of environmental conditioning equipment is only slightly less critical than that required of electrical power. A failure or outage of a few seconds to a few minutes can usually be tolerated before performance of the computer system begins to be affected. This means that suitable temperature and humidity detectors can be used to sense trouble conditions and sound alarms in time to permit stand-by equipment to be manually switched into service and avert a computer shutdown. However, it is obviously better to plan the system in such a way that manual intervention is not necessary in case of failure of a particular component.

The provision of fail-safe electric power and reliable environmental conditioning for the computing facilities of the Westinghouse Tele-Computer Center was a matter of serious concern from the outset of the planning for the Center in 1961. The measures eventually taken to achieve these goals considerably exceed those initially thought to be adequate; so it is obvious that a few lessons were learned along the way. The purpose of this paper is to describe the evolution of these facilities; to recount some of the problems encountered and their solutions; and generally to share with others the benefits of some experience in this relatively neglected area of planning for computer systems which must provide a high degree of operating reliability.

It should not be inferred that the solutions described in this paper are the only suitable ones. Recent developments have placed a variety of approaches to these problems at the disposal of the planner of a new facility. In the evolution of the systems at the Center, the designs were based on the best available balance between economy and the degree of reliability desired, using the approaches available at the time.

*Power supply facilities at the Westinghouse
Tele-Computer Center*

The Westinghouse Tele-Computer Center is the cor-

porate-level computing and data communications facility of the Westinghouse Electric Corporation. Operations began late in 1962 with a single UNIVAC 490 Real-Time System, initially performing teletype message switching on the Corporation's private teletype network, along with a variety of straightforward batch processing applications. The Center is located at a site in suburban Pittsburgh, where the only available utility power supply was a conventional 4160-volt radial distribution circuit, serving many small commercial establishments and residences in the vicinity. It was obvious at the outset that this circuit was inadequate to serve the needs of the Center, not only from the standpoint of reliability but even its ability to provide enough power under normal conditions to supply the projected load. It was therefore necessary to have power at 23-kv brought to the site by the utility from a 23-kv sub-transmission circuit approximately a mile away which interconnected two of the utility's 23-kv substations. This would have provided adequate capacity, but considerations of reliability led to the decision to have a second 23-kv feeder brought into the site from a geographically separate 23-kv interconnection approximately three miles away. The two 23-kv feeders were built on separate pole-lines on each side of the road along which they run together for several hundred yards before entering the property, and they share common poles only after entering the site. It was also decided to install a private substation on the site, in order to incorporate in it special relaying and switching equipment for transfer of the load from one line to another in case of failure.

The substation (Figure 1) incorporated two identical 1500 kva transformers, 23 kv to 4160 volts regulated, either of which alone could supply adequate power for the whole building to the 4160-volt bus in the outdoor substation. The 750-kva power center inside the building, supplied from this bus, stepped the 4160 volt power down to 120/208 volts for distribution within the building. The power distribution unit for the UNIVAC 490 computer system was supplied by one feeder from this power center. It supplied power directly to blowers, motors, and other "non-critical" units in the computer system, as well as to a motor-generator set which in turn provided regulated voltage, presumably free of surges and transients due to external system disturbances, to the central processor, communications control units, and other units which were deemed to be critical in sensitivity to power fluctuations.

Both the "raw" power and the "regulated" power were distributed throughout the computer room in enclosed common wireways, separate from signal cabling which was laid in open cable trays.

The original scheme of operation was to supply the

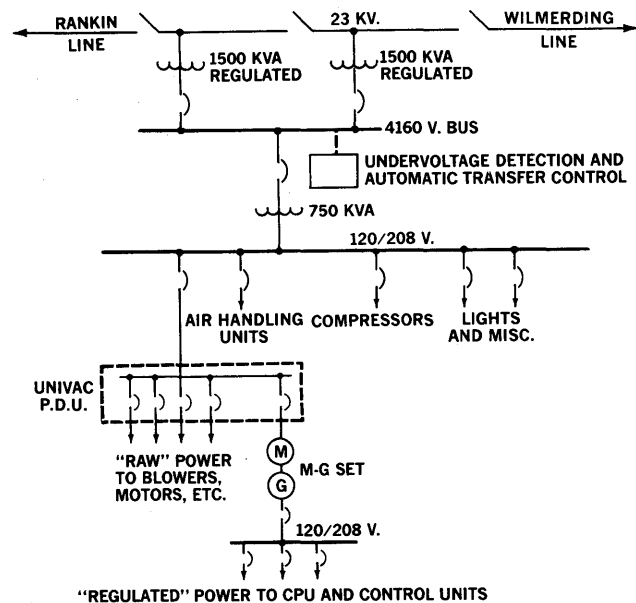


Figure 1—Original configuration of power-supply system

total power requirement of the site from one of the 23-kv lines with its associated transformer. The other transformer was kept energized by its 23-kv circuit, but its 4160-volt breaker to the bus was kept open. An undervoltage relay on the 4160-volt bus detected loss of voltage on the bus due to failure of voltage on the 23-kv circuit normally in use and transferred the load to the other transformer and line by sequentially opening the breaker from the faulty source and closing the breaker to the other. Relaying and circuit breaker operating times were adjustable to considerably less than a second, and it was felt that the inertia of the m-g set supplying the critical computer units would be sufficient to maintain adequate speed to keep these units supplied with power during this period. Also, it was felt that any fluctuations in frequency at the output of the generator would be gradual and therefore not disturb the operation of the system it supplied.

These assumptions were probably correct, and the theory of operation described was probably very good, as far as it went. However, during repeated tests of the transfer scheme during the remainder of the winter of early 1963, and upon the one or two occasions when power actually did fail and initiate a transfer, the transfer was never accomplished without resulting damage to the programs running in the UNIVAC 490 and an abrupt shutdown of the entire computer system. During this period more and more units of the computer system were moved to the isolated supply, on the theory that perhaps some of them which previously were deemed noncritical were actually injecting false inter-

rupts or other spurious signals into the central processor and affecting its operation. The transfer time was shortened to the minimum possible, less than half a second. However, these efforts were to no avail, and although the possibilities of this line of approach were exhausted, the transfer scheme was still unsuccessful.

But the worst was yet to come. With the advent of spring came the thunderstorm season, and it was quickly learned that when the skies began to darken with thunderclouds it was good tactics to get the recovery-program tapes out of the racks and ready to mount, because an outage was probably imminent. It was also found that even an instantaneous transfer scheme would not have solved the problems, because surges due to lightning strokes and resulting switching operations on remote parts of the utility's power system of duration sufficient only to cause a momentary light flicker at the site, were enough to shut down the computer system.

It was then concluded that it was these transients in the "raw" power source, induced into the wiring of the presumably isolated power source due to their proximity in the common wireways, which were causing the trouble. Furthermore, it was apparent that no scheme for transferring to an alternate source of power would work under these conditions, because the switching transients set up by the transfer itself would defeat its major purpose.

A solution might have been to rewire the entire computer system, isolating the wiring for critical units in separate wireways and keeping it separate throughout all parts of the system. However, this would not only have been quite expensive but would have required extensive shutdowns of the system, which was not feasible at the time. It was decided to take a "brute-force" approach and supply the entire computer system with isolated power from a single large motor-generator set, with the output power leads from the generator well separated from the raw power source to the motor.

The motor-generator system selected is called the "constant-frequency" or "CF" system, in which a conventional m-g set is mechanically coupled to a higher-speed flywheel through a controllable eddy current coupling or electrical clutch (Figure 2). Under normal conditions the motor receives power from the input line and drives the generator in the conventional way. The flywheel, disconnected from the system by the electrical clutch, is driven by a separate motor at a higher speed. Upon failure of normal power, which is sensed by voltage and frequency sensors on the input line, the clutch is energized under control of a frequency regulator at the output of the generator. The coupling of the flywheel to the m-g set is controlled in such a way that the flywheel gives up its energy to the system at a

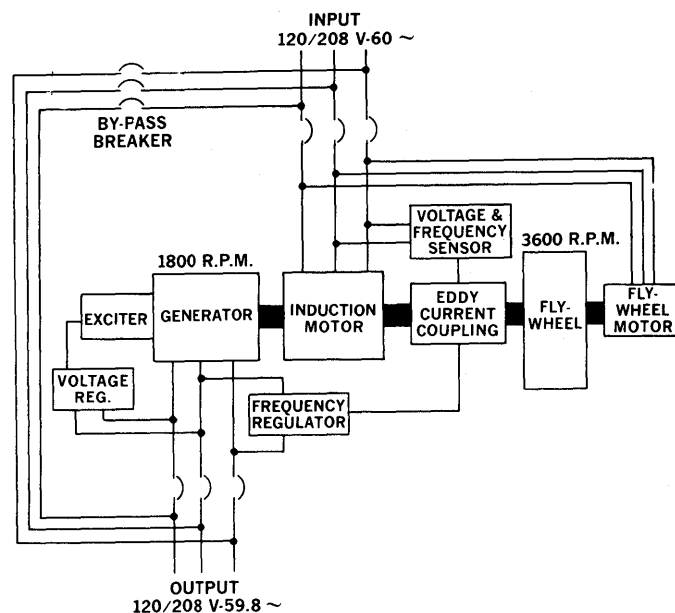


Figure 2—"Constant-frequency" motor-generator system

rate which maintains the speed and therefore the frequency and voltage of the generator until the flywheel slows down to the speed of the m-g set.

Such a system, rated at 80 kilowatts, with a flywheel sufficient to supply full output for a minimum of 12 seconds after loss of input power, was ordered for installation before the lightning season of 1964. Upon installation of this system, it was immediately possible to realize the planned benefits of the alternate power transfer scheme. Several failures of the normal power line subsequently occurred. Except for a momentary outage of the computer room lights (which were on a separate circuit from the computer system), computer operating personnel would have been unaware of the fact that transfer to an alternate power source had taken place, because the computer system was unaffected. Furthermore, during the lightning season of 1964 alone, this system averted many outages and the loss of information and time that would have resulted from the many power-line disturbances that were noted.

Although several failures of the normally-used power line have occurred each year, they have been of very short duration. And in every case but one, the alternate source was available, and the transfer was made automatically and successfully without interruption to the real-time operations. Fortunately the one case referred to, in which the alternate source failed simultaneously with the normal source, occurred during a weekend when real-time operations were already suspended.

The computing facilities of the Center have expanded tremendously from the single UNIVAC 490 system with which operations were begun. These facilities (Figure 3) now include dual UNIVAC 494 central processors in addition to the original 490, which is still in service; a dual CDC 6600 system with a 6416 scheduler and control processor; and an IBM 360/50-75 system in an ASP configuration. This concentration of processing power correctly suggests that the volume and variety of the applications handled by the Center have expanded greatly also, and it is no exaggeration to say that a prolonged loss of power could have very serious consequences to the operation of the entire Westinghouse Electric Corporation.

The construction of a second building at the Center during 1966 and 1967, larger than the first, required re-planning of the power supply facilities. The lessons of the widely publicized Northeast power blackout led to a reappraisal of the initial policy of complete dependence upon the public utility supply for power. The decision was made to install a quick-starting diesel-engine-driven generator as an alternate supply. With each of the three major systems supplied by its own constant-frequency m-g set, capable of providing full power for 12 seconds after failure of the prime source, it is possible to sense the failure, start the engine and bring it up to full speed in 10 seconds, adequate time to pick up the load of all three systems without interruption. The engine-driven generator is rated at 800 kilowatts, sufficient to supply all the computer systems as well as the critical auxiliary items needed to maintain computer operations. These critical auxiliaries are listed in Table I. The control system is arranged to drop automatically from the power centers all loads except these critical items whenever the engine-generator must be brought into service.

TABLE I—Westinghouse Tele-Computer Center Critical computer and auxiliary load

	<u>KVA</u>
Computer Systems	
UNIVAC 494 Systems (CF Set)	125
IBM ASP System (CF Set)	260
CDC 6600 System (CF Set)	260
Air Conditioning Compressors	
IBM and CDC Chiller Compressor	160
UNIVAC Air Conditioning Compressors	40
Fans and Pumps	
IBM Room Fans	30
CDC Room Fan and Pump	30
UNIVAC 494 Room Fans	15
Cooling Tower Auxiliaries	
Pump	15
Diesel Heat Exchanger Pump	10
Chiller System	
Electric Heater for Humidity Control	30
Lighting—All Critical Areas	20
Total	995

An important factor to consider in planning a system using an engine-generator is the ability of the generator to supply the starting KVA requirement of any large motors which may be dropped during the outage. In the case of the CF units, because the motor-generator combination is kept at rated speed by the flywheel until the engine-generator is ready to provide power, there is a momentary transient lasting only a few cycles, while the motor adjusts itself to the supply frequency. The control system allows a short time delay for this to occur before re-connecting the chillers, air-handling units, and pumps to the 208-volt bus. The KVA capacity of the generator is adequate to start the chiller units, which are by far the largest motors on the system and the only ones which might present any problem.

In this system (Figure 4), the bus-tie breaker is normally open. Undervoltage relays detect loss of power on the portion of the 4160-volt bus normally used to supply the computer systems. If the other portion of the bus is still energized, the transformer breaker is opened, the bus-tie breaker is closed, and the other 23-kv source supplies power to the entire bus. Meanwhile, the same signal is used to energize the cranking system for the engine-generator. If power is not available from the other 23-kv source, the tie breaker remains open and the generator is connected to the 4160-volt bus as soon as it achieves rated speed and voltage, provided normal power has not returned in the interim. Once the engine-generator is connected to the bus, the system does not reverse itself automatically even though the normal supply is re-energized. The transfer back to normal power

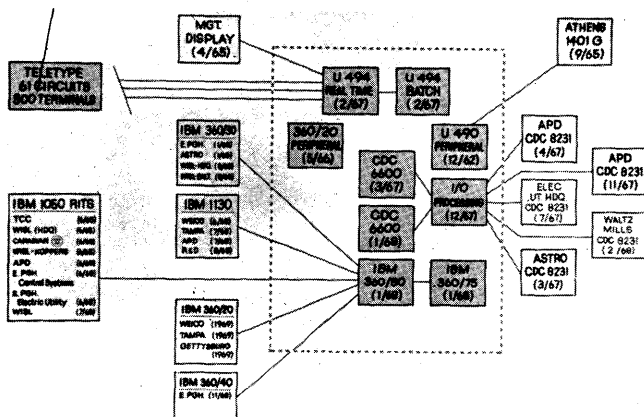


Figure 3—Computer configurations at the Westinghouse Tele-Computer Center

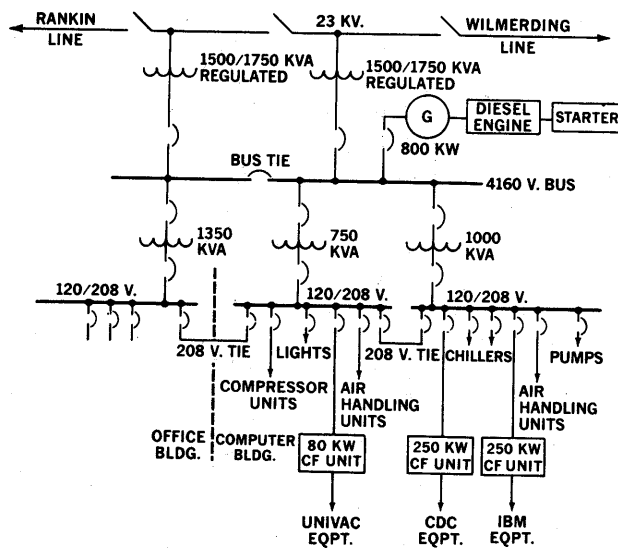


Figure 4—Present configuration of power-supply system

supply is performed manually. In this case again, the CF units provide a 12-second period during which the generator may be removed from the bus and the transformer breaker energized to restore normal supply from the main power transformer.

Static inverter power supplies

An alternative to the use of motor-generator sets for reliable power supply has been developed since the system described for the Center was designed. Rapidly increasing power capability and decreasing costs of power semiconductors have now made static inverter systems feasible as power supplies for large computer installations.

The basic elements of a typical inverter system are shown in Figure 5. The system consists essentially of a battery charger (static rectifier), a battery, and a static inverter. The rectifier converts input a-c power to d-c,

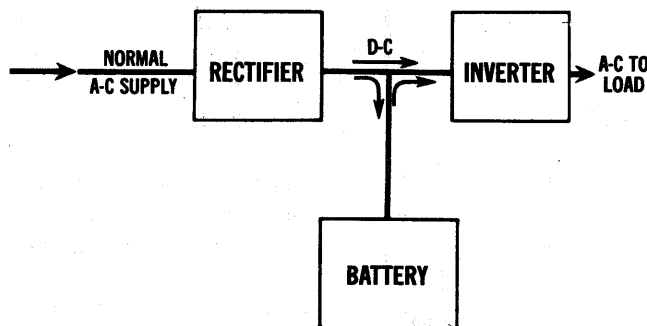


Figure 5—Essential elements of rectifier/battery/inverter power-supply system

maintaining the charge on the battery as it supplies d-c to the inverter, which converts the d-c back to a-c to supply the load. In case of failure of the input power, the battery supplies d-c to the inverter, maintaining the power to the load without interruption for a period depending upon the battery rating (several minutes in a typical installation).

In addition to providing a temporary source of power until alternative supply sources can be brought into service, the battery performs another important function. It acts at all times as a filter of almost infinite capacity to prevent power-line transients and surges from being transmitted through the system to the load.

Being completely static, the rectifier/battery/inverter system requires little maintenance and is clean and free of noise.

Such systems are now available with ratings from 1 kva to 400 kva, and several systems can be paralleled if necessary to achieve still higher total power capability.

Although the initial cost of a typical complete system is generally somewhat higher than that of a motor-generator set, the lower installation and maintenance costs and the operating advantages make the system an extremely attractive alternative.

Air-conditioning systems

In the planning of the air-conditioning system for the original 490 computer installation, it was concluded that a total of 20 tons of compressor capacity would be adequate to supply the units of the system which used room air for cooling. Certain critical units, including the central processor, were cooled by a separate ducted system, for which 20 tons of cooling also was sufficient. Each system had adequate cooling capacity for future expansion. Here again a stand-by system was designed, and the system finally selected used two 10-ton reciprocating units each for the room and the closed supply. For each system, an additional 10-ton unit was provided as a stand-by for each 20-ton combination. Air-to-air systems were used, partially to avoid the dependence upon continuous water supply that would have been inherent in a condensing-water system.

The controls were designed so that on each system, one 10-ton unit supplied the base cooling requirement, the additional compressor being cycled on and off under thermostatic control to supply the incremental cooling requirement. The system provided for manually-controlled substitution of the stand-by compressor for either of the normally-used compressors. Each of three compressors supplied its own separate set of cooling coils in the air ducts to the room system and the closed system respectively.

The initial control design provided for damping or modulation of the air flow by means of thermostatically

controlled vanes in the air ducts, so that control of temperature could be maintained within one degree despite the "on-off" cycling of the second compressor in each system. Since there was more than adequate cooling capacity provided when both compressors were on, the vanes cut down on air flow under this condition to maintain steady supply-air temperatures in the room and in the equipment.

The most important lesson learned from experience with this scheme was that a simple, reliable control system with a minimum of complication is far preferable to a complex, sophisticated one which theoretically provides more precise regulation. One of the requirements of a reciprocating compressor is that once it is started it should be run for some minimum period of time in order to insure proper lubrication (in this case a minimum time of four minutes). The control system was arranged so that once a compressor was started it automatically ran for this minimum time, even though there was no longer a cooling requirement for it. The result was that the air-flow modulation system occasionally closed the vanes in the ducts so far that ice formed on the cooling coils due to the restricted air flow. Once this began, the effect was cumulative, and it was not long before the cooling coils completely restricted air flow in the system. The result was a complete loss of air flow until all compressors had been shut down long enough to permit the ice to melt away.

The high-precision temperature control system was quickly abandoned, and the vanes in the air ducts were permanently propped wide open. The temperature was allowed to decrease as it would during the minimum cycle time of the cycling compressor on each system. Although this sometimes meant a variation of temperature in the ducts of as much as three or four degrees below the control temperature, this cured the icing problem and provided much more reliable operation.

Another restriction involved in the use of an air-to-air system is that it is harmful to the compressor units to operate such a system when outside air temperature drops below a certain minimum figure, in this case 15 degrees Fahrenheit. The control system was arranged to shut down the compressors and use outside air directly for cooling when this outside temperature was reached. This involved another complicated system of controls and vanes to mix outside air with recirculated air to maintain proper temperatures, one with which adequately reliable results were never obtained.

In general, experience with the air-to-air system and reciprocating compressors was not good, and in the planning for expansion a chilled-water system was designed, using centrifugal compressors (chillers), which are theoretically much more reliable. This has proven to be the case. Before adopting this system, however, arrangements were made for water supply from alternate directions to the water input line to the site,

with isolating valves on each side of the tap, so that a water-main break in either direction could be quickly isolated.

The system as it is now operating uses two 120-ton centrifugal chillers, providing chilled water to the water-to-air heat exchangers in the CDC-6600 and IBM computer rooms. The chillers in turn are supplied with condensing water circulated through outdoor cooling towers. Although the chillers are physically essentially in parallel, the control system operates the chillers in sequence. One chiller is sufficient to supply the entire cooling requirement of both computer rooms. The other is normally de-energized, and its condensing-water and chilled-water pumps are stopped. Thus, the first chiller takes the entire load. If, however, it should fail, or if any of its auxiliaries should fail, the resulting rise in chilled-water output temperature will cause the second chiller and its auxiliaries to be energized automatically, restoring normal cooling capacity without manual intervention.

Experience with this system has been very satisfactory. The centrifugal chillers are very quiet, reliable, and vibration-free. The water-to-water system presents no seasonal problems as does the air-to-air system. Because chilled-water supply temperature to the air-handling units in the computer rooms is a minimum of 45 degrees, there are no cooling-coil icing problems. Also, as described, the system provides its own back-up protection without manual switching.

SUMMARY AND CONCLUSIONS

1. For important computer installations, it is desirable to provide a back-up source of power which can be brought into service without interruption of continuous power to the computer system.
2. Unavoidable transients and surges on a utility power system can cause malfunction of a computer, and the power supply system should be designed so that these transients do not appear in the supply lines to the computer system.
3. Motor-generator sets with controlled-energy flywheels and rectifier/battery/inverter systems constitute two alternative means of isolating a computer system from transient voltages and maintaining continuous power while an alternative source is brought into service.
4. Simple environmental control systems with less precise regulation are preferable from the standpoint of reliability to more complicated systems with greater precision.
5. Operating experience at the Westinghouse Tele-Computer Center has proved chilled-water cooling systems with centrifugal compressors to be more reliable than air-to-air systems with reciprocating compressors, despite the requirement for essentially continuous water supply.

A generalized methodology for computer simulation of numerically controlled production systems

by GASTONE CHINGARI

Sperry Rand Corporation
Philadelphia, Pennsylvania

INTRODUCTION

Numerical control (N/C) is generally acclaimed as the largest single advance made in the techniques of industrial production in the last decade. It represents a key technical innovation that has significant effects on productivity, engineering design, product marketing, factory organization, employment and industrial relations.

The basis of these new types of production systems is the numerically controlled machine tool. In most types of metal-working operations the cutting or forming tool is told exactly what to do by means of a pre-recorded program. The direction in which the tool moves, its speeds and feeds, type of machining operation, together with auxiliary machine tool commands, are directed by digital instructions read from a punched tape or a magnetic storage. The N/C concept, as it applies to machine tools, emphasizes the significance of this new form of automation: the merging of information handling systems with production facilities.

Numerically controlled machine tools can be looked at primarily as information utilization devices which operate at the periphery of a digital computer. Their function is the utilization of information to give output in the form of finished physical parts. Computing power is necessary to prepare and organize the machining instructions which, when fed to the machine tool, represent a "fixed logic" program describing product dimensions, machine member direction, speeds, feeds and other process control data.

At present, a variety of numerically controlled machines are encountered in various industries, all of them requiring data processing services of various complexity and frequency. In the metal-working machine tool area, N/C machines such as turret drills, jig borers, lathes, multi-axis milling machines, engine lathes and multi-purpose machining centers are most common in large manufacturing companies but are also found in medium and small machine shops. Other numerically controlled devices that are being used extensively are drafting

machines, flame cutting machines, riveters, tube benders, welding machines, and inspection machines. Figure 1 shows a general-purpose N/C machine tool considered in the simulation model discussed in this paper.

The concept of computer-based numerical control is not relegated to the machine tool itself but can embrace the full spectrum of N/C production in an engineering and manufacturing company. Indeed, the entire process may go from functional specifications of the piece-part through production control, individual part manufacture, assembly testing and follow-on statistical performance evaluation. In addition, there is a definite trend for computer support in N/C manufacturing for part selection, machine center loading and scheduling in order to improve N/C/machine tool utilization. Information processing for numerical control is carried out in all three of the major phases required in the production of a part. Phase 1, information generation, constitutes the process of defining

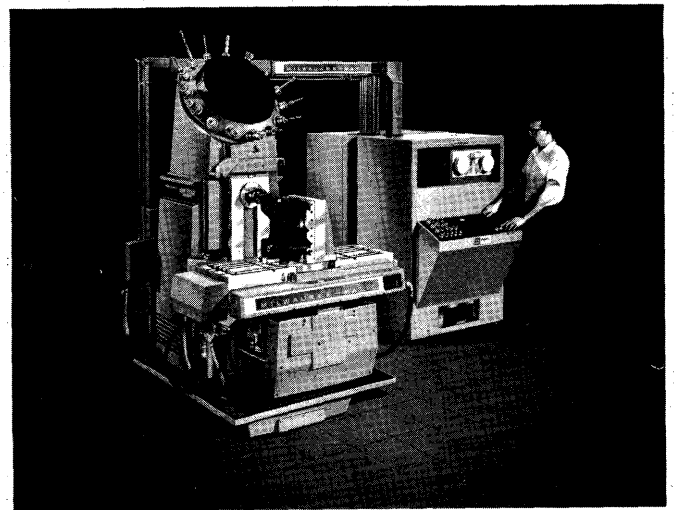


Figure 1—Numerically controlled machining center

a part. This process aims at arriving at some specific and detailed specification of the configuration of the part to be produced and at its method of machining. Phase 2, machine tool program generation, has the objective of providing the computations required for the production and verification of the complete set of machine tool instructions. Phase 3, information utilization, covers the final phase where the instructions in the machining program are used at the machine tool site for actual production of the desired part as visualized in the part design. The main objective in programming an N/C machine tool is the solution of the overall machining problem, i.e., the optimum application of an automated machine tool to a given piece of work according to pre-defined geometry and the attainment of the highest speed in the machine operation compatible with machine dynamics and efficient metal-working techniques.

Computer-based programming systems for numerical control remove the part programmer from direct contact with detailed machine tool coding and allow him to define the machining problem in broader and more meaningful terms. He can describe his problem and procedures with a symbolic or graphical language which is problem oriented and has little relationship to the computer itself. The data coming out of the computer are in the language that the machine tool director can understand. A symbolic-type manufacturing language (APT) with English-like terms can be used by production people to "talk" to the computer. In this language, which should be easy to master, the part programmer may describe the profile, surfaces or hole patterns which define the part to be machined, and specify the sequence of required tool motions to generate the desired geometry in metal forgings or castings.

As shown in Figure 2, the present state of the art in the use of general-purpose computers in machine tool programming reflects the recent advances reached in computer technology in general. Experimental graphi-

cal part-programming systems are in existence today that permit, through the use of a cathode-ray tube display equipped with a light-pen device, the definition of the part, and subsequently, of the machining problem. The part programmer can immediately see the results of the process as the solution of the problem progresses.

These programs can also produce true perspective pictures of any curvilinear segment of series of lines including coordinate axes, and provide the part programmer with an added degree of flexibility in controlling what he sees on the scope. By manipulation of console switches, the operator may rotate the three-dimensional view of the part being shown about many axes and thus observe it from various aspects as an aid in deciding if it has correctly defined the part configuration. The graphic communication capability is expected to increase part programmer efficiency and decrease the preparation time of verified N/C tapes.

Formulation of the generalized model

The formulation of a generalized model for N/C production stipulates parameters that provide the prime operating modes of the production system, and describes means for evaluating alternative design solutions.

A simulation methodology in numerical control requires the definition of a framework achieving integration of physical processes and information handling processes. Figure 3 shows a simplified representation of such process integration. There are essentially four component systems or subsystems considered in the modelling philosophy presented here. There are the N/C Data Processing System, the Physical Processing System, the Emergency Repair Support System and the Production Information System.

The Numerical Control Data Processing System has the function of generating machine tool control informa-

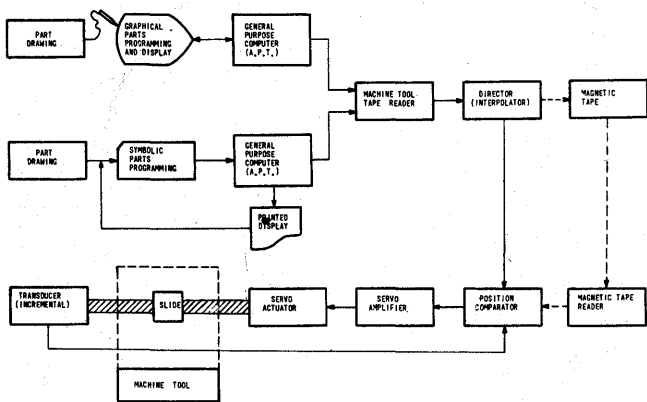


Figure 2—Information flow in a computer-controlled machine tool

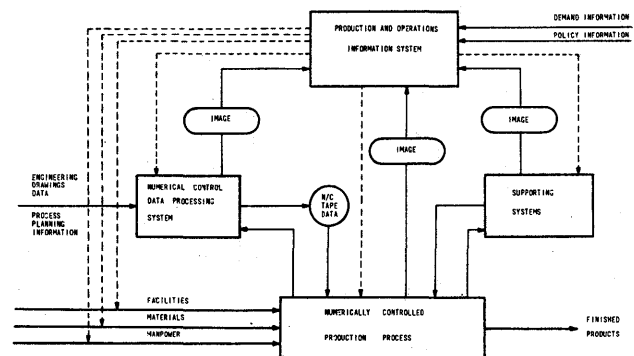


Figure 3—General model for N/C production systems—Basic schematic diagram

tion from engineering drawings and process planning data.

The Physical Processing System covers the production processes carried out by the N/C machine tools and the Emergency Repair Support System provides the services needed for repair of machine tool breakdowns.

The Production Information System encompasses all the activities of the three principal systems mentioned above. Sensing elements report to this information system events that take place in the subsystem. The end result of the information arriving at the Production Information System is a series of images that describe the activities within the subsystems.

The generalized model treats the numerical control plant as a man-machine system of discrete-type production and it makes use of feedback links within each subsystem and between the subsystems themselves. The production system may be composed of procedures, equipment, information, methods to compile and evaluate the information, as well as the people who operate and use the information. The basic schematic model of the generalized N/C production system is shown in Figure 4. The model is a detailed representation of the

operations and flows illustrated in elemental form in Figure 3.

The flows as indicated in the diagram are of four distinct types. The solid-black line network shows the materials flow from raw material storage to finished part storage, going through the numerical control machining operation. The double-line network provides the carrier for all information issued from the arrival point of the engineering drawings to the issuance of numerical control tapes for the production process, going through various points for processing of the numerical control programs. The broken lines identify information flow for machine repairs. The thin-line network links all the other three networks and their work stations to an information center designed for the measurement of system performance during the simulation.

Shop orders are treated as purely exogenous inputs, accompanied by engineering drawing releases and process planning data. These orders represent the transactions which go through the system in a stochastic flow. Based upon decisions on the extent to which plant capacity is used, the model allows incoming orders to

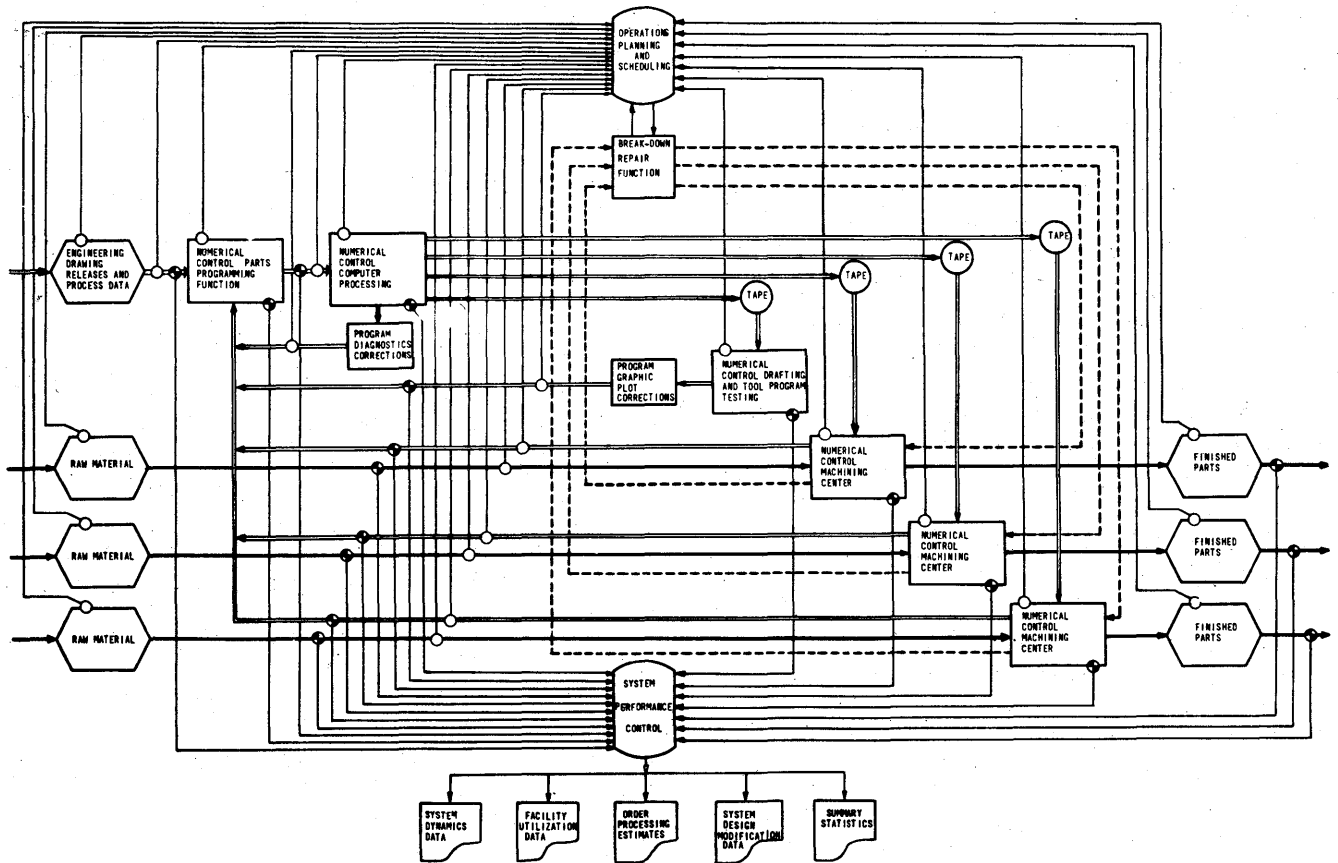


Figure 4—General model for N/C production systems—Complete model diagram

enter the system or to be routed to "sinks" which represent subcontracting functions. Orders entering the system are routed to the numerical control part programming function which handles all the operations necessary to code, process, validate and machine-tool test the information necessary to run the N/C machines. The computer processing operation simulated in the model produces families of N/C tapes to be used in the major routings considered in the model. The machining programs are released to the machine centers for tape proofing prior to the final release to production.

The highly aggregated function blocks shown in the general schematic, as well as the various linkages of direct flow and feedback loops, may be broken down to a much greater level of detail through the use of the General Purpose Systems Simulator (GPSS) language. GPSS is the computer simulation language which was used in the experimentation connected with the development of this methodology. Flows through the numerical control parts programming function, for example, can be described with a finer resolution when put through the sub-functions of Part Programming proper, Computer N/C Processing, Graphic Verification and Control Tape Proofing. Within the Breakdown Repair function boundary logical blocks are introduced representing sources of machine breakdown events, together with blocks representing repair functions and termination of repair operations.

The Numerical Control Machining Centers may have one or more machine tools of the same type operating in the center. Parameters affecting the machining center operations are the number of working shifts per day, the length of the individual shift and the number of working days per manufacturing week. Wide flexibility exists in programming shifts for the various centers, including the assignment of different shifts of varied duration.

The scheduling program requires a description of available facilities per center in terms of a facility identification, number of shifts of operation, and number of machine hours of capacity. The scheduling function operates on the given job information to calculate for each job the probable number of manufacturing days needed to process the job and the manufacturing week number in which the job processing will start at the machine center. It determines the load that would exist at each machining center according to the over-all job lot processing time and to the frequency of job order and tape arrivals.

Design criteria and structure of the N/C production systems simulator

A methodology of simulation of N/C production systems requires a formulation of the various characteristics of total system components, subsystem interde-

pendencies, system flows, and stochastic functions. The resulting structure may be called an N/C Production System Simulator, which is a computer program capable of tracing the activities of the N/C production system as they change in time.

The composition of the N/C Data Processing System and of the Physical Processing System can be described by program modules representing part programming functions, computer processing functions and machining functions. The simulation of the flow of the job orders and part programs through the production system can be achieved by transmitting "transactions," representing job orders and part programs through the network of simulated queues and processing stations. The arrangement of the queues and the processing station elements also includes information feedback loops which are required by the nature of numerically controlled operations.

Production system input characteristics

The first task in the definition of the input characteristics is to identify the various types of transactions which flow in the various information streams of the GPSS model and in its control strings. The latter are GPSS model segments operating outside of the main model, which control events taking place in the simulator, such as machine breakdowns, or control duration and number of work shifts at the machining centers of N/C data processing units.

Production orders entering a shop generally follow two main streams—new orders and reorders—flowing as transactions into the network of the model. The new orders require production planning, the definition on work routings, setups, part programming and preparation of N/C tapes. The reorders have the necessary N/C tapes already available from previous job-lot runs. These tapes are usually stored in the tool room and are treated as any other tool which may be required for setup and fabrication of the specific job lot.

Each transaction entering the model represents a production order for new parts or for additional production of parts identical to ones produced in previous orders. The transactions are assigned to certain parameters defining the production order configuration and the characteristics of the processing centers through which the order is expected to go, either for N/C information processing and/or parts fabrication. Due to the different nature of the operations carried out in the N/C Data Processing System and in the Physical Processing System, the same transaction may have entirely different parameters assigned when it goes through the two subsystems. The GPSS parameter assignment is made according to the system in which it operates or to the seg-

ment of the specific system in which the transaction is active at the moment.

An analysis of the functions and elements of the various subsystems should be preceded by a review of the assumptions made in relation to the generalized model structure, the subsystem interfaces and total system inputs. Technological constraints pertaining to the metal-working processing should also be considered.

The model represents a data processing operation used to produce numerical control tapes, which is integrated with a numerically controlled machining complex that manufactures parts. It does not recognize adjacent departments such as shipping or purchasing. Arrivals from outside the system may come from departments such as purchasing or engineering, or they may be direct customer order arrivals, as in the case of small job shops. A production order can coincide with a shop order in the context of this simulation. In practice shop orders are released 10–15 days before the first operation is scheduled to begin, since a preparation period is required to make sure that raw materials and proper tooling are available when needed. In the case of numerically controlled manufacturing this process preparation should take place concurrently with the operations necessary to produce and test the numerical control tapes for the specific job on order.

It should be noted that the model does not recognize transit times between one center and the other. The transaction that leaves one center simply joins the waiting line (if any) of the next station. During simulation of job shop operations the simulator does not take into consideration certain standing shop practices such as lap phasing, job-lot splitting and the saving of setups. Lap phasing is a form of parallel scheduling in which a lot can be simultaneously processed on at least two machines. As far as capacity is concerned, the model, once defined in its structure and mode of operation for a specific simulation run, reflects only a fixed machine tool capacity.

As discussed previously, the processing facilities are grouped in centers along specific routings. The transaction of the part program which follows a common routing in the N/C data processing system is channelled in a specific routing as soon as it reaches the physical processing system, according to part configuration and to the technology of the machining process. The differences in the machining process are primarily dictated by the geometry of the piece-part, the type of N/C machine tool, the way in which material is removed and the type of material being machined. The model takes into consideration machining centers of entirely different nature where the discrete and/or continuous process of metal removing produces parts shaped as solids of revolution (lathe operation), or bounded by surfaces ma-

chined in three-coordinate cartesian space (milling operations) or by removing metal at discrete positions as by drilling or spot facing (multi-purpose machine). No exchange of routings is allowed for a transaction in case of excessive machine loads or machine breakdowns.

There is no allowance for interaction among similar production orders at the various facilities. The possibility of accumulating jobs requiring nearly identical setups in order to save setup times is not considered. Other important assumptions concerning the shop operations are the absence of delays due to the work piece not being available at the scheduled time at the machine site, or delays due to cutting tools, holding fixtures, special tooling, and N/C tapes not available at scheduled machining time.

Rejects or scrap due to fabrication errors or machine tool programming errors recorded in the tape program are assumed impossible in the simulation. Although in the case of conventional machine shops this assumption may seem quite unrealistic, it may hold true in many cases of efficiently run numerical control shops where validation procedures in tape programming and tape testing are enforced and machine tool operator intervention is kept at a minimum. It is also assumed that machining breakdown does not cause any scrap or interruption of the machine cycle during a specific part production cycle.

In the simulation of the man-machine system described in this paper, consideration of the human element leads to the following basic assumptions. Part programmers can be assigned to any part programming job irrespective of the N/C machining process or type of N/C machine tool requiring programming. The part programmer responsible for the symbolic programming of a specific drawing also runs the drafting machine for visual verification of the part contours and tool paths and attends the first machining tool run for the tape proofing operation.

Although machine tool operators cannot be exchanged for part programmers, complete flexibility is assumed in shifting operators from one N/C machining center to another. Machine tool operators in this model need no additional training or special skills, even though the same man may run an N/C lathe or a 3-axis N/C milling machine, which in practice is a reasonable assumption. Machine tool operators and part programmers are assumed always present at processing stations as no consideration in the simulation is given to fatigue, personal time and absenteeism. Machine tool operators are expected not to intervene during the machining process, while in real life situations the operator may slow down cutting rates and feed rates due to piece-parts showing different characteristics in their raw material piece form or in the quality of the raw material.

Secondary operations are not needed as supporting machinery of conventional type is assumed not present in the system. Shop operations sometimes used in preparation of forgings and castings for the N/C process are not considered necessary or relevant for the simulation. Deburring, degreasing or heat-treat operations are not included. Part inspection is not considered in the simulation model although inspection with measuring devices mounted on the N/C machine, or use of tape-controlled inspection machines is technically feasible.

Preventive maintenance does not occur in the scheduled production work period and all emergency maintenance takes place during scheduled work time. No attempt is made to combine the two types of maintenance operations. Finally, adequate storage space is assumed available in front of machining centers and in job-lot staging areas; saturation of storage areas is not considered in the simulation in case of extreme build-up in machine center queues.

A few constraints in the simulation described in this paper are concerned with the limitations in system definition imposed by the GPSS program itself. Computer simulation programs, by necessity, have boundaries in the amount of data they can retain at any stage of the process, the number of attributes that can be assigned to transactions, the number of variables and composition of variables that can be defined in the system and, obviously, the computer time that can be economically spent on any one simulation run.

The physical processing system

The Physical Processing System represents the physical processes and the resources required to produce the end product, i.e., machined parts. To control the scale of the model built with the methodology discussed here, the physical processing facilities were limited to three multi-machine centers of different type. However, from one to four numerically controlled machine tools can be operated within each center. The organization of the machine centers is of sufficient detail to permit useful evaluation of congestion conditions in shop-order flow through the centers. It also allows the analysis of the effects of overloading in the Numerical Control Data Processing System and of the effects of intervention by the Emergency Repair Support System in case of machine breakdown at any of the centers.

Each machine tool is characterized by a capacity represented by the maximum work time period that may be allowed daily for the machine in the shop. Machine capacity is measured in minutes and makes reference to the three capacity levels set by first, second or third shift of work. Provision is made for the machines to work five days, six days or seven days per week. Flexibility is provided by the model in the number of daily shifts or du-

ration of the weekly work period. This arrangement is quite similar to actual working situations in numerical control machining operations since numerical control machines on determined shop routings are, in practice, shut down without affecting the operation of the remaining centers. If desired or deemed necessary, machine centers can be operated independently on entirely different schedules one from the other. One of the striking characteristics of numerically controlled machinery is the consolidation of many machine tools of conventional type in one processing center.

The emergency repair support system

The simulation of breakdown repair operations in a machine shop has all the elements of a queueing situation. Machine breakdowns in the shop require repairs which are treated as jobs to be performed. The maintenance man becomes the service facility for which competition may generate when more than one machine is down at any given time. The time necessary to repair the facility represents the service or processing time.

In the N/C production system simulation breakdowns occur as a result of the intervention of a separate GPSS model, which is run concurrently with the main program to simulate the emergency maintenance operations required to repair the malfunction or malfunctions which have occurred in a single facility unit. Simulated breakdowns are classified by type of fault: i.e., electronic, hydraulic or mechanical, and by the type of machine, with faults occurring on multi-purpose machines, milling machines and lathes.

The production information system

The Generalized Model Schematic Diagram of Figure 4 illustrates the various linkages of the Production Information System. The sensing elements shown in the schematic diagram allow management to follow the flow of shop orders through the Numerical Control Data Processing System and the Physical Processing System, and the stoppage of shop orders through the Emergency Repair Support Subsystem. It is then possible to study the effects of congestion caused by the need of accessing processing stations, or caused by limitations in the capacity of sections of the network.

Corrective action may be issued to affect the operations of one or more subsystems. Changes are not issued during a particular simulation itself, but they are rather the result of decisions made in view of the output supplied at the end of the simulation, which covers a predetermined time span. Changes in scheduling procedures, manpower and machine capacities, or changes in incoming order configuration can be introduced in order

to improve the over-all performance and to evaluate specific measures of effectiveness.

The generalized simulation methodology is designed to study the flow of job orders through the N/C production system and to analyze how the system reacts to various conditions of operation. The output data obtained from the simulation is, therefore, in terms of physical performance of the total system, its subsystems and their components. There are many measures of physical performance of the total system, its subsystems and their components, which can be generated for any given set of decision rules through the reporting function. These measures include the transit times required for an order to flow through the system or sections of the system, the number of orders in delay, the length of delay times at various points of congestion, the degree of utilization of the facilities or the utilization of manpower. The total flow time—or time from the receipt of a production order to the completion of that order for delivery to another department or directly to the customer—represents the best over-all measurement of physical performance. This measure does not provide a direct economic evaluation of performance. The measures of performance which can be obtained from the N/C Production Systems Simulator may be grouped in the five categories which follow:

- I. Output Data on Queues Forming at Processing Centers
 - Distribution of queue delay times at N/C processing centers and N/C data processing centers.
 - Maximum and average contents of queues at pre-processing centers.
 - Average length of time spent by production order in queue of specific processing center.
 - Record of queue lengths as a function of time taken at selected queues of the system.
- II. Output Data on Facilities Control
 - Time covering period of operation of facilities.
 - Average utilization of each facility based on period of operation. This also constitutes the loading factor of the facility.
 - Total loading levels and capacity levels at each N/C machine center, over period of operation considered.
- III. Output Data on Production Order Control
 - Total number of job-lot orders accepted and completed by system during period of operation considered.
 - Average flow time of new orders through N/C data processing system and through total production system, including physical processing system.
 - Average flow time of reorders through physical

system, and average flow time of reorders and new orders combined through total system.

- IV. Output Data on Operating Schedule Control
 - Number of manufacturing days and manufacturing weeks covered by scheduling.
 - Number of manufacturing days expected to process specific new order or reorder, and number of manufacturing weeks when specific new order or reorder is expected to be processed at machine center.
- V. Output Data on Breakdown Repair Control
 - Distribution of queue delay times at machine center for the handling of repair order.
 - Total number of breakdowns which occurred at specific machine center.
 - Maximum and average contents of queue for handling of repairs at machine center.
 - Average time spent for repairs at machine center.
 - Average utilization of repair technicians assigned to machine center.

Experimental investigation

The system configurations analyzed in the experiment comprised the majority of components expected to be found in an integrated computer-based N/C production system. It is felt that the type, the number and relationship of components used in the model, as well as the operating policies employed in the simulation, give a complete demonstration of the use of the methodology. The physical processing subsystem and the N/C data processing subsystem defined in the model used, have been structured to incorporate, as much as possible, the dynamic complexity to be found in numerically controlled manufacturing operations using computer-assisted programming.

The primary objective of the experiments was to determine whether significant differences in performance could result from changes in selected design aspects of the two major subsystems. The discussion of the problem which prompted the subject of the research indicated that fully integrated computer-based N/C manufacturing systems are not in wide use in industry today. Of the few in operation, integration was achieved as an evolutionary process rather than on the basis of a completely new approach to N/C production system design.

The formulation of the simulation model was based on a large amount of data on N/C machine characteristics and N/C production operations obtained from various industry sources over a period of several years. Information processing times, physical plant characteristics and actual performance factors of man-machine components of the system were obtained from aerospace manufacturers and research institutions.

The tests showed that the behavior of the model is acceptable and contained within realistic limits. During the many runs preceding the final tests, the model was manipulated over wider ranges of operations than the ones which are normally encountered in actual system operations. The effects of changes in critical design parameters were examined and the computer results compared with results obtained from a base configuration of the system structures under study.

The objectives of the experiments carried out were as follows:

1. To structure the model in such a form to adequately represent a typical N/C production system of metal-working manufacturing.
2. To set up the model in such a way as to use operating policies normally encountered in N/C job-shop operations.
3. To obtain steady-state statistics for a variety of configurations in the numerical control data processing subsystem.
4. To carry out a statistical analysis on the change of system performance and to compare statistics generated for the new system configurations.
5. To study the dynamic relationships between queueing conditions in the N/C data processing system and the physical processing system.

As to the characteristics of the particular model used in the experiments, assumptions were made on the loading of the production system and the system operating rules based on actual operating data. The generalized model is structured so that it produces a collection of statistics for later analysis and for conducting, through changes in model geometry and through operations manipulations, more experiments to obtain further statistics for comparative purposes.

The study of the behavior of the N/C production system model, carried out as part of the experimental investigation, was based on eight different measures of performance. "Flow time" was defined as the difference between the time at which an order (or transaction) arrives at the system or subsystem, and the time at which the same order (or transaction) leaves the same system or subsystem. For convenience, the term "transit time" may be used alternatively for flow time.

The first four measures of performance investigated were:

1. Flow times of new orders through the numerical control data processing system.
2. Flow times of new orders through the total production system.
3. Flow times of reorders through the physical processing system.
4. Flow times of all orders through the total production system.

The flow times of reorders relate to the physical processing system only. It should be recalled that reorders in N/C production do not require the preparation of machine tool tapes by the N/C data processing system since these N/C tapes can be made available simply by requesting them at the tool room where they are kept in storage racks after being used for the processing of previous production orders.

Although this paper does not cover the complete set of experiments performed and related analyses, one of the most interesting conclusions drawn from studying the simulation output is the behavior of the model under a configuration of the system operating with three part programmers. The flow times through the physical processing system and total system appeared to be consistently shorter when the simulation is run with three part programmers. This seems to confirm the practice in industry which aims at assigning one part programmer per N/C machine. This assignment is particularly necessary when programming jobs are of complex nature, as in the case of milling machine tape preparation. The low flow-time levels of the specific configuration investigated may also be the result of interaction between new orders and reorders when congestion develops at the part programming center. One effect of this interaction is the reduction of new order arrivals at the N/C machine centers.

Separate dynamic analysis runs were carried out for each one of the system configurations chosen for the study. The different configurations were characterized by the part programming function simulated with one part programmer, three part programmers and nine part programmers, identical to those introduced to obtain steady-state statistics. The results of dynamic behavior tests showed a high degree of interaction between the part programming operation and the machining operations at the N/C centers, with marked changes in the rate of build-up and decay of waiting lines at the various processing points. These variations seemed to be quite evident in the system configuration having three part programmers. It should be noticed that the peaks at the part programming queue were followed closely by peaks of varying amplitude at the N/C machine centers. It was apparent that the build-up of queues at the part programming center depended upon the arrival rates of new orders and the part program processing rates. The build-up was also influenced by the three feedback loops of the APT diagnostics check-out, the N/C drafting machine output check-out and the N/C tape proofing operations. It should be recalled that the part programming operation, as represented in the model, is constrained by work shifts which are dissimilar from the work shifts governing the N/C machine center operation. The different operating rules caused a

pulsing effect in the operation of N/C data processing facilities.

The physical processing system seemed to be quite sensitive to variations in the part programming center queue. The release of N/C tapes is an intermittent operation, which is paced by work shifts. It may present peaks of different intensity, according to factors such as part complexity and number of reworkings of part programming jobs. Correspondingly, build-up at the waiting lines in front of the machining centers may well depend on the size of part programs, size of job-lots, timing of N/C tapes releases and on the processing rates governing the operation at the N/C machine center.

To summarize, the comparisons made for the four different measures of performance and the dynamic analysis show, both by visual comparison and by statistical test, that the simulator is highly sensitive to changes in the part programming manpower levels.

CONCLUSIONS

Numerical control is a production technology that forces management to look at a manufacturing enterprise as a whole, in the interdependency of all the steps from product conceptualization through the processes that yield the finished product. Fundamental to increasing the success of numerical control and the rate of acceptance of N/C machinery, is the systems approach to forward thinking in this new area. This mode of attack views all components of a N/C production system in the light of their interrelationships and their functional objectives. It has been applied with outstanding success in the development and operation of complex computing and communication networks, especially by the requirements of intricate and interlocking systems of military installations.

It is felt that the use of advanced techniques of digital simulation may become in the near future a major contribution to the development of a body of knowledge on application of numerical control to discrete-type production and to job-shop manufacturing systems. A purpose that may be pursued by research through examining large-scale models could be to show how N/C production and related activities, particularly decision-making and control of shop operations, can be explicitly modelled. The research should be aimed at the major goal of developing and refining a methodology for building production models embodying the new principles of N/C manufacturing through the use of computer simulation.

Finally, it can be said that the discipline of reducing production organizational concepts to precise formulations, and the requirements of specifying decision

rules and procedures in a straightforward unambiguous fashion is undoubtedly an education in itself, whether for the N/C user or the the manufacturers of computers and numerical controls.

BIBLIOGRAPHY

- APT encyclopedia*
1108 Multiprocessor System Reference Manual Up-4078
Univac Data Processing Division 1968
E L BUFFA
- Models for production and operations management*
John Wiley and Sons New York 1963
D F BOYD H S KRASNOW A C R PETIT
Simulation of an integrated steel mill
IBM Systems Journal Vol 3 No 1 1964
L BROTHMAN J MINKER
- Digital simulation of complete traffic problems in communications systems*
Operations Research Vol 5 No 5 October 1957
S A BROWN C E DRAYTON B MITTMAN
A description of the APT language
Communications of the Association for Computing Machinery
Vol 6 No 11 November 1963
G CHINGARI
A methodology for the simulation of numerically controlled production systems
Ph.D Dissertation University of California Los Angeles 1966
R W CONWAY
Some tactical problems in digital simulation
Management Science October 1963
General purpose systems simulator II
1108 Multiprocessor System Reference Manual UP-4129
Univac Data Processing Division 1966
W B JOHNSON
Total NC system—A new management tool
Metalworking January 1963
E LEGRANDE
The development of a factory simulation using actual operating data
Management Technology May 1963
Machine tools with numerical and prerecorded motion program controls
United States Department of Commerce Series BDSAF-630-1
Publication 1963
Numerical control in manufacturing
American Society of Tool and Manufacturing Engineers
McGraw-Hill Book Company Inc 1963
Proceedings of industrial symposium on numerical control data
Department of Defense Tinker Air Force Base Oklahoma October 1966
D T ROSS S A COONS J E WARD
Investigations in computer-aided design for numerically controlled production
MIT Report ESL-1R-241 Electrical Engineering Department
MIT 1965
D T ROSS
Data processing for numerical control in the USA
IFIP Congress 62 North-Holland Publishing Company Amsterdam 1963
A REISMAN E S BUFFA
A general model for production and operations systems
Management Science Vol 11 No 1 September 1964

A J ROWE

Applications of computer simulation to production system design
Modern Approaches to Production Planning and Control American Management Association Inc 1960

L K WILLIAMS C B WILLIAMS

The impact of numerically controlled equipment on factor organization
California Management Review Winter 1964

Subsets and modular features of standard APT

by CLARENCE G. FELDMANN

Massachusetts Institute of Technology*
Cambridge, Massachusetts

INTRODUCTION

The APT (Automatically Programmed Tool) N/C language was developed in 1956 at M.I.T.'s Servomechanisms Laboratory by D. T. Ross.¹ Since that time, the language has been widely used at N/C installations in the United States for all categories of N/C programming. Since 1961, further APT System development has been directed by the Illinois Institute of Technology Research Institute under Dr. S. Hori.^{2,3,4,5,6} Use of the APT Language became so universally accepted that in 1963, the American Standards Association (now the United States of America Standards Institute) initiated an activity to generate a United States standard for the APT Language.

Since the original organizational meeting in 1963, the USASI X3.4.7 Subcommittee has been preparing a Standard for the APT N/C language. This activity early encountered a unique problem in that the APT language had been, since its first industrial use in about 1958, an evolutionary computer system, designed for any user who had the computing facilities to add useful modules onto the system. Although the APT language began as a problem-oriented language for three-dimensional milling, through the process of evolution, it soon added language for drafting, lofting analysis, turning, boring, flame-cutting, lathe control, and whatever other user's peripheral areas of interest came along.

One of the basic criteria agreed upon by the members of USASI X3.4.7 is that a standard language would be

of little use if no user or computing equipment manufacturer could reasonably implement a processor for the language. On the other hand, if only a small "core" subset of APT were standardized, this would not be very useful to industry either.

APT subsets and modules

To resolve this problem, it was decided to standardize the entire language, and to also designate certain classes of language syntax rules in APT as belonging to a particular "subset" or "modular feature" of the language.

The terms "APT Subset" and "APT Modular Feature" take on very specific connotations, and must be carefully defined before proceeding. In general, the total APT Language is divided into five basic subsets, where each subset is designed for a specific level of N/C programming and hardware. The subsets are "nested" in that each subset wholly contains all statements in the next smaller subset. The five APT Subsets are pictured below.

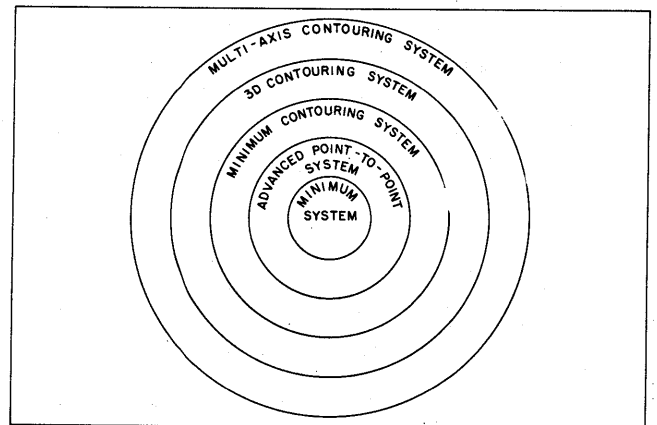


FIGURE 1—APT subsets

*The work reported in this document has been made possible through support and sponsorship extended to the M.I.T. Electronic Systems Laboratory by the Manufacturing Technology Laboratory, RTD, Wright-Patterson Air Force Base under Contract F33 615-67-C-1530. Project DSR 70429.

The work reported herein was supported (in part) by Project MAC, and M.I.T. Research Program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract NOnr-4102(01).

The APT Subsets are designed as "core" languages, i. e., they contain just enough basic facilities to provide a usable language for the indicated level of N/C programming. The largest subset (Multi-Axis Contouring) therefore does *not* contain all APT statements. It was the vote of the USASI X3.4.7 subcommittee that it would be undesirable to specify large, unwieldy language subsets in which many of the statements are not used at most APT installations. Instead, the "core" subset philosophy was adopted, and the less-essential, special-purpose language "packages" which have been developed over the years by APT users were categorized into a set of "modular features" which can be selected by anyone having a particular need for that feature and "plugged in" to a core subset. There are at present 25 such modular features being considered by X3.4.7

Another aspect of the Modular Feature approach is that each module has a "natural" subset below which its use is not feasible. For example, it does not make sense to plug in the "Fitted Surfaces" modular feature to a point-to-point subset. Each APT modular feature therefore has a "minimum subset" associated with it.

Relation to USASI X3.4 criteria

The USASI Subcommittee for "Common Programming Languages" (X3.4) is currently working on a document entitled "Procedures for the Standardization Process" which, among other things, contains a section on "Determination of Language Subsets." At this point, it is of interest to discuss their work in this area and show the relationship of the X3.4.7 language breakdown.

The X3.4 document discusses language subsets, but does not describe a "modular feature" setup as adopted by X3.4.7. In defining subsets, they state:

"It is assumed to be a basic requirement of any establishment of subsets that there be two clearly recognizable extreme cases: the maximal language, L , and the minimum feasible subset, N (for nucleus). The nucleus language, N , is the agreed upon minimum version of the language. At least N must be completely specified as a single standard language.

The maximal language, L , includes all facilities incorporated as features in a single language. L need not be a standard language.

Whether or not L is a standard language, it is desirable that there be a complete specification for L .

If there are, as well, additional subsets I_1, I_2, \dots, I_m then it is required that (1) each I_j be a proper

subset of L and (2) N be a proper subset of each I_j .

In set-theoretic notation,

$$L \supset I_1 \cup I_2 \cup \dots \cup I_m \quad (1)$$

$$N \subset I_1 \cap I_2 \cap \dots \cap I_m \cap L \quad (2)$$

where " \cup " denotes the union of features and " \cap " denotes the intersection of features (selection of only those common to both the two operands)."

The APT subsets comply with these criteria, where N corresponds to the APT "Minimum System" subset and L corresponds to all statements in all subsets and modular features taken together. In fact, we have seen that the APT subsets satisfy the further criterion (not required by X3.4) of being nested. Stating this in an additional formula,

$$I_1 \supset I_2 \supset I_3 \supset \dots \supset I_m \quad (3)$$

The USASI X3.4 document describes a *module* approach as an equivalent way of defining language subsets. However, this description does not fit the APT modular feature approach and the two should not be confused.

In an attempt to put the modular feature approach into set-theoretic notation, consider the modular feature element

$$MF_{sm}$$

where s is a minimum recommended subset number and m is a modular feature number. An APT Language A_ℓ may thus be built from any combination of the form

$$A_\ell = I_j \quad (4)$$

or
$$A_\ell = I_j \cup MF_{hm} \quad (5)$$

or
$$A_\ell = I_j \cup MF_{hm} \cup MF_{hm} \quad (6)$$

etc.

for any subset number $k \leq j$ and $p \leq j$, where I_j represents APT Subset j .

To complete the definition of APT modular features, the following additional relationship should be stated:

$$I_j \cap MF_{hm} = \varphi \text{ (the null set)} \quad (7)$$

for all j, s , and m .

In other words, no modular features have any statements in common with any of the subsets.

Subset and module breakdown

With the above definitions in mind, we shall now describe the various subsets and modules of the APT Language breakdown.⁸ You will recall that there are five basic subsets:

- (1) Minimum
- (2) Advanced Point-to-Point
- (3) Minimum Contouring
- (4) Three-Dimensional Contouring
- (5) Multi-Axis Contouring

The modular features associated with each of these subsets is as follows:

Subset 1: (Minimum System)

- (1) Feedrate attached to motion commands.
- (2) Copy statements to duplicate sequences, with or without translation and/or rotation.
- (3) Synonyms for APT vocabulary words.

Subset 2: (Advanced Point-to-Point)

- (1) Matrix transformations and rotations
- (2) Macro facilities
- (3) Nested definition capability
- (4) Special printing, punching, and reading (I/O language).
- (5) Point patterns
- (6) Control of multiple postprocessing.
- (7) Program looping and computing (algebraic calculations, functions for square roots, etc.)
- (8) Advanced point-line-circle definitions.

Subset 3: (Minimum Contouring)

- (1) Conic Section contouring, including "LOFT conics"
- (2) Surface "thickness" capability
- (3) Conditional cut termination program control (what to do next depending upon how the last cut sequence terminated)
- (4) Deletion of specified cutting sequences (turning off output for tool positioning).
- (5) Pocket cleanout language
- (6) Assignment and manipulation of symbolic names of surface variables.
- (7) Tabulated Cylinder Surfaces
- (8) Special "offset" startup language

Subset 4: (Three-Dimensional Contouring)

- (1) Generalized cutter capability
- (2) Additional three-dimensional surfaces
- (3) Additional vector definitions
- (4) Additional cutter-to-part control language
- (5) Fitted Surfaces
- (6) Individual surface tolerance on all surfaces.

There are no modular features having subset 5 as a recommended minimum set.

In addition, a set of post processor language modular features are being defined, but are not sufficiently finalized to be presented at this time. Also, this rather large number of modules is not necessarily the final breakdown in the standard, but is the status at the writing of this paper.

We will next go into more detail on each subset of the language. Finally, a unique computer system for defining and maintaining the standard will be described.

Minimum subset

The basic criterion which was used when deciding how small to make the minimum subset was that the subset be the bare minimum which could still be considered usable. Quoting from a draft of the X3.4.7 working paper on subsets,⁸ "The minimum numerical control system contains no geometric definition capability. The motion commands are limited to those using position coordinates or delta motions."

Specifically, the minimum subset contains the following APT Statements:

- (1) Cutter orientation statement:
FROM/x coord, y coord, z coord
- (2) Motion command:
GOTO / x coord, y coord, z coord
GODLTA / delta x, delta y, delta z
- (3) Program bounds:
PARTNO (identifying string of characters)
FINI

In addition, the minimum system postprocessor commands are included.

Clearly, the minimum language as presented above is so minimal that, except for the postprocessing control functions, it represents little more than a technique for converting the tool positions into the input format required by a machine tool. It is debatable whether some additional features such as symbolic names on points should also be included, but it was decided that a user might very well wish to use this simple language, especially if he has only a very small computer at his disposal, or if he is supplied cutter orientation data from some non-APT computer analysis routine. The latter case occurs frequently with sub-contracted jobs to small shops.

Modular features which make useful additions to the minimum subset include the ability to add feedrate values to motion commands, to define synonyms for the APT basic vocabulary words, and to COPY a point a specified number of times with or without rotation and translation.

Advanced point-to-point subset

This subset contains the minimum system syntax plus the ability to process geometric definitions and assign symbolic names to definitions of lines, circles, and points. Since the subset is still a point-to-point language, the motion commands are limited to positioning commands. A limited "ZSURF" capability is included to permit setting the Z coordinate of intersection point definitions of point definitions containing only X and Y coordinates.

One of the outstanding features of the minimum subset is the extremely limited geometric capability which includes only points specified by coordinates and delta motions. The advanced point-to-point subset adds six line definitions, four circle definitions, and six point definitions, listed below:

Line definition

- (1) Through two points described by coordinates.
- (2) Through two points described by symbols.
- (3) Through a point and tangent to a circle.
- (4) Tangent to two circles.
- (5) Through a point and making a specified angle with the positive X axis.
- (6) Through a point and parallel to another line.

Circle Definitions

- (1) By coordinates of center, and radius value
- (2) By center point and radius value.
- (3) By center point and line to which it is tangent.
- (4) By a center point and circle to which it is tangent.

Point Definitions

- (1) By x, y, and z coordinates.
- (2) By x and y coordinates.
- (3) By the intersection of two lines.
- (4) By the intersection of a line and a circle.
- (5) By the intersection of two circles.
- (6) On the circumference of a circle, located by angular measurement from the positive X axis.

From examining the above set of definitions, we see that they afford a flexible basic set of definition formats for ruler-and-compass type constructions for point definitions. Clearly, many more APT definitions exist in the library and could have been included, but this set was chosen (based on part programming experience) as the most basic.

Modular features which could reasonably be added include matrix operations, a macro facility, nested definition formats, special input-output statements for printing and punching of both cutter paths and can-

onical geometric definition data, point pattern definition and motions, algebraic and geometric operations and functions (+, -, *, /, SQRTF, SIN, etc.), and a part program looping facility. The extended set of geometric point, line, and circle definitions is also a natural module to add to this subset.

Minimum contouring subset

The Minimum Contouring Subset incorporates all features of smaller subsets and adds the ability to calculate cutter offset paths for line and circle, two-dimensional cuts. Since most common line and circle definitions were included in the previous, point-to-point subset, the most significant additional language added by this subset consists of the cutter definition statement and contour motion statements, along with a series of additional plane definitions.

Specifically, the following features are added:

- (1) Establish part surface
- (2) Cutter specification by diameter and end radius
- (3) Specify cutter path inside and outside tolerance.
- (4) Establish a sense of direction for future motions
- (5) Motion commands to go right, left, forward, and back
- (6) Startup procedure to place the tool relative to a new curve to be machined.
- (7) Cutter position commands to orient the cutter to the left, right, or on the curve.
- (8) Additional and plane definitions:
 - (a) By coefficients of a plane equation.
 - (b) By three points not in a straight line.
 - (c) Through a point and parallel to a given plane.
 - (d) Parallel to and offset from a given plane.

Reasonable modular features which could be added to the minimum contouring subset include definitions for the conic sections, and tabulated cylinders, surface "thickness" capability, conditional check surface control (motion depending upon which of several check surfaces are reached first), an automatic pocket clean-out feature, more generalized startup facilities, a "DNTCUT-CUT" feature for turning off cutter motion output during cutter positioning, and additional flexibility in obtaining and naming values in canonical geometric definitions.

Three-dimensional contouring subset

The Three-Dimensional Contouring Subset adds the vector definition language, three-dimensional cone and cylinder definition language, and a three-dimensional startup procedure. The motion commands re-

main the same except for the addition of GOUP and GODOWN.

Specifically, the following features are added:

- (1) Motion commands GOUP and GODOWN
- (2) Vector Definitions:
 - (a) By x, y, and z components
 - (b) By two points
- (3) Cone Definition:
 - (a) By coordinates of vertex, axis, vector components, and cosine of half angle.
 - (b) By vertex point, axis vector, and cosine of half angle.
- (4) Cylinder Definition:
 - (a) By coordinates of point on axis, axis vector components, and radius.
 - (b) By point on axis, axis vector, and radius.
- (5) Three-Dimensional Startup Procedure GO/...

$$\left. \begin{array}{l} \text{TO} \\ \text{ON} \\ \text{PAST} \end{array} \right\}, S1, S2, \dots$$

The modular features which are natural additions include a more general cutter definition format, additional vector and three-dimensional geometric definitions, independent tolerance controls on all surfaces, and additional control of the cutter-to-part-surface relationship.

Multi-axis contouring subset

The Multi-Axis Contouring Subset adds the ability to control the cutter axis. This may be done by explicitly specifying the axis vector values or by specifying that the axis is to remain normal to the Part Surface or Drive Surface.

Summary

In summary, we see that each subset represents a basic language for one level of numerical control programming. The smaller subsets are clearly processable on small computers, and additional modular features could easily be made available for special applications or to make a more flexible language. We have also seen that each modular feature has a "natural" subset level at which it may be applied. Clearly, the same features could also be applied to *larger* subsets, but application at a *lower* than the recommended level either makes no sense or destroys the natural subset hierarchy logic.

Lastly, it should be pointed out that a complete subset and modular feature breakdown has also been prepared for the Postprocessor language area. This

breakdown adds as much again as what has been discussed above, and is not presented in this paper.

Computer implementation

All of the above discussion has been a quick, high-level look at the APT Standard subsets and modules to give the reader a feeling for the ground rules used in breaking down the language. We now shift to the problem of rigorously defining and maintaining the actual standard.

One commonly-used technique is to produce an elaborate document, carefully defining each syntax rule by means of an appropriate "meta-language" and then further elaborating meaning through use of semantics and examples. The document, therefore, is the physical representation of the standard.

The APT Standard is likewise defined via a list of syntax rules, semantics statements, and examples. However, the physical standard exists as a *computer data deck* rather than a printed document. There also exists a computer program to read this data deck and produce a syntax and semantics document which is very similar to other language standard documents.⁹

The use of a data deck as the physical standard represents a significant step in rigorously specifying and updating the standard. Computer checking programs can be written to check syntax rules for completeness and for ambiguities. New meta-language prints can be produced automatically by the computer whenever changes or additions to the data deck are made.

More significant for the subject at hand, by including subset and modular feature data with each rule in the data deck, the complex job of producing and maintaining the subsets and modular features is greatly simplified. For example, a computer run might change a series of data cards and then produce a new print of selected subsets and modular features, as dictated by control options of the program. The resulting print represents the very latest version of the standard and shows any inconsistencies which might have been introduced by the changes.

Subset and modular feature data

The subset and modular feature data are punched into specific card columns reserved for this purpose. The data are associated only with the syntax rule data, and not with the data cards which define the vocabulary words or semantic statements. In this way, the subset and modular feature data need be punched only once, and the related basic word and semantic data are brought in through chaining of references, starting with the syntax rules. We shall see exactly how this

works in the current IBM 360 implementation of the APT Standard routines.

Data deck example

The APT Standard data deck is divided into decks for the basic words of the language, the semantics, and the syntax. Each deck is identified by a deck number punched in a column reserved for this purpose. The syntax deck contains a rule and subrule number for the rule being defined, and a series of positive integers, where integers less than a certain size represent syntax rule referents, and larger integers refer to basic words of the APT language.

For example, let us examine the syntax rule for a "point specification" as used in

GOTO / (point)

or

L = LINE / (point), (point)

where the first statement causes the cutter to move to the point specified, and the second statement defines a line "L" which passes through the two points specified.

The syntax rule which defines a point specification ("(point)", above) is given in the syntax data deck as

Rule No.	Subrule No.	Syntax
52	1	42
	2	2040 2334 2047 452 2041
	3	2040 42 2061 2334 2047 452 2041

By substituting the entry in the semantics deck for rule 52, replacing the large (2000 series) numbers by the actual basic words, and printing in a "prettier" form, the rule becomes

52 POINT SPECIFICATION

IS(1) 42

OR(2) (POINT / 452)

OR(3) (42 = POINT / 452)

This is known as the "short form" produced by one of the available computer print routines.

By substituting the names of the rule referents 42 and 452 and using the accepted Backus-Normal form, the print becomes

<POINT SPECIFICATION> ::=
<IDENTIFIER>

(POINT / <POINT PARAMETER LIST>)

(<IDENTIFIER> = POINT / <POINT
PARAMETER LIST>)

Subset and module numbers

The data giving the subset and module numbers for each rule and subrule in the syntax deck are maintained as a separate card deck. Columns are reserved in this deck for the subset or modular feature number for each rule, as well as columns for the recommended subset, if the rule falls into the modular feature category. For example,

Rule No.	Subrule	Subset	Modular Feature	Recommended Subset
		12345		12345
16	1		1	01000
18	1	01111		
50	1		1	01000
142	1	00111		

where each number shown is punched in a specific card column. These selected rules shown above give a representative sampling of the subset and module deck. We see that rule 16(1) is part of a modular feature which has subset 2(01000) as a minimum recommended subset. The modular feature number is given by the column in which the "1" appears. Rule 18(1) is a member of Subset 2. The nested subset characteristic is evident from the fact that any member of a subset also has a "1" in every higher subset card column (e.g., 01111 for rule 18(1)).

CONCLUSION

In conclusion, it should be noted that the information contained in this paper represents a particular level of the work of USASI X3.4.7. By the time the reader sees this paper, it is likely that changes will have occurred, and the latest information may be obtained by contacting the committee.

It should also be made clear that all of the work on the APT Standard has been performed by representatives of the Numerical Control community, including the Aerospace Industry, computer manufacturers, machine tool manufacturers, and universities. This background has made for an APT Standard which is generally agreed upon by a representative group of users, and should therefore be readily acceptable.

REFERENCES

- 1 *Automatic programming of numerically controlled machine tools*
A series of eleven Interim Progress Reports on the M.I.T. APT Project June 1 1956 through November 30 1959 totalling 695 pp

-
- 2 SHORI
Future of numerical control in industry
IEEE Transactions on Industrial Electronics May 1963 pp 15-21
- 3 S A BROWN C E DRAYTON B MITTMAN
A description of the APT language
Communications of the ACM Vol 6 No 11 November 1963 pp 649-658
- 4 C E DRAYTON
APT for continuous path N/C programming
Technical Paper No 576 American Society of Tool and Manufacturing Engineers 1964
- 5 P ARANDA SHORI R N LITTLE
APT—Present and future
Paper Society of Automotive Engineers Midyear meeting Detroit Michigan June 6-10 1966 Paper No 660354
- 6 R CHANDLER
Design for numerical control machinery
Machine Design Magazine February 15 1968
- 7 *Procedures for the standardization process*
USASI X3.4 Subcommittee Common Programming Languages Document No X3.4/68-1 June 25 1968
- 8 T C HARRIS J ALBERT C G FELDMANN et al
Report of the subsets and modular features subgroup
USASI X3.4.7 Working Paper No 102 June 17 1967
- 9 C W WILSON III
Computer routines for editing numerical control meta-language
June 12 1968 USASI X3.4.7. Working Paper No CWW/11

A remote processing system for the APT language

by MALCOLM E. WHITE

Radio Corporation of America
Princeton, New Jersey

INTRODUCTION

With the advent of time shared computers and the development of remote terminals capable of providing fast access to these computers, it became evident that processors could be developed to greatly increase the efficiency of an N/C part programmer. In general, a majority of the errors which occur in the processing of N/C programs are errors of syntax, and these errors cause nearly as much loss of time in processing as the more complex arithmetic errors. Therefore, any system which could provide a partial or complete interactive APT processing capability would be a valuable aid in reducing processing time. A system has been devised whereby a part programmer can sit at a remote console (teletype, etc.), and directly communicate with the computer. This system can produce output for immediate display or for punching onto a tape. Such a system requires a highly specialized processor which can accept APT language statements, check each statement for detailed errors in syntax, and then perform the necessary computations needed to produce an N/C output tape.

System description

The APT language is very large and complex and generally follows quite unique and rather rigid guidelines. Studying the APT language, one sees that, with very few exceptions, each statement consists of well-defined, syntactical elements. Thus, for the most part, it is possible to describe the language in "Backus Naur Form" (BNF).¹ Some work has been previously reported and demonstrates methods of describing APT in BNF.²

A language which can be described in BNF can be readily analyzed by a compiler type program for correctness of syntax. The RCA Basic Time

Sharing System, available on the SPECTRA 70/45 computer,³ contains a compiler, the syntax of which allows one to alter compiler functions and perform ones which are nonstandard Fortran.⁴ This computer feature allows the user to specify the general format (syntax) and the output (semantics) of his own special purpose statements, which can later be used as he desires. Thus, the user may take control of the compiler process to any extent he desires, from the addition of special-purpose extensions to the compiler, to designing a wholly new, on-line, interactive compiler.

The RCA Fortran PI compiler (a subset of USASI Fortran IV) has special non-Fortran language elements in it which are especially useful in describing the syntax of a language. The special symbol ":= " (colon equals) is interpreted to mean "is" (or "is defined as") and the symbol "/" is interpreted as a logical "or." Parentheses may be used to enclose syntactic elements or strings of elements, and thus can serve as delimiters. The symbol "\$" is used to mean the "arbitrary sequence of" a given character, string, or syntactic element. Other features which are part of the compiler language are semantic operators which allow backup in case of error, operators to do label table lookup, operators to scan real or integer expressions, etc. There is also a feature which enables the user to generate arbitrary machine language instructions for immediate execution or for placement in the output area for execution at run time. With the availability of these features it is easy to completely describe the syntactic elements, as well as the semantics of a language such as APT.

Program operation

An APT programmer may sit at his teletype console and type statements, a record at a time.

Upon completion of each record (signified by a carriage return), the computer syntax analysis takes place. After each correct statement the computer returns a line feed, carriage return, and a sequence number awaiting further input. The execution of the interactive APT processor is the same as that of most compiler programs. After the interpretation and compilation has taken place, the program can be executed. There is a feature available on the RCA time-shared system which allows the execution of statements a line at a time, after they are compiled. This feature facilitates the use of on-line plotting of statements, one at a time.

The user may make corrections or modifications to his APT program by using the text editor available as part of the RCA Basic Time Shared System. The editor allows insertion of records anywhere in the program or corrections to existing records.

Syntax checking (compilation)

Using the time-shared Fortran PI compiler, an interactive APT processor has been written. The program operates in much the same manner as a typical time-shared, interactive computer program. Each input record (APT statement) is scanned to determine its validity according to the syntactic rules set forth in BNF. This scanning process (compilation) produces machine language instructions which then can be executed in order to produce APT output.

As each APT statement is scanned, it is classified according to type (cutter motion statement, surface definition statement, APT post-processor word, etc.). If the statement is determined to be an allowable APT postprocessor word, the program branches to that section (subroutine) which then checks the entire record (calling other subroutines as needed) for the proper occurrence and usage of syntactic elements. The presence of the word "GO" as the first two characters of a record may signify a cutter motion statement and causes a transfer to a subroutine which checks the remainder of the record for correctness of syntax.

If the record does not fit one of these two categories, it is tested to determine if it is a surface definition statement. Such a statement, in a record by itself, will always be headed by a symbolic name. Thus, the statement is first tested to ascertain if the symbol is a valid one. If the symbol

is acceptable, it is placed in a label table (the same label table used by the Fortran PI Compiler). Upon determination of the type of surface definition specified, the label table is again entered, and the label is classified according to the type of surface defined. This is especially important because nested definitions and backward references to symbolically defined surfaces are allowed. After classifying the label, the surface definition is checked for completeness and accuracy. The number and type of APT surface definition statements is large, and therefore the interactive syntax checker must carefully check each possibility, and back up in case of error to check other logical branches, where applicable. The APT Compiler completely checks all symbols, flagging multiply defined ones, and checking all referenced symbols to determine if they are properly used.

An APT Surface is often defined in terms of other symbolically defined surfaces, and it is possible that these referenced surfaces can be nested. This means that the APT compiler must be able to analyze any arbitrary surface definition while it is scanning definition or motion statements. It is a simple matter to define a surface in terms of a similar surface type (i.e., to define a line which is parallel to another line). Such a statement is analyzed by the compiler by use of recursive subroutine calls, another special feature of the RCA Fortran PI Compiler.

During the compilation process the compiler, as it scans a record, generates executable machine language code. In the case of a surface definition, the code which is generated is that which will place the named variables into Fortran Common for subsequent execution. For cutter motion statements, the code for moving the canonical forms of all referenced surfaces (drive, part and check surfaces) into the next available common location is generated. Finally, at the conclusion of the analysis of each record, a link is generated to the appropriate Fortran subroutine where further analysis will occur at run time.

The program scans each record, a character at a time, halting when an error is detected. Thus if there is more than one error in any one statement, only the first will be detected and flagged. When an error is found, the computer responds with a print of the characters immediately preceding the incorrect ones and will insert a question mark after the character (or string of characters) in error.

Because the APT compiler is actually embedded in, and is in fact an adjunct to the Fortran PI Compiler, it is possible to intermix APT statements and Fortran statements in any desired manner. It is feasible to use all the computation features of the full Fortran PI compiler to help describe the mathematical shape of some surface. In addition, the standard Fortran looping capabilities are available (arithmetic and logical IF statements, DO loops, computed GOTO statements, etc.). It is also possible to use all the read and write features available in the Fortran PI language, and therefore, the user has available a much broader language capability than is normally provided by the APT language.

The number and type of APT statements which can be interpreted by the interactive syntax analysis program is limited only by the size of the program which can be operated in the RCA Basic Time Shared System. The interactive APT processor contains not only syntax analysis programs, but also a number of Fortran subroutines used for producing APT output. Due to the present limited size of the programs which can be run efficiently on the time-shared system, the number of allowable APT statements has been restricted to those which are vital to a two-dimensional APT part program. A list of these statements appears in Table 1. A syntax analysis program which does not generate any semantics has been written and encompasses a large part of the allowable APT statements, including all point, line, circle and plane definitions, all cutter motion statements, and a considerable number of miscellaneous APT vocabulary definitions. This program could easily be expanded to handle the syntax analysis of the entire APT vocabulary.

Program execution

Upon completion of the syntax analysis of any input record, the generated code can be executed. The execution of an APT surface definition statement begins by branching to a Fortran Subroutine where the canonical form of that surface is generated. The program has the capability of generating canonical forms for a number of different definition formats of points, lines, circles, and planes. Each surface is handled by a unique subroutine. The output of each surface generation subroutine is left in the lowest available common locations, and upon return to the calling program,

TABLE I—Allowable APT statements

-
- I. Surface Definition Statements
- A. Point Definitions
1. SYM = POINT/X,Y,Z
 2. SYM = POINT/X,Y
 3. SYM = POINT/(*), INTOF, (SYM. LINE), (SYM. CIRCLE)
 4. SYM = POINT/(*), INTOF, (SYM. CIRCLE), (SYM. CIRCLE)
 5. SYM = POINT/INTOF, (SYM. LINE), (SYM. LINE)
- B. Line Definitions
1. SYM = LINE/X,Y,Z,X1,Y1,Z1
 2. SYM = LINE/X,Y,X1,Y1
 3. SYM = LINE/(SYM. POINT), (SYM. POINT)
 4. SYM = LINE/(**), TANTO, (SYM. CIRCLE), (**), TANTO, (SYM. CIRCLE)
 5. SYM = LINE/(SYM. POINT), (**), TANTO, (SYM. CIRCLE)
- C. Circle Definitions
1. SYM = CIRCLE/X,Y,Z,R
 2. SYM = CIRCLE/X,Y,R
 3. SYM = CIRCLE/CENTER, X,Y,Z, RADIUS, R
 4. SYM = CIRCLE/CENTER, (SYM. POINT), RADIUS, R
 5. SYM = CIRCLE/(*), (SYM. LINE), (*), (SYM. LINE), RADIUS, R
- D. Plane Definitions
1. SYM = PLANE/I,J,K,D
- II. Cutter Motion Statements
- A. Point-to-Point
1. GOTO/X,Y,Z
 2. GOTO/(SYM. POINT)
 3. GODLTA/X,Y,Z
 4. FROM/X,Y,Z
 5. FROM/(SYM. POINT)
- B. Continuous Path Motion Statements
1. GO/(***), (SYM. SURFACE)
 2. GO/(***), (SYM. SURFACE), (***), (SYM. PLANE)
 3. GO/(***), (SYM. SURFACE), (***), (SYM. PLANE), (***), (SYM. SURFACE)
 4. (****)/(SYM. SURFACE), (***), (SYM. SURFACE)
- (*) = XLARGE, XSMALL, YLARGE, YSMALL
- (**) = RIGHT, LEFT
- (***) = TO, ON, PAST, OR NOTHING
- (****) = GOLFT, GORGT, GOFWD, GOBACK
- III. Other Statements
- A. TOLER/(VALUE)
 - B. OUTTOL/(VALUE)
 - C. INTOL/(VALUE)
 - D. CLPRNT
 - E. CUTTER/(VALUE)
 - F. MACHIN/(VARIABLE LIST)
 - G. ORIGIN/X,Y,Z
 - H. FINI
 - I. SCALE/(VALUE)
-

this canonical form is stored in the reentrant storage area, by the calling program. A reference to that area in reentrant storage had been previously generated during the compilation process, and thus backward references to these surfaces are permitted.

The algorithms used to generate the surface canonical forms resemble, as much as possible, those contained in the standard APT system. The format of the canonical forms is also the same as in the APT system, with the exception that a circle is defined only as its center coordinates and radius, but does not include the components of the unit normal vector through its center. Each canonical form is described by five words in core, the first of which is a code describing the type of surface and the remaining four being the canonical form. For a point, the canonical form consists of the x, y, z coordinates; the fourth word is always zero. For a line, the canonical form consists of the components of a unit normal vector to the line, and the absolute value of the distance from the origin to the line. The canonical form of a plane is the same as that for a line because in APT usage a line is considered to be a plane which is normal to the X-Y plane.

During the compilation of a cutter motion statement, code is generated for moving the canonical form of the named surfaces into the lowest available common location. If the named drive or check surface is nested, the routine which generates the canonical form leaves this data in the lowest available common location. Therefore, the routine which computes the cutter center path can always find the required surface data in a specific common location.

The determination of the proper drive and check surface intersection is handled by developing two new surfaces (line or circle) which are parallel (or concentric) to the original surfaces, but displaced from them an amount equal to one-half the cutter diameter (for cutter offset condition). When the two new surfaces have been determined, the proper intersection of these surfaces is found, and this exact intersection point is used as the cutter stopping point for that particular symbolic instruction.

Since the present interactive APT system is limited to only four surface types, there are only four intersection problems which must be solved—namely, the solution of line drive surface to line check surface, line drive surface to circle check

surface, circle drive surface to line check surface, and circle drive surface to circle check surface. Since the line and plane have the same canonical form, they are treated as one. The intersection solutions are found for any value of cutter offset, and in the case of a circle drive surface, the circular interpolation data is automatically computed to facilitate post processing. In the case of the circle drive surface to circle check surface, both intersections are computed, a line is constructed through these points, and the problem is then reduced to that of the circle drive surface to line check surface.

The problem of cutter path generation is greatly simplified over the traditional APT cutter path solution (which is an iterative procedure) because of the fact that the interactive APT processor is limited to two dimensions. Since only the intersection of circles and lines need be solved, it is possible to find an exact, closed form solution to the cutter center path. The limitations of a program to only two-dimensional parts is not as severe as might be thought because a great majority of N/C parts are limited to only two dimensions.

Circular interpolation is performed whenever the cutter is moving along a circle drive surface. The starting and ending points are used to determine the angle through which the cutter must move, and the number of steps needed to maintain the specified tolerance is computed. The intermediate points are calculated with a straightforward trigonometric computation. Tolerance is specified in the usual APT manner with a tolerance statement. A default tolerance of .002 inch is used if no tolerance is specified.

During execution, the APT cutter center path output is passed to a Fortran subroutine which generates output in a format acceptable for plotting on a storage oscilloscope or a digital incremental plotter. This formulated output data is transmitted from the computer back to the teletype console, then through a hardware interface to one of the plotting devices.⁵ With the added capability of on-line instantaneous plots of 2-dimensional APT part programs, errors of part description and cutter motion can be quickly determined by visual inspection. That is, it is possible to type, analyze, and plot APT statements a line at a time. Thus one can very quickly determine the validity of one's part program and drastically reduce turn around time.

At the time of writing, no post processors have

been developed for the interactive APT compiler, but it would not be a difficult task to generate them as needed. It is not likely that a post processor would fit in the same file as the interactive APT processor due to the present program size limitations of the RCA Basic Time Shared System. Therefore, it is expected that any post processors which are developed would have to operate as a separate autonomous program. This is not considered to be a limitation, however, because one can be reasonably certain that his program is valid by using the interactive compiler and instantaneous plotting capabilities.

Remote batch processing of APT

If one finds that the capabilities of the interactive APT processor do not fulfill his needs, it is possible to use the remote terminal to communicate with the APT batch processing system. The part programmer can create his symbolic source file (in the time-shared environment) in exactly the same manner as described earlier. The interactive syntax analysis program is used to determine the validity of each statement, as it is typed.

After analysis of each record, its symbolic source code as well as the generated object code can be deleted. The original APT program can be saved on an auxiliary file for processing by the APT batch processing system. Such a file would consist of a series of card image records (72 characters maximum) which would then serve as input to the batch processing APT system. The processed APT input file is headed by a pair of records which indicate that it is a file to be transferred from the time shared environment (via a switching module) to the Tape-Disc Operating System (TDOS) monitor batch input tape for subsequent processing. Thus, it is necessary that all control records be an integral part of the file.

As a part of system housekeeping, the time-shared files are checked periodically to determine whether there is any data for transferral to the TDOS monitor batch input tape, and if so, the data is copied without removing it physically from the user's file in the time-shared system. In order to avoid redundant processing of this data, a new header record is written by the executive routine, indicating the date and time of transferral.

At the conclusion of APT batch processing, the user has several options available to him. He may

proceed to the computer center to examine his output. If it appears satisfactory, he can then ask to have the output punched. If there was a program error, he then must make corrections to his source file. He may, by using the time-shared system's text editor, make corrections to his source file, and the job can then be submitted for another run. Such a process does not save much turn around time, but does allow one to communicate with the batch processing system through phone lines from a distant station, and does guarantee a source program which is free of syntax errors.

It is possible, if the output file is not too large, that the user may request his output be switched to his time-shared users file for subsequent processing. In this mode of operation the user may examine the APT output data (CLFILE), and print all error diagnostics at his teletype. He may also selectively scan the CLFILE in order to determine if the proper tool path has been generated. Naturally, if an error was made, he may elect to edit his data file, without reprocessing the APT job. It will then be suitable for post processing.

Once all APT processor errors are eliminated and postprocessing is complete, the user has several options available to him. He may request that his tape be punched at the computer center and mailed to him, or, if the job is of a relatively short nature, he can ask that the output be transmitted to him for punching at his teletype.

CONCLUSIONS

The system described in this paper has been developed with the idea of providing an N/C part programmer with the capability of producing his N/C tapes as quickly as possible. The interactive APT system with its on-line graphic output provides an N/C programmer an extremely efficient method of generating and debugging his part program. It is now possible to produce a plot of an APT program as quickly as one can type APT statements at a teletype console. Post processing for a specific machine tool can be accomplished very rapidly after final debugging of the part program. Thus, the programmer has the ability to sit at his console (which can be located in his office), and produced an N/C machine tool control tape within a matter of minutes, rather than hours or days (or longer), as is generally the case with batch processing computer systems.

REFERENCES

- 1 P NAUR
Revised report on the algorithmic language ALGOL 60
Communications of the ACM January 1963
- 2 S A BROWN C E DRAYTON B MITTMAN
A description of the APT language
Communications of the ACM November 1963
- 3 *BTSS-II users manual*
RCA Systems Programming Information Systems Division
Radio Corporation of America June 1968
- 4 J T O'NEIL
Meta PI compiler/compiler
Private Communications
- 5 J C MILLER C M WINE
A simple display for characters & graphics
IEEE Transactions on Computers Vol C-17 No 5 May 1968

Software compatibility: What was promised, what we have, what we need

by JOHN A. GOSDEN

The MITRE Corporation
Bailey's Crossroads, Virginia

INTRODUCTION

Software compatibility is an extremely complex and pervasive topic. Unfortunately it is not well defined nor well documented. To say anything useful within the space and time constraints we must confine ourselves to generalities and readily admit in advance that there are many exceptions.

What Is software compatibility?

Many people confuse compatibility (which is a state to be achieved) with standardization or commonality (which are techniques by which compatibility may be achieved). In default of being able to define the terminology formally, we define software compatibility by examples.

If we assume that "software" implies "not hardware," then we can implicitly define software compatibility as:

- (a) common man/system interfaces—e. g., the ability to move a user, programmer, or operator from one type of computer to another without further training;
- (b) program exchange—e. g., the ability to move a program from one type of computer to another without any changes;
- (c) data exchange—e. g., the ability to send data from one program to another with only automated changes;
- (d) program pooling—e. g., the ability to assemble, combine together, various sub-programs from diverse sources into one program;
- (e) data pooling—e. g., the ability to combine diverse source files into an integrated data base.

Varieties of compatibility

In default of formal definitions, it is useful to attempt to describe a few of the properties of compatibility.

Kinds of compatibility

There are three usefully distinct kinds of compatibility:

- (a) *Identical*—i.e., one thing is exactly the same as another, or two things can interface directly;
- (b) *Convertible*—i.e., a simple set of algorithms exist for two-way translation. If we translate "X" to "Y" and back to "X" then the resulting "X" is identical to the original "X";
- (c) *Translatable*—i.e., some complex process is needed for translating the output of one thing to the input of another.

Degree of (how much) compatibility

There are only two pragmatically useful degrees of compatibility:

- (a) complete, or algorithmic—can be fully automated
- (b) close, or nearly algorithmic—can be automated except for a few exceptions.

A complete simulator and a COBOL-to-machine-code compiler are examples of (a).

EXODUS¹ and LIBERATOR² are examples of (b); they translate most of a program from one computer to another and flag some exceptions that must be hand-coded.

Extent of compatibility

The extent of compatibility can be characterized in two ways:

- (a) *Entire*—i.e., everything in “X” is compatible with “Y”;
- (b) *Limited*—i.e., a set of limiting but acceptable conventions exists such that anything in “X” conforming to the conventions is compatible with “Y.”

A simple example is that for a program at installation “X” to be transferable to “Y” it must usually satisfy two criteria:

- (1) conform to a POL standard—both “X” and “Y” accept an identical programming language;
- (2) conform to a set of conventions, such as limits on storage space and peripherals needed.

We call this limited compatibility from “X” to “Y.”

Directions of compatibility

Direction of compatibility can be characterized in three ways (assume “Y” represents a greater capability than “X”):

- (a) *Upward*—i.e., there is “entire” compatibility from “X” to “Y”, but “Y” to “X” is unspecified.
- (b) *Downward*—i.e., there is “entire” compatibility from “X” to “Y” and “limited” compatibility from “Y” to “X”;
- (c) *Complete*—i.e., there is “entire” compatibility from “X” to “Y” and “Y” to “X.”

Compatible zones

All the previous characteristics were pair-wise considerations of compatibility. A more useful concept is a compatibility “zone.” A compatibility zone is a collection of EDP centers all of whom are downward compatible with some notional “zone model.” The zone model is a set of conventions such that anything—be it a program, a file, or a language—that conforms to the zone conventions is as a result, automatically entirely compatible with all other members of the zone.

Now we can readily see that any EDP center is a potential member of many zones, e.g.,

- (a) one computer at the center
- (b) two different makes of computers at the center
- (c) three of the same make of computer at different centers
- (d) all computers in one corporation
- (e) all computers in SHARE
- (f) all computers in Air Force Logistics Command.

In general, the more extensive the zone the more restrictive the conventions, and for each thing developed, a tradeoff decision must be made between the extent of compatibility desired and the consequent restrictive conventions to be followed.

What were we promised?

In this paper we are restricting ourselves to promises made by those who thereby commit themselves to deliver—we are excluding prophets who only make predictions. Searches of the literature showed that few of these promises are recorded in print, and the best evidence for them lies in the lack of explicit disclaimers. Overall we were not promised a great deal and the promises were largely implicit rather than explicit. Let us number the promises Pi.

Common man/system interfaces

With reference to common man/system interfaces, we were promised that, using POL’s, we would:

- P1—not have to learn a new language when we moved to a new computer;
- P2—be able to use the same language on many computers with no relearning;
- P3—have upward compatible subsets of each language that would be available on different sized computers.

Now these promises were not all-embracing, and not all were meant to cover each POL on all computers; the general concept was clear, but the scope was vague. P1 and P2 were made in the late 50’s and P3 during formal standardization during the early 60’s after COBOL “electives” caused many problems.

Note that we were not promised interfaces for:

- (a) operating systems (control input or error messages)
- (b) diagnostics (variety available or format)

- (c) utilities (facilities available or control input)
- (d) on-line dialogs (semantics or syntax).

Program exchange

With regard to program exchange, a general search of the literature shows three interesting things. First, except for a recent study by the author that also discusses commonality,³ nearly all the literature cites standardization and conventions as the basic ways to obtain compatibility. Second, most authors, notable A. Holt⁴ in 1960, SHARE⁵ in 1961, and more recently Morenoff and McLean⁶ in 1966 have explicitly concentrated on program exchange as being the essence of software compatibility—although Morenoff and McLean also mention data exchange as well. These people have looked to computer independent languages as a solution. Third, the developers and proponents of common programming languages, notably Bromberg⁷ and Heising,⁸ have noted that the prime objective is to simplify programming and that they only offer a first step to compatibility. Sammet⁹ also notes that the goal is program compatibility, not automatic transfer, and certainly not data tapes.

Overall we were apparently promised a great deal:

- P4—that a program written in a POL on one type of machine could easily be run on a different type by using a common source language program and the appropriate compiler on each computer type.

We are excluding the promise of running the same object program on several computers of the same family because that is hardware compatibility.

There is no doubt that many people thought such a promise existed and their disappointments are recorded many times. As an example, see Gordon.¹⁰ However, the promises one can find in print are a little vague: a statement by Rosen¹¹ in 1964 of a requirement that the first Philco 2000's have a compiler that would accept 704 FORTRAN substantially without change; a loose statement by Luebbert and Collom¹² in 1959 that compilers will be available to automatically translate a program "for any computer desired"; a mention by Shaw¹³ in 1961 that suggests the only cost of mov-

ing programs coded in JOVIAL from one computer to another are those of writing a new compiler; an expectation by Schwalb¹⁴ in 1963 that GECOM programs could, with no change, be compiled on future GE computers. On the other hand, RCA¹⁵ in 1960 clearly stated that COBOL only put them into a better position to cooperate in achieving compatibility.

One major problem about program exchanges was that although the caveats and conditional clauses were explicitly stated^{7,8,16} as early as 1960 and again in 1964, they were not remembered. The concept of conventions for zones of compatibility was not developed and emphasized. In 1961, SHARE did make one attempt,⁵ but Grems¹⁷ and Sanborn¹⁸ immediately criticized it as grossly inadequate, and the false notion of a kind of "universal" zone became a myth of our time.

Data exchange

With reference to data exchange, we were promised that we could:

- P5—send a file from one program to another program of the same kind (typically COBOL) and use a common data description for both within the same environment.

We were not promised any equivalent ability for data between programs in the same language on different computers let alone for data between programs produced in different languages, although it was stated to be feasible and desirable by Mahoney¹⁹ of GAO in 1964.

Program pooling

With reference to program pooling, we were promised that we could:

- P6—combine various subprograms written in a common POL or in assembly language.

We were not promised that we could mix routines written in different POL's.

Data pooling

With reference to data pooling, we were promised that we could:

- P7—merge, amalgamate, and subset files within one POL family.

We were not promised that we could do this with arbitrary files.

Summary of promises

The interesting thread through all the promises is that they tended to promise only "intra-software-family" compatibility, where a software family is, for example, "all programs written in, and the files described in, COBOL."

What do we have?

Common man/system interfaces

It does seem that the basic promises concerning common man/system interfaces have been largely fulfilled.

P1—We do have widespread compilers for COBOL and FORTRAN, and others coming along slowly, but at least improving every day.

P2—We do have substantial compatibility; various COBOL's generally do look alike.

P3—We are getting reasonable subsets.

Program exchange

P4—We are only just beginning to be able to exchange programs—there is a long way to go. Even today papers such as those by Morenoff and McLean⁶ are still suggesting conventions for zones of compatibility. Only a few groups such as Westinghouse^{20,21} have established such zones by careful planning and management.

Data exchange

P5—In some cases, we can exchange files within a software family on similar systems; unfortunately, physical files are not operating-system-independent, and this is a major problem. Even so, in cases such as Westinghouse²¹ the exchange is based upon prior agreement of formats. In other cases, at present it is necessary to build a converter for each file, and no general purpose converter exists for any pair of software families.

Program pooling

P6—We only have a very limited amount of

program pooling, being restricted usually to one software family, with the major exception being an "escape" to assembly language. We have some partial solutions in combining tasks into a job, but these are severely limited by the lack of compatibility in data exchange.

Data pooling

P7—We are able to merge, amalgamate, and subset files within a software family. To construct a data base from diverse files it is necessary to construct individual file converters, but most data base systems are limited in the structures that they can accept.

What do we need?

To put it simply, we need all the five items listed earlier:

- (a) common man/system interfaces
- (b) program exchange
- (c) data exchange
- (d) program pooling
- (e) data pooling

Most of the important ones are missing because we did not anticipate the need. What we were promised was largely

"intra-software-family compatibility"

and we are moving towards it with reasonable speed. What we were not promised, but now see that we need, is

"inter-software-family compatibility."

This divides into four problem areas:

- (a) language compatibility—the general ability to mix statements or procedures written in different programming languages (e.g., COBOL statements in a data management system query);
- (b) data exchange—the sending of files from a program of one family to one of another family (e.g., a FORTRAN program using a COBOL file);
- (c) operating system compatibility—similar and compatible services expected by and provided to each software family and to each human family—operators, programmers, and users;

- (d) data pooling—the ability to build an integrated data base from diverse sources.

Language compatibility

If we want to be able to mix routines written in different languages we have two possible solutions. We need either:

- (a) one comprehensive language which covers all needs so that mixtures are not necessary, or
- (b) a harmonious (compatible) family of languages.

Alternative (a) is unattractive. Some steps have been taken in that direction. PL/I is an amalgam of the scope of COBOL and FORTRAN, but we agree with Barton²² that languages will proliferate. Proliferation will occur in two dimensions at least:

- (1) languages for special groups of users;
- (2) languages for different levels of sophistication of users.

To try to blend all these is likely to lead to a "kludge." All these points were recognized by SHARE²³ in 1958 when they published the UNCOL concept, which they noted had "been around" since 1954. This provides a mechanism upon which to build an unlimited number of compatible languages.

Data exchange*

There is a growing need for data exchange, particularly the passing of files of data between programs that were produced independently. This will be needed in the development of computer networks and data bases; for example, a head office installation collecting files from various divisions of a corporation to build a data base. Both the development of formal and informal computer networks as well as the economic feasibility of large data bases are favoring the development of arrangements for a considerable volume of data exchange, whether directly over communication systems or by the dispatch of reels of tape and boxes of cards. These are very significant areas of growth that are just beginning to emerge in commercial EDP and are already creating problems within the Federal government.

The development of data interchange is straightforward when the correspondents have agreed on the format, but where there has been no prior agreement, conversion usually involves considerable manual intervention. Some typical problems are that:

- (a) The sender's format may not be specified rigorously and an informal description may have to be debugged.
- (b) The sender's format may not be expressible in the receiver's system.
- (c) The sender's format descriptions may be embedded in the program.
- (d) The format in the sender's system may vary from record to record and be embedded in the data.

Any of these problems may arise when either an existing application is converted to a new system, or a new member of a cooperating network has a system different from that of any existing member.

There are two basic problems:

- (a) Few existing systems have any ability to deal with a new format automatically, and those that do are limited to data described in the same system.
- (b) The number of different, and often incompatible, ways of describing data is increasing; e.g., Format statements in FORTRAN, Data Description in COBOL, COMPOOL in JOVIAL, FFT's in FFS.

Any solution to this problem should not restrict participants in the use, within their own local system, of any internal data structure they like or any programming or query language they like. Therefore we need a standard data description language for data exchange. It is expected that systems should interface with a single way of describing data for interchange and provide conversion processes in their interfaces. If a suitable interface is to be developed, we will not want to standardize formats, which would be absurd, but we would want to standardize ways to describe formats. We also will want to attach the data descriptions to the data, so that the transmission of both data and its description can be performed without manual intervention.

A data description language for data interchange does not principally have to be read and

*The text of this section is largely drawn from reference 24.

understood by humans. It can be thought of as a complicated coding to be generated and interpreted by the interface modules of systems in a network. In a well-designed system a user would describe data in the form provided for local use, and the system would translate to data interchange conventions. Therefore, the data description language should be generally compatible with data descriptions in current programming languages. Later, developments in programming languages may be influenced by a desire to remain compatible with data interchange conventions.

Operating system compatibility

Operating systems differ because they interface closely with those parts of hardware that vary a great deal, and because there is still a great variety among operating system design philosophies. As a result, compatibility among operating systems provided by different suppliers is minimal. There are some rare exceptions: at North American Aviation, the UNIVAC 1107 operating system was modified to provide control card compatibility with the IBM 7090; and currently, Lockheed is attempting to develop a sub-executive to provide a common input-output interface between COBOL programs and future operating systems.

The solution is to develop a standard operating system specification. We need to concentrate on the external standards that have a primary effect upon the compatibility goals; they include:

- (a) System Control Cards
- (b) Message Formats
- (c) Calling Sequences
- (d) Security Procedures
- (e) Label Conventions.

However, the future for operating system standardization does not look optimistic and USASI standards are not likely to be agreed upon and implemented before the next generation of computer systems. It was only as recently as 1967 that Halstead²⁵ noted the need for operating-system-independent programs. The only current activity is that M. Perstein of SDC has recently begun to try to determine if there is sufficient interest in operating system standards to form a USASI working group. On the other hand, there has been some progress in a few of the basic interfaces (e.g., the adopted, but not yet widely ac-

cepted, ASCII code;²⁶ and the drafted, but not yet approved, USASI standard for labels.²⁷

The ideal long-range (fourth generation?) solution is to have a standard machine-level interface provided to the outside world by an "extended computer." An "extended computer" is the hardware plus the operating system and any microprograms or read-only storage, and is one level of the extensible machine concept of Goodroe and Leonard.²⁸ This could be the long awaited "UNCOL,"²³ or the standard machine language advocated by Barton.²²

Data pooling

We are right in the middle of a series of developments that are addressed to the concept of data pools. These are the data management systems. Unfortunately they have given little attention as yet to the problem of amalgamating varieties of existing data files except by special converters built for each specific file.

There are two possible solutions to this problem:

- (a) build a set of converters for each data management system—each data management system needs two for every other system with which it wants to interface;
- (b) use a common data description language for data exchange.

We favor the second alternative. The next need is to ensure that each data management system can handle all possible data structures. A recent CODASYL report²⁹ addressed this topic in a straightforward manner, and Mealy³⁰ points out the need for later binding and more explicit representation of data description in programs. A data description language would provide a means to send a self-describing file to another installation, the file might include code tables, format descriptions and pointers. It is the "internal" specification discussed by Moores.³¹ His external format is for man/machine use but we believe more than one standard will be needed for data description by men, just as many programming languages will be needed.

Summary of needs

What we need is a radical change in software architecture to handle all these problems. One

obvious solution is separate explicit specifications of procedures and data:

- (a) a standard "extended computer" interface (UNCOL);
- (b) a standard data description language for data exchange.

REFERENCES AND BIBLIOGRAPHY

- 1 M E CONWAY
Proposal for an UNCOL
CACM 1 10 p 5 October 1958
- 2 F J CORBATO et al
The compatible time-sharing system
MIT Press Cambridge Mass 1963
- 3 J A GOSDEN et al
Achieving inter-ADP center compatibility
The MITRE Corporation MTP-312 May 1968
- 4 A HOLT
Overall computational control and labelling
CACM 3 11 p 614 November 1966
- 5 G F RYCKMAN
Operational compatibility of systems—Conventions
CACM 4 6 p 266 June 1963
- 6 E MORENOFF J B McLEAN
An approach to standardizing computer systems
Proc ACM 22nd Nat Conf p 527 1967
- 7 H BROMBERG
What COBOL isn't
Datamation 7 9 p 27 September 1961
- 8 W P HEISING
FORTRAN: Compatibility and standardization
Datamation 10 8 p 24 August 1964
- 9 J E SAMMETT
More comments on COBOL
Datamation 7 3 p 33 March 1961
- 10 R M GORDON
COBOL and compatibility
Datamation 10 7 p 47 July 1963
- 11 S ROSEN
Programming systems and languages—An historical survey
AFIPS Conf Proc 25 SJCC 1964 p 1 Spartan Books
- 12 W F LUEBBERT P W COLLOM JR
Signal Corp research and development on automatic programming of digital computers
CACM 2 2 p 22 February 1959
- 13 C J SHAW
JOVAL
Datamation 7 6 p 28 June 1961
- 14 J SCHWALB
Compiling in English
Datamation 9 7 p 28 July 1963
- 15 RCA Letter to Editor
Datamation 6 4 p 35 July/August 1961
- 16 C A PHILLIPS
Letter to Editor
Datamation 6 3 p 24 May/June 1960
- 17 M GREMS
Letter—Standards conventions
CACM 4 9 p 365 September 1961
- 18 T G SANBORN
Letter—Standards conventions
CACM 4 9 p 366 September 1961
- 19 E J MAHONEY
Federal EDP
Datamation 10 1 p 26 January 1964
- 20 Westinghouse Electric Corp
COBOL Programming TIPS
Management Systems Department April 1967
- 21 W B FRITZ
Computer change at Westinghouse Defense and Space Center
AFIPS Conf Proc 31 FJCC 1967 p 581 Thompson Books
- 22 R S BARTON
A critical review of the state of the programming art
AFIPS Conf Proc 23 SJCC 1963 p 169 Spartan Books
- 23 J STRONG et al
The problem of programming communication with changing machines
CACM 1 8 p 12 August 1958
- 24 J GOSDEN
Suggested standards for data description for use in data interchange
Prepared for USASI X3.4 and reproduced in Appendix D of reference 3 June 1967
- 25 M H HALSTEAD
Machine independence and third generation computers
AFIPS Conf Proc 31 FJCC 1967 p 587 Thompson Books
- 26 USASI
Standard code for information interchange
USAST X3.4 1967.
- 27 USASI Proposed USA Standard
Magnetic tape labels for information exchange
CACM 10 11 p 737 November 1967
- 28 J R GOODROE G F LEONARD
An environment for an operating system
Proc ACM 17th Nat Conf 1964
- 29 Report to the CODASYL COBOL Committee
COBOL extensions to handle data bases
January 1968 Reprinted in SIGPLAN Notices 3 4 April 1968
- 30 G H MEALY
Another look at data
AFIPS Conf Proc 31 FJCC 1967 p 525 Thompson Books
- 31 J L LITTLE C N MOORES
Standards for user procedures and data formats in automated information systems and networks
AFIPS Conf Proc 32 SJCC 1968 p 89 Thompson Books
- 32 H BROMBERG
COBOL and compatibility
Datamation 7 2 p 30 February 1961
- 33 J BROOKS
Letter to Bureau of the Budget Director, Schultz
CACM 11 1 p 55 January 1968
- 34 A LIPPITT
COBOL and compatibility
CACM 5 5 p 254 May 1962

Interactive systems—Promises, present and future

by JULES I. SCHWARTZ

System Development Corporation
Santa Monica, California

INTRODUCTION

In recent years, "interactive" systems have become synonymous with "time-shared" systems for most people. Time-sharing has been emphasized by those interested in providing interactive (on-line) access to a computer. On the other hand, there are a number of other kinds of systems that provide interactive service. First, there are systems like SAGE¹⁰ and the airline reservation systems (Ref.²⁶, for example). These are single-purpose systems providing the capability for users to communicate directly with a computer to accomplish a well-defined task. Then there are the numerous third-generation operating systems, generally considered to be multiprogrammed systems. There is an increasing tendency for these systems to provide an interactive capability to some set of users.

These multiprogrammed systems provide a number of interesting examples of interactive application programs. But generally, they limit the truly interactive use to one user or one application at a time; that application receives response satisfactory for interactive activity, but little opportunity exists for others to do the same. These systems also tend to limit the core storage available to a given user, so that operation on small configurations is impractical. Also, the number of simultaneous programs is limited by core memory on most multiprogrammed systems. Time-sharing systems, on the other hand, attempt to provide a certain "aura" of user orientation which usually doesn't exist on typical multiprogrammed systems. Time-sharing systems are designed to serve the on-line user (although off-line service is also provided by most time-shared systems today). Con-

sequently, there is an attempt to orient the services and communications to the person at the console, making them concise, forgiving, and easy-to-learn, while multiprogrammed systems tend to base their interactive access on the off-line methods normally associated with batch-processing systems.

There are, of course, exceptions to the generalizations mentioned above. The Michigan Time-Sharing System (MTS),³⁶ which was derived from a standard multiprogrammed system, now services 30 or more simultaneous users in what appears to be a normal interactive fashion. Other systems (e.g., Allen-Babcock¹), have adapted OS/360 in such a way that the interactive user—although limited in capability—apparently has the interactive characteristics provided in more complete time-sharing systems, while the basic OS/360 system services non-interactive programs.

Essentially, the major difference between time-sharing and other multiprogrammed systems lies in the method of scheduling. The concept of the "time-slice" is characteristic of most time-sharing systems, whereas the other operating systems rely on priorities to provide rapid service to a subset of users. Also, most time-sharing systems rely on "swapping" to provide access to many programs of up to full physical core size. (Incidentally, the systems mentioned in the preceding paragraph are not swapping systems.)

In summary, time-sharing systems concentrate on the interactive user, and have been in the forefront in providing this service. Consequently, this discussion will concentrate mostly on the development of time-sharing systems per se, while not overlooking the fact that interesting work on in-

teractive activities has gone on in non-time-shared systems. In fact, future interactive computing services may be provided by either.

Promises

The early ideas and hopes

It is always difficult to recount the history of an idea. In this case, some of the first ideas are recorded. In 1959, Strachey hypothesized a time-sharing computer and operating system.³⁴ In 1962, McCarthy foresaw: "We can envisage computing service companies whose subscribers are connected to them by telephone lines. Each subscriber needs to pay only for the capacity that he actually uses, but he has access to all programming languages characteristic of a very large system."²¹

In one of the first papers on an operational time-sharing system, Corbato et al.⁵ stated that a time-sharing system of the kind described in their paper would "improve the ability to program by one or two orders of magnitude" and "several new forms of computer usage" would be opened up. This paper also discussed some of the problems associated with building such a system, including file handling, debugging techniques, mingling of foreground and background, and scheduling. Although the paper was generally optimistic (based on some limited experience), it ended—interestingly enough—with a warning: "it is essential for future system design and implementation that all aspects of time-sharing system problems be explored and understood in prototype form on present computers so that major advances in computer organization and usage can be made."

Without doubt, one of the first and most influential papers on the subject was presented by Licklider in 1960.¹⁸ His basic theme was that in solving problems, men do some things well, such as developing hypotheses, while computers are much better at the lengthy computational tasks. By forming a partnership—a symbiotic relationship—they can help each other and thus speed the solution of problems. He stated, "It seems reasonable to envision, for a time 10 or 15 years hence, a 'thinking center' that will incorporate the functions of present day libraries together with anticipated advances in information storage and retrieval and the symbiotic functions suggested earlier in this paper. The picture readily enlarges itself into a network of such centers, connected to

one another by wide-band communication lines and to individual users by leased-wire services. In such a system, the speed of the computers would be balanced, and the cost of the gigantic memories and the sophisticated programs would be divided by the number of users." In sum, he discussed the equipment and programming concepts which he thought would assist the human in his contact with the computer.

In addition to these papers, numerous other statements have been made on the future of time-sharing. To mention but a few: "All computing will be on-line by 1975," "Ninety percent of all computing will be done on-line by 1970," and "checkout will be 100 or more times faster on-line"—all made possible by time-sharing.

The first systems

In 1963, the ideas of some of the early papers began to be demonstrated. At MIT,⁸ SDC^{30,31} and BBN,³ time-sharing systems that were more than demonstration vehicles came into existence, and communities of users began to form. The number of simultaneous users permitted on systems such as these (using second-generation hardware) ranged from 5 to about 30, although some predicted that a time-sharing system could serve 100 users on the IBM 7090.³

The next steps

It seemed easy to extrapolate to hundreds of users once the hardware had been built specifically for time-sharing. Consequently, preparation for third-generation time-sharing machines began in earnest in 1964. Dennis⁸ described a scheme whereby names—representing "pages" rather than arbitrary blocks—would be associated with locations through associative hardware. Pages would be part of a larger memory hierarchy called "segments," sufficiently large so that expanding dynamic requirements for data space would not require continual reallocation of space. Implementation of these hardware concepts was begun on several major computers (GE 645 and IBM 360/67). In theory, they would ease the problems of storage management for large-scale time-sharing systems by providing automatic access only to those portions of data space that were required, and by providing considerably more flexibility in the mapping of larger random-access storage onto core storage.

By 1965, these hardware systems were part of rather ambitious plans to provide the next major steps in large-scale general-purpose time-sharing at MIT and IBM. MIT embarked on the development of the MULTICS System.⁷ This system promised—by 1966 or 1967—to be the first version of a true “computer utility,” which would serve several hundred to a thousand simultaneous users solving major problems on the GE 645 (Ref.⁹). IBM followed soon after with plans for a similarly ambitious effort on the IBM 360/67 (TSS). In addition to the addressing hardware described earlier, both of these machines were the first time-sharing computers that would contain dual processors—this is, essentially balanced systems where more than one control and computing element would have equal access to the total machine. This dual capability was to provide more power to time-sharing systems, which until then were severely limited in computing power.

Special-purpose systems

One other important area in time-sharing history should be discussed at this point. While the initial efforts at the development and use of large-scale general-purpose time-sharing systems were under way at places like MIT and SDC, efforts in the development of special-purpose systems tailored to a problem-oriented user environment were being pursued (frequently on relatively small computers). One of the first was JOSS,³³ which was a system devoted exclusively to the solution of computational problems by people unsophisticated in computer usage. Another was BASIC,¹⁵ an extremely simple but reasonably efficient system for handling small computational problems. These systems promised to be an effective tool within a prescribed problem area for a number of simultaneous users.

The present

Dr. Licklider, who has been a great stimulus in the field of time-sharing, recently stated informally that he was surprised and pleased by the number of current time-sharing systems, but was disappointed at the lack of coherence in them. It is interesting to note that in the past few years the number of systems labeled “time-sharing” has increased from around five experimental systems to about 30 different commercial systems operating in 70 installations (as of early 1968), and prob-

ably several hundred experimental or research systems.³⁷ This is particularly surprising since most computer manufacturers—who traditionally lead the field in operating system development—have tended either to ignore, or treat as subordinate, the development of time-sharing. Several attributes of these systems can be isolated, but the general picture is rather incoherent. Relatively few standards apply, and an examination of these systems shows that the definition of time-sharing is by no means simple. (See Ref.³² for further discussion.)

Commercial time-sharing

Most commercial services now are effectively one-language systems: GE’s BASIC, Allen-Babcock’s version of PL/I, BBN’s TELCOMP (JOSS), and IBM’s QUIKTRAN (FORTRAN) are examples of such systems. Other systems stress one kind of application—for example, KEY-DATA for business applications, and the IBM Text Data Service.

It also appears that few *large-scale* general-purpose systems have made the grade yet on a commercial basis. One system that has at least some general-purpose attributes is the SDS 940 Time-Sharing System, which began as an experimental vehicle at the University of California at Berkeley, but is now in commercial use at a number of installations. Other than the fact that the computer is rather small and file capacity is limited on this system, the 940 Time-Sharing System shows promise that a multi-purpose system can be commercially successful. The GE BASIC System,¹⁵ which started as a university and one-language system, is now also beginning to offer multi-language facilities commercially.

Laboratory systems

There are still a large number of systems in the laboratory. Systems like MULTICS and IBM’s TSS, which were presumed by many to represent the next major step in large-scale interactive computing, still haven’t fulfilled their promise. At this writing, MULTICS is just beginning to work in a minimal way, and the IBM system is being used in scattered installations with considerably fewer simultaneous users and services than one might have envisioned for this time. The reasons for the relatively slow progress of these systems are many. For one, they require large

programming system efforts on relatively new hardware. These kinds of efforts are always much slower in production than smaller efforts. Also, they are attempts at order-of-magnitude improvement in quantity and quality of service. Whether these hardware and software designs will result in dramatically improved performance is, of course, a subject of debate at this time. Theoretically it seems possible, but perhaps the previously noted admonition by Corbato et al.⁵ should be re-examined in assessing these systems. An attempt at assessing the effects of hardware and software decisions on efforts such as these was made by Nielson.²³ One result of his study was the (not unexpected) finding that there are a considerable number of parameters affecting the performance in complex systems like these, and thorough simulations of them are in themselves quite difficult, but should be a requirement.

Other reasonably general systems are now operating on third-generation computers at the University of Michigan, Lincoln Laboratory, Livermore Laboratory, SDC, and the California Institute of Technology. These systems can handle about 30 or more simultaneous users on single-processor configurations.

In sum, then, we have not yet achieved orders of magnitude improvements in service over that offered in the early days of time-sharing. The original systems at SDC and MIT are still running, and still approximate the current limit on capability of large-scale general-purpose time-sharing systems. Many of the special-purpose commercial systems, however, now serve between 40 and 100 simultaneous users on reasonably small computers. Thus, it seems quite possible that populations of many hundreds could share a large computer where the application was limited.

Applications of interactive systems

So far we have discussed the progress of time-sharing in terms of raw power (i.e., amount of service provided). It now remains to discuss the ways in which interactive systems have been used in recent years. Even though the *amount* of service provided is important (particularly commercially), a more interesting aspect of these systems concerns the *kind* of service provided. Under time-sharing, certain kinds of processing can be accomplished which would not be economically possible on systems that do not take advantage of the

lag in human response time to service other users' computation requirements. In the last few years, a wide-ranging set of interesting applications have been exercised on general-purpose time-shared systems. These include game-playing, on-line simulations, computer-assisted instruction, retrieval of information from data banks, and numerous others which take advantage of keyboard devices. Experimentation with devices other than keyboards on some general-purpose systems has resulted in a number of interesting applications, including examination of three-dimensional figures through rotation on a display, graphical data analysis and plotting, real-time recognition of handwritten characters and mathematical expressions, and hypothesis-testing utilizing a display to present the alternatives and relative weights of factors.

Some operating systems have been devoted to specialized interactive applications. For example, in the APEX System²⁰ at Lincoln Laboratory, the structure, language, and data forms of the system are oriented toward the construction of a variety of graphical applications. Applications on this system include aids to drafting, electronic component design, analysis of three-dimensional figures and the hidden line problem, and the analysis of electronic circuits and program flow diagrams. For another example, Jacks²⁴ has reported on an interesting application of an interactive system to automated design. Several simultaneous users were able to utilize displays in the design and study of automobile components.

Earlier in this paper we quoted Licklider's view that by 1975 networks of interactive computers would be available to service users. How far have we progressed in this area? Marril and Roberts¹⁹ have written a report on the current state of networking. By actual count, there have been few real applications of the networking concept in conjunction with interactive computing. Experimental two node networks have been formed between various installations. Three examples include SDC's Q-32 TSS and SRI's CDC 160A; the Q-32 and Lincoln Laboratory's TX-2; and the TX-2 and a PDP-9 in Washington. The concept generally demonstrated was the use of a small, remote, on-line device-driver utilizing a distant large computer for computing power—a concept also used in other installations, but not under the "network" label. The TX-2 to Q-32 link did go fur-

ther, however. In this case, programs running on one computer were able to invoke and run programs on the other, thus permitting quite incompatible computers to share capabilities (for example, a LISP compiler on the Q-32 was used by a display program running on the TX-2).

Thus far, slow progress has been made in sharing information among a network of interactive computers, which would provide a massive library at a user's fingertips. Many reasons exist for this slow progress in the information-network field. The handling of information itself (even on one computer) is a subject of considerable technological complexity. Also, experiments in networking are expensive, requiring several or more computers and high communication costs. Few agencies are willing to investigate an expensive concept until it is, in fact, proven economical.

Evaluations

Like most new concepts, interactively-oriented systems have generated their share of controversy. One area of debate lies in the pure "economics" of running such systems. In order to sustain a large number of simultaneous user programs with rapid response times, one must be willing to pay some overhead in swap-time and program management (and perhaps in equipment) when operating under time-sharing; such overhead is not required in more sequentially oriented systems. The amount of overhead is a subject of some question and, of course, varies widely on different systems. Some early quantitative studies of this subject were made by Scherr²⁹ and Fine and McIsaac.^{11,22}

For several reasons, it is hard to draw firm conclusions from studies such as these. For one thing, there is little basis on which to compare the "efficiency" of a particular time-sharing system with a non-interactive system. There are few quantitative studies of time-sharing, but there are probably fewer (relatively) for other kinds of systems. Comparing raw percentages of the time in object program state on a time-sharing system (such as the 60-70 percent demonstrated for the Q-32 TSS) versus the efficiency of a particular batch-processing system (such as OS/360) doesn't really give much insight into the problem.

Furthermore, it is argued by some that the gain in productivity using a computer interactively is so great that a considerable loss in "computer

efficiency" can be tolerated. This theory has been generally accepted, but since the early days of time-sharing the feeling that interaction with a computer greatly increased productivity has been primarily intuitive. There have been some attempts at testing this hypothesis over the last few years. For the most part, these attempts have consisted of experiments comparing program production and problem-solving under time-shared and non-interactive systems.

A discussion of five such studies was presented by Sackman.²⁸ Attempting to summarize here what is in effect a summary of a controversial set of experiments is obviously precarious, but a few of the conclusions given by Sackman might be mentioned. Using interactive systems, fewer man-hours but more computer time is utilized in solution of problems. (The differences demonstrated in these experiments are much smaller than the orders of magnitude predicted in the early days.) User preference for interactive over conventional off-line operation was evident. Also, in many cases individual subject differences in the experiments overshadowed any differences in the modes of computer operation. Sackman also came to other conclusions, and made a plea for improvements in methodology in several areas which would make future experiments more meaningful. It should be mentioned that there are some who disagree with this kind of experiment, or at least the experiments that have been performed so far. This criticism has included—among others—accusations that inadequate or obtuse sets of statistical methods were used; that the experimental design was poor; that unrepresentative systems were used in the experiments (what is a representative time-sharing system?); and that since time-sharing is relatively new, most systems used had to be experimental. A particularly vitriolic critique of an experiment was made by Lampson.¹⁷ Patrick²⁷ was slightly—very slightly—more reserved in his critique of some of this work. (Interestingly enough, Patrick and Lampson differ strongly over the value of time-sharing.)

O'Sullivan²⁴ has made a number of interesting observations on existing time-sharing systems. In his work (primarily computational), his installation used a fairly large set of different commercial systems. Among other things, he pointed out that none was perfect, some had unique advantages, and the variety of pricing schemes as well as range of capabilities warranted the use of a com-

puter to automate access to the various systems.

Thus an assessment of the current state of time-sharing and interactive systems is difficult to summarize. Such systems appear to be proliferating. A large number of enthusiastic users exist, as well as a wide variety of systems. Access to a commercial system for on-line computing requires only a decision as to which one to use. Many interesting applications are now being supported. Some early predictions and hopes for the future have not yet been attained. Some people are trying to produce experimental evidence of the gains (or losses) due to this technology. Others are seemingly satisfied with the intuitive evidence (whichever it happens to be for them).

The future

On what basis can one predict the future of interactive systems? In some sense, one should be able simply to extrapolate from current systems and make assumptions on improvements in computer power that will provide a parallel improvement in capability. But of course such predictions are risky. For example, some new third-generation developments (which were based on prototype large-scale time-sharing systems on second-generation computers) have not yet reached the power of the second-generation efforts, let alone surpassed it. This is partly due to the fact that designers are generally not satisfied just to improve the capability of an old system. They always want to add considerably more, even revolutionary, capabilities to the system. In computing, these ambitions can delay, if not do away with, visible progress. Consequently, simple extrapolations from old systems don't work.

On the other hand, a total lack of confidence isn't in order, either. One can comfortably predict that within the next five years, at least some special-purpose time-sharing systems will be capable of handling a thousand simultaneous users. Whether or not one computer's handling of many hundreds of users makes commercial sense depends to a large extent on the communications industry. Current rate structures make communications the most expensive part of conversational computing for all but close centers. There are efforts to change this situation, but the obstacles are great.

Perhaps as large-scale general-purpose systems on third-generation machines get shaken down

over the next several years, we shall see their capacities rise to the one or two hundred user class. But until the advent of fourth-generation computers, or the economical and efficient utilization of three or more multi-processors, general-purpose systems will not have the user capacity of special-purpose systems.

Multi- or parallel-processing will certainly tend to facilitate increased capability in time-shared systems. Systems now exist on some computers, such as the Burroughs B5500 and CDC 6600, which take advantage of multiple processors. These are not now strictly interactive systems, but obviously demonstrate the usefulness of truly simultaneous processing capability. The two major efforts at using multi-processor computers (IBM 360/67 and GE-645) for large-scale time-sharing have not yet progressed to the point where this value for interactive computing is demonstrable.

Access devices

Most interactive system terminals today use typewriter-like devices. Their prime advantages are their relatively low price and the use of a hard-copy medium, which provides automatically a record of all input and output. Cathode-ray tube devices avoid some of the problems of typewriters. They can operate rapidly, and are considerably more flexible in format and editing control. CRT's are gradually becoming more widely used as terminal devices, and over the next few years should be used as much as typewriter devices. Obstacles to their acceptance into the field have included high cost per terminal and communication limitations, which make the rapid data-rate necessary to update and maintain distant displays prohibitively expensive or impossible. Prices are coming down, slowly, and the recent influx of reasonably inexpensive keyboard-CRT alphanumeric displays has accelerated the trend away from paper output devices. Thus systems of the future should be largely CRT terminal oriented. This should tend to further stimulate interest in interactive systems, since the use of CRT's consistently attracts more attention than applications oriented to other devices.

Other techniques are being investigated which will facilitate new methods of dialogue with computers in the future. These include direct use of touch-tone telephones; hand-written input via devices such as the RAND Tablet or Grafacon;

voice input and output through a set of software and hardware constructions. Both of the latter are far from becoming generally useful. Interesting demonstrations of on-line hand-written input have been given,^{2,13} but the mass use of this technique currently would be prohibitively expensive as well as somewhat limited in capability. With a few minor exceptions, use of voice-recognition or output equipment with computers is strictly a laboratory exercise at this time.

Networks

It is unlikely that sufficiently powerful computers will be developed within the next decade to handle the large volume and variety of information and applications required by populations of interactive users. It is also questionable whether use of a single, monolithic computer is the best technical approach. A network of computers, each with an independent set of information files and applications, may represent the ultimate potential for the interactive user.

Experience in this area is limited. Experimentation is expensive, and again, remote communication is relatively costly and is limited by present facilities. The programming problems are also significant. Procedures for minimizing communication paths and optimizing loads on computers must be developed. The decision to maintain centralized or decentralized directories of data and programs is one not yet solved in the general case. Standards for data formats and query languages must evolve. These and other problems will have to be solved before networks of computers can become a reality.

Considering the difficulties and expense involved in evaluating networks as a viable technique for providing the ultimate capability for populations of users, it is clear that a major sponsor or set of sponsors who can fund and coordinate these activities is necessary. One such effort is now beginning. The Information Processing Technique Office of ARPA, which was largely responsible for the development of time-sharing over the last five years, has begun the first large-scale network effort in this country. In 1969, they expect to have a nation-wide network of about 20 interconnected computers operating in a preliminary fashion (most of these will have interactive operating systems running on dissimilar hardware). At each installation, a satellite computer will serve

as the buffer and language translator between the home computer and adjacent nodes of the network. Efforts such as these are necessary to develop the technology needed to put multi-computer power at people's fingertips.

Improving communication and understanding

The major emphasis of most interactive systems to date has been in the area of individual problem-solving. It may be, however, that the ultimate value of these systems will lie in their ability to bring together and assist in assimilating diverse ideas from large collections of users. Licklider and Taylor have discussed these ideas at some length.³⁵ They have described a meeting at Stanford Research Institute, at which speakers and attendees had direct access to a shared computer. With this mechanism, speakers and listeners were able to refer quickly to significant background and demonstration material, so that the discussions could proceed at a reasonable pace while permitting all participants to stay involved. Extrapolating from this point, they envisioned the eventual formation of "user communities" communicating and mutually developing ideas over a widespread network of computers.

Interactively oriented systems as operating systems

An examination of the characteristics of large-scale general-purpose time-sharing systems raises an interesting question: Could they be used as the basic operating system in a variety of installations, instead of the more common form of batch-processing or multiprogrammed systems?

Time-sharing systems have a range of response characteristics; they are always "up"; they have been shown to be evolutionary, and at least in truly general-purpose systems, they don't exclude any kind of application. Several manufacturers have recognized the basic capabilities of time-sharing and have promoted it as their operating systems. Digital Equipment Corporation provides an example of this with their PDP-6 system. Scientific Data Systems also emphasizes time-sharing on some of their line. At SDC, a system called ADEPT,¹⁶ is designed for this purpose on third-generation hardware.

If time-shared systems really provide the capabilities described here, one might legitimately ask why they have not yet come to predominate third-

generation computers. The major third-generation operating systems, although generally multiprogrammed and with some capability for on-line interaction, are primarily oriented to the off-line, or nonterminal user. One reason, mentioned before, is reluctance on the part of most manufacturers to make time-sharing the major operating system. To some extent, this is due to the necessity for manufacturers to cater to the tradition existing in most computing installations. Computer customers have been used to operating in a non-interactive environment. The manufacturers believe their customers would not accept willingly a rapid change in this condition.

Another reason that general-purpose interactive systems have been slow in coming to the fore is the fairly slow emergence of the better known time-sharing efforts. It is quite possible that, had the MULTICS and IBM TSS efforts strongly demonstrated in 1967 the value of large-scale general-purpose time-sharing, the momentum would have increased in favor of time-sharing. Of course, from the days of the early systems, there have been controversies over the relative values of interactive systems and more traditional systems. The volume of demand for "hard facts" regarding interactive operation is probably unprecedented in computing history. Whether this requirement for statistics is necessary or not, it certainly serves to make a now reasonably conservative industry hesitate to take seemingly radical steps.

There is little question that on-line interaction for certain kinds of applications is desirable. On the other hand, there will always be problems for which human interaction is undesirable. However, we must avoid thinking in terms of closed systems. Some think that time-sharing systems permit only interactive use, and batch-processing systems do not allow on-line access. As stated before, this is not true. Both types of systems are providing the two kinds of service. Third-generation time-sharing systems emphasize the availability of batch-processing services in the background, and in a similar fashion third-generation batch-processing systems provide interactive capability. In fact, some of the standard operating systems are even beginning to use the word "time-sharing" to describe part of the services provided (see Ref.³). Thus, one can predict with some degree of confidence that all systems will eventually provide some interactive capability to more than one application or user class. Time-

slicing may well be part of the scheduling scheme, although it may be coupled with priority assignments. Swapping may be used in configurations where large, fast, random-access devices exist but operating storage is small, although in a few years, operating memories may be large and cheap enough so that swapping won't be necessary. The command language for interactive users will be simple enough so that on-line access will not require preparation equivalent to that needed for preparation of off-line inputs. Whether these systems will evolve from current time-sharing or batch-processing systems is not very clear, and probably not too important. What is important is that in order to be successful, the batch-processing operations will have to be efficient, and the interactive access will have to satisfy the principles learned in today's time-sharing installations.

CONCLUDING REMARKS

Today we take for granted the airline clerk's interactive use of a remote computer to make inquiries and reservations for our flights. Soon he will be able to plan meals, schedule flights on other airlines, and make hotel reservations at the same terminal. Ultimately, he will probably make use of other services to check the credit status of the customer making the plans, as well as call up displays pointing out the variety of routes and costs on planned itineraries. Using interactive systems for school management, engineering computations, business data processing, program composition, various military activities, and other purposes is becoming common. The use of these systems to assist engineers, architects, draftsmen, and others using elaborate display techniques is now in its infancy. Communication facilities, computers, terminal devices, and operating systems are being adapted to accommodate communities of interactive users. Techniques for integrating software and hardware complexes are being investigated in order to provide truly large amounts of information to many users. There is no question that the future generation will find their access to computers immediate and an essential part of their daily life, whether at home, school, business, or in the library. (Parkhill's book²⁵ considers the future possibilities and problems of the computer "utility" in much greater detail.)

Recently, an informal conversation was held discussing the possibility of starting an experi-

ment with a "computerized town." All industry, school, utility, financial, and domestic facilities would have direct access to a network of computers. This would ultimately eliminate money transactions, lengthy individual income tax calculations, uncoordinated record keeping, many problems in transportation, and an unlimited variety of other conditions. The social and psychological problems involved in installing such a system were apparent in this discussion (but not attacked). It was significant to note, however, that the *technical* problems mentioned—although not deemed trivial—were within our grasp. In no small measure, this is due to the experience gained during this decade in interactive computing.

BIBLIOGRAPHY

- 1 RUSH terminal user's manual
Allen-Babcock Computing Inc November 1966
- 2 M I BERNSTEIN T G WILLIAMS
A two-dimensional programming system
Proceedings IFIP Congress 1968 in press
- 3 S BOILEN et al
A time-sharing debugging system for a small computer
Proceedings of the Spring Joint Computer Conference 1963
pp 51-58
- 4 D CAMPBELL W COOK W HEFFNER
General comprehensive operating supervisor—GECOS III
Software Age Vol 2 No 1 January 1968 pp 8-14 38-39
- 5 F J CORBATO M M DAGGETT R C DALEY
An experimental time-sharing system
Proceedings of the Spring Joint Computer Conference 1962
pp 335-344
- 6 F J CORBATO et al
The compatible time-sharing system: A programmer's guide
Cambridge Massachusetts The MIT Press 1963
- 7 F J CORBATO V A VYSSOTSKY
Introduction and overview of the MULTICS system
Proceedings of the Fall Joint Computer Conference 1965 pp
185-196
- 8 J B DENNIS
*Segmentation and the design of multi-programmed computer
systems*
IEEE International Convention Record Part 3 1965 pp
214-225
- 9 General Electric GE 645 time-sharing system
Digital Computer Newsletter Office of Naval Research April
1966
- 10 R R EVERETT C A ZRAKET H D BENINGTON
SAGE—A data processing system for air defense
Proceedings of the Eastern Joint Computer Conference
December 1957 pp 148-155
- 11 G H FINE P V McISAAC
Simulation of a time-sharing system
Management Science Vol 12 No 6 February 1966 pp B-180-
B-194
- 12 G H FINE C W JACKSON P V McISAAC
Dynamic program behavior under paging
Proceedings of the 21st National ACM Conference 1966 pp
223-228
- 13 G F GRONER
*Real time recognition of handprinted text: Program documenta-
tion*
RAND Corporation document RM-5550-ARPA May 1968
- 14 E L JACKS
*A laboratory for the study of graphical man-machine communica-
tion*
Proceedings of the Fall Joint Computer Conference 1964 pp
343-350
- 15 J G KEMENY T E JURTZ
BASIC programming
New York John Wiley & Sons Inc 1967
- 16 P R KENNEDY
*The ADEPT-50 system: a general description of the time-sharing
executive and the programmer's package*
SDC document TM-3899/100/01 June 7 1968
- 17 B W LAMPSON
*A critique of an exploratory investigation of programmer per-
formance under on-line and off-line conditions*
IEEE Transactions on Human Factors in Electronics Vol
HFE-8 No 1 March 1967 pp 48-51
- 18 J C R LICKLIDER
Man-computer symbiosis,
IRE Transactions on Human Factors in Electronics Vol
HFE-1 No 1 March 1960 pp 4-11
- 19 T MARRIL L G ROBERTS
Toward a cooperative network of time-shared computers
Proceedings of the Fall Joint Computer Conference 1966 pp
425-431
- 20 Graphics Semiannual technical summary report 1 June-30
November 1967
Massachusetts Institute of Technology Lincoln Laboratory
ESD-TR-67-570 January 1968
- 21 J McCARATHY
Time-sharing computer systems
In Greenberger M ed Management and the Computer of the
Future Cambridge Massachusetts The MIT Press 1962 pp
221-236
- 22 P V McISAAC
*Job descriptions and scheduling in the SDC Q-32 time-sharing
sharing system*
SDC document TM-2996/000/00 June 1966
- 23 N R NIELSEN
The analysis of general-purpose computer time-sharing systems
Stanford University document No 40-10-1 December 1966
- 24 T C O'SULLIVAN
Exploiting the time-sharing environment
Proceedings of the 22nd ACM Conference 1967 pp 169-175
- 25 D F PARKHILL
The challenge of the computer utility
Massachusetts Addison-Wesley 1966
- 26 R W PARKER
The SABRE system
Datamation Vol 11 No 9 September 1965 pp 49-52
- 27 R L PATRICK
Time-sharing tally sheet
Datamation Vol 13 No 11 November 1967 pp 42-47
- 28 H SACKMAN
Time-sharing versus batch processing The experimental evidence
Proceedings of the Spring Joint Computer Conference 1968
pp 1-10
- 29 A L SCHERR
An analysis of time-shared computer systems
Massachusetts Institute of Technology document MAC-TR-

- 18 June 1965
- 30 J I SCHWARTZ C WEISSMAN
The SDC time-sharing system revisited
Proceedings of the 22nd National ACM Conference 1967
pp 263-271
- 31 J I SCHWARTZ C WEISSMAN E G COFFMAN
A general-purpose time-sharing system
Proceedings of the Spring Joint Computer Conference 1964
pp 397-411
- 32 J I SCHWARTZ
Observations on time-shared systems
Proceedings of the 20th National ACM Conference 1965 pp
525-542
- 33 J C SHAW
*JOSS: A designer's view of an experimental on-line computing
system*
Proceedings of the Fall Joint Computer Conference 1964 pp
455-464
- 34 C STRACHEY
Time-sharing in large fast computers
Proceedings of the International Conference on Information
Processing UNESCO June 1959 Paper B 2 19
- 35 R W TAYLOR J C R LICKLIDER
The computer as a communication device
Science and Technology No 76 April 1968 pp 21-31
- 36 University of Michigan terminal system (MTS)
University of Michigan 2 Vols December 1967
- 37 D C WHITNEY C H WHITE JR
Time-sharing services
Modern Data Systems Vol 1 No 1 February 1968 pp 40-50

Multiprogramming—Promise, performance and prospect

by THOMAS B. STEEL, JR.

System Development Corporation
Santa Monica, California

INTRODUCTION

“Multiprogramming” is the label given to the concept of a dynamic sharing of the resources of a given computer system among two or more programs. An operating multiprogramming system presents to external observers the appearance of effecting the concurrent execution of several object programs. There may or may not be truly simultaneous operation of more than one program, but it will be the case that a second program begins execution before the first program has run to completion. Simple sharing of storage among several programs in a systematic way to facilitate serial execution is insufficient to qualify an operating system as incorporating multiprogramming. There must be an *oscillation* of control among the several programs for multiprogramming to come into play.

In order to make a rational analysis of the relation between the claims for multiprogramming and the actual performance of multiprogramming systems, it is essential to have a clear understanding of the motivation for attempts to develop such systems. It is unexceptionable that the primary justification for multiprogramming systems is the economic advantage that accrues from their use. Many areas on the frontier of information processing science, such as artificial intelligence and mechanical translation, draw their research and development energies from the desire to augment the functional capabilities of computing systems; others, such as programming language development, are pursued in the hope of improving the ability of humans to cope with the computer. Multiprogramming is intended to improve the performance of a machine *qua* machine. A multiprogramming system provides no capability that could not be obtained in its absence *at a price*. It must be kept in mind, however, that there are certain everyday activities that, while they could be accomplished in the absence of multiprogramming, could not be afforded with contemporary

hardware. A case in point is giving every graduate student his own computer.

To explicate the essentially economic nature of multiprogramming it is necessary to examine the usual justifications for its use.¹ Three arguments are advanced: (1) improved throughput by maximizing utilization of machine components, (2) time-sharing, with all it implies, and (3) real time response.

Improved throughput is currently the most prevalent reason for employing multiprogramming and is likely to remain so for the foreseeable future, time-sharing advocates notwithstanding. In this context, multiprogramming requires some multiprocessing capability in the hardware. If it is not possible for two or more explicitly programmable functions to occur in parallel, nothing can be gained from multiprogramming in the way of improved throughput and the attendant overhead is merely expensive waste motion. For this reason, multiprogramming did not arrive on the computing scene until the advent of elementary multiprocessing.²

If coupled systems and satellite computers are disregarded as irrelevant in the present context,* the only pertinent parallelism in generally used computers has been, and continues to be, overlap of computing and input-output. Typically, the computing time required to service an input or output device is far less than the time required for that device to perform its function. Thus, unless another part of the program initiating the input-output operation, or another program altogether, can employ the computing circuits of the machine while the input or output request is being processed, time is lost and throughput is diminished. Most multiprogramming systems are designed explicitly to overcome this loss. Improved throughput means

*The satellites and the drivers of coupled systems should be regarded as external, asynchronous signal sources and their relationship to multiprogramming is, therefore, to be found in the real time context.

more computing per unit time which means more computing per dollar. Here the motivation for multiprogramming is palpably economic.

The relationship between multiprogramming and the technology of on line, multi-access systems, unfortunately almost universally mislabelled "time-sharing," must be carefully delineated in order to absolve multiprogramming from unmerited responsibility for the many difficulties that have plagued on-line systems. An on-line system involves multiprogramming in a special way only in the event that it is also a multi-access system. The purpose of this multiprogramming by rapid cycling through many users is to provide an economic impedance match between slow men and a fast computer. If, as may some day happen, machines were inexpensive enough it would be reasonable to provide every user with his own computer, a tour de force eliminating the need for "time-sharing." Admittedly, this observation ignores certain significant elements of multi-access systems such as mutually interacting users and shared files, but it does illustrate the primacy of the economic motivation for multiprogramming in this context.

Real time systems pose something of a problem for this discussion as it is not clearly appropriate to view, say, an airline reservation system in the performance of its principal duty as a multiprogrammed system. Terminal servicing of this sort under control of a master program is really more akin to the treatment of conventional programs in machines with arithmetic overflow and zero divide interrupts. There is a hint of multiprogramming in the situation but it is extremely primitive. The full employment of multiprogramming in real time systems occurs when it is found desirable to occupy machine resources that would otherwise be idle while waiting for an external signal. Here the background task filling the idle time could be performed on another computer at additional cost. Again, the motivation is primarily economic, although dynamic access to a changing data base by the background program may be a factor.

Having evoked maximization of the cost effectiveness of computer resources as the dominant excuse for multiprogramming systems, it follows that instances of these systems must be judged primarily in economic terms. Care must be taken, however, to examine all aspects of the cost equation, for capturing idle machine time at the expense of programmer and operator frustration may well be a poor trade. It will be seen below that this is a nonempty caveat.

Promise

The first explicit mention of multiprogramming in the general literature was, as noted above, in the con-

text of overlapping computing with input-output.² Here, as elsewhere initially, there was no hint of a multiprogramming *system*, merely the suggestion of employing the technique in a single program. These early claims were modest and generally remained so in the responsible literature. As might be expected, performance claims got a little out of hand in the sales brochures of equipment manufacturers, a recurrent theme that will generally be disregarded in this account.

In the case of certain early military applications of computer systems for command and control, such as SAGE,³ it is at least arguable, in hindsight, that the rudiments of what would now be called a multiprogramming system were in evidence. Since for reasons of military security the details of such systems were not generally available, their development exerted little direct influence on the future course of multiprogramming. The indirect impact of these military activities was considerable, however, as a technical expertise unavailable to the less affluent civilian sector was acquired by the personnel engaged in the development of these big, complex and advanced systems. With the passage of time this expertise percolated to all corners of the industry. It is not within the scope of this paper to discuss the hopes and realities of these military systems.

Probably the first, and certainly the most ambitious, early attempt to create a multiprogramming capability in an operating system context where the object programs were expected to be totally independent was the SHARE 709 System (SOS).⁴ SOS was designed for the IBM 709, a machine with a multiple channel, asynchronous input-output capability. Among the design objectives was the ability to perform, *in parallel*, input for job $N + 1$, compute for job N , and output for job $N - 1$. The conceptualization and design of this capability to the level of detailed flow diagrams was complete by the Fall of 1957 and the prospect of such facilities was widely accepted soon thereafter.

The next major step in the advocacy of multiprogramming systems was the explicit recognition by Strachey of the hierarchical nature of immediacy on computer time demands.⁵ The key idea in this work is not, as is usually claimed, the invention of the multi-access concept; it is the introduction of the "director," a program element now usually referred to as a "scheduler." In the example detailed by Strachey there was only a single on-line user envisioned, but given the concept of *an* on-line user with priority over all off-line users, and the director, only a modicum of imagination is required to arrive at the rest of the multi-access, on-line system concept. Subtract the on-line

user, however, and what remains is a reasonable prescription for a modern batch multiprogramming system. The real advance in this approach over the ideas inherent in the SOS structure was freeing the system from the constraint that object programs must pass through the system in a lock step order that is dictated by the initial input stream sequence.

By the end of 1961 a number, far more than cited here, of intuitive claims and theoretical analyses of multiprogramming behavior had appeared.⁶⁻⁸ The consensus of these studies was that effective multiprogramming was feasible, valuable and imminent. The meaning of "effective" varied, of course, from author to author. Two measures of multiprogramming effectiveness have generally been given: (1) the amount of time devoted to the overhead activity of keeping the system operating, and (2) the improvement of throughput over that encountered in a strict batch system. Both are normally quoted as percentages, and neither is really satisfactory as a measure of multiprogramming effectivity.

The first measure indicates the minimum distance from perfection—the unrealizable state where all of the machine is busy on useful work all of the time—that the system could attain with an ideal job mix. It gives an absolute measure of the best case. The second measure indicates the relative improvement in performance but provides no indication of how this relates to the ideal. Even taken together the two measures do not close the gap. Something else is needed, perhaps a set of standard job mixes, but this paper must stand on the available data.

These early claims varied but generally suggested overhead figures of between one and fifteen percent and throughput improvements of up to several hundred percent. The low figures for overhead were quoted by manufacturers, such as Honeywell for its Model 800, who planned to employ hardware for much of the overhead activity. The extra hardware costs money, however, and this cost is normally not factored into the equation.

By 1962, and continuing to date, the scientific literature began to carry reports of actual experience with multiprogramming systems.⁹⁻¹⁴ Most of these reports were made by representatives of hardware manufacturers and, therefore, exhibited a natural tendency to minimize difficulties. Since this time, the performance claims for multiprogramming systems have remained more or less static in the responsible literature. Two revealing changes have occurred: (1) the word "multiprogramming" no longer contains a hyphen, providing at least linguistic legitimacy to the concept, and (2) the availability dates for most multiprogramming

systems have slipped, often considerably, but happily at somewhat less than real time.

At the present time most large, commercially available computing systems are designed with the expectation that they will be multiprogrammed in many environments. Generally they are accompanied by a manufacturer supplied, batch multiprogramming system. Some are vehicles for on-line, multi-access systems that are now in being. Since these things exist and can be evaluated by the customer, current claims tend to be rather realistic and representative of the actual situation, at least with respect to performance. Ease of use is another matter altogether.

Performance

Having reviewed in deliberately general terms the promises for multiprogramming, it is now necessary to review in equally general terms what has actually been accomplished in providing multiprogramming capability. As noted above, the early claims were modest and made in the context of multiprogramming within the confines of a single object program. Even here the claims went a bit beyond reality. While substantial gains in program performance were obtained through overlapping computing with input-output operations, the results were less gratifying than it seemed they ought to be. Two reasons for this can be isolated.

In the first place, insufficient information was provided by the hardware for programmers to make completely effective use of the actual capabilities of the machine. The extent of the requisite interaction between hardware and program for efficient handling of asynchronous interrupts was long unrecognized, despite some careful studies on the subject.⁵ Indeed, this subject is not perfectly understood today, but at least there seems no doubt that, at long last, machine designers are aware of the problem. Attention to its resolution cannot fail to improve the performance of future systems.

The second difficulty with early multiprogramming efforts was more subtle and went unappreciated until attempts were made to instrument computing systems to determine, among other things, just how much overlap of computing and input-output actually went on. The depressing nature of the results could be traced to the fact that the average programmer simply was not good enough to make full use of the multi-channel hardware capability, even where the problem cried out for its use. It was the recognition of this situation that paved the way for the design of the first multiprogramming *systems*.

SOS was a failure if measured by its impact on the community. It is the author's contention—exhibiting

a bias justified in one of the designers—that the failure was in implementation, not in design. To put the matter bluntly, IBM blew it! The system that finally came out, to be distributed and maintained by the vendor, was a pale ghost of the original conception (in many aspects, not just in the multiprogramming features) that eventually evolved into IBSYS, a tolerably good batch processing system, but hardly a multiprogramming system. The fact that reasonable approximations of the original were used by the RAND Corporation, the Applied Physics Laboratory and others indicts the IBM effort. The multiprogramming aspect of this situation must be carefully weighed. It was possible to get approximately the same amount of overlap between computing and input-output in both IBSYS and SOS, but it required much more programmer effort in IBSYS. This difference is significant in view of the difficulty programmers have with this problem.

The multiprogramming aspects of large, real time systems and on-line, multi-access systems are somewhat more severe than those found in batch multiprogramming systems, due to the added pressure of vital time requirements on the scheduler not usually found in batch situations. Nevertheless, the literature reveals that the serious problems in these more complex systems really lie elsewhere in such areas as command language, paging, multi-access files and lack of re-entrancy in generated code.¹⁶⁻¹⁷ It is a slight oversimplification, but not amiss in principle, to assert that the current performance of *multiprogramming* in real time and multi-access systems has about the same relationship to the earlier promises as can be shown to apply in batch systems.

The literature now abounds in reports on the performance of various multiprocessing systems,¹⁸⁻²³ but with some notable inadequacies; e.g., for OS/360. The trouble with these reports is that they don't say very much concrete. The lack of carefully defined measures hurts the analyst and the lack of hard data hurts him even more. In view of this situation the author has forborne from constructing a table of performances of the specific existing systems. It would be invidious to attempt an explicit comparison from the available data, and no individual has had sufficient experience with several systems to permit the gathering of even intuitive judgments. What evidence there is suggests that all of the major systems share roughly the same strengths and weaknesses. Thus, it makes sense to compare a kind of generic, current multiprogramming system against the industry's broad claims for what it, collectively, expected to produce.

Viewed in these terms, the situation is rather better than one might expect. While it is only in special situations that throughput improvement has attained the

several hundred percent initially claimed, the average figures of between thirty and seventy percent are but a binary order of magnitude below the claims. More or less the same factor of two appears in the overhead figures; ten to thirty percent in the real situation as against half that in the projections. The serious problem has been in delivery dates, a not uncommon element of software development. As has been observed, much of this is due to inadequacies in the hardware. Good multiprogramming systems simply were not in the cards until the current class of machines were available.

The glaring failure of current multiprogramming technology is the complications it has introduced for the programmer and operator. The current Job Control Languages (JCL) required to specify what the system is to do are, by and large, disasters. It takes far too much of a programmer's time to construct the appropriate JCL statements, and an even larger amount of time to debug them, not to mention the effect on morale of aborted runs deriving from trivial JCL errors. The implications of JCL in the machine room are even worse. The operators must be a good deal smarter (and, therefore, necessarily, paid more) than has been standard in the past. Computer center managers are faced with the realization that poor operations will destroy and overwhelm any gain in throughput with no trouble at all.

Prospect

In considering where multiprogramming may be expected to go from here, it is worth noting the reason why this technology fares considerably better than most when measured against the early claims of its proponents. In multiprogramming there is a well defined, finite upper limit for improvement in capability. The most one could ever claim was to use all of the machine all of the time. This is hardly true in other cases; witness artificial intelligence where the more flamboyant claims defy the imagination. As multiprogramming technology approaches its natural limit, it clearly ceases to make sense to expend much effort trying for the last fraction. Satisfaction is obtained by near optimum performance at acceptable cost.

Some performance improvement can be obtained through judicious use of the various empirical²⁴ and theoretical²⁵⁻²⁷ studies of multiprogramming, and through attention to the problem of hardware-software interaction. It is likely that use of a firmware approach to multiprogramming supervisors by placing them in read-fast, write-slow storage units will provide the last big reduction in overhead cost. Improvements beyond a factor of two or three will probably cost more than they are worth. Of course, the introduc-

tion of multiple, interacting arithmetic, logic and control units may start a whole new ball game.

The real improvements in multiprogramming systems must come in the area of making them easier to program and to operate. As indicated above, much is left to be done in this area, and it shows little sign of happening so far. Programmers sweat, operators err and managers complain, but JCL marches on.

REFERENCES

- 1 E F CODD
Multiprogramming
Advances in Computers 3 Academic Press New York 1962 p 77
- 2 N ROCHESTER
The computer and its peripheral equipment
Proc EJCC p 64 1955
- 3 R R EVERETT C A ZRAKET H D BENNINGTON
SAGE—A data processing system for air defense
Proc EJCC p 148 1957
- 4 O MOCK C J SWIFT
The SHARE 709 System: Programmed input-output buffering
JACM 6 2 p 145 1959
- 5 C STRACHEY
Time sharing in large fast computers
Proc ICIP p 336 1959
- 6 E F CODD
Multiprogram scheduling
CACM 3 6 p 347 1960
- 7 B L RYLE
Multiple program data processing
CACM 4 2 p 99 1961
- 8 T KILBURN D J HOWARTH R B PAYNE
F H SUMNER
The Manchester University Atlas operating system Part I:
Internal Organization
Computer J 4 p 222 1961
- 9 R D SMITH
Multiprogramming the RCA 601
Preprints of 16th ACM National Meeting p 12C-1 1961
- 10 G FELTON
Multiprogramming on the Ferranti Orion computer system
Proc IFIP p 570 1962
- 11 R CLIPPINGER
Multi-programming on the Honeywell 800/1800
Proc IFIP p 571 1962
- 12 J L FRANCOIS
Programmation simultanee dans le Cas du Gamma 60 Bull
Proc IFIP p 573 1962
- 13 E F CODD
Multi-programming stretch: A report on trials
Proc IFIP p 574 1962
- 14 A J CRITCHLOW
Generalized multiprocessing and multiprogramming systems
Proc FJCC p 107 1963
- 15 F P BROOKS JR
A program-controlled program interruption system
Proc EJCC p 128 1957
- 16 J SCHWARTZ
A general purpose time-sharing system
Proc SJCC p 397 1964
- 17 F J CORBATÓ V A VYSSOTSKY
Introduction and overview of the multics system
Proc FJCC p 185 1965
- 18 B B CLAYTON E K DORFF R E FAGEN
An operating system and programming system for the 6600
Proc SJCC v 2 p 41 1964
- 19 H S BRIGHT
A Philco multiprocessing system
Proc SJCC v 2 p 97 1964
- 20 D M DAHM
Using the B5500 disk file master control program
Proc IFIP v 2 p 362 1965
- 21 J W LEWIS
Multiprogramming on Leo III
Proc IFIP v 2 p 363 1965
- 22 J BOUVARD
Experience with multiprogramming on the Honeywell 800/1800
Proc IFIP v 2 p 364 1965
- 23 R E FAGEN
Time-sharing the Control Data 6600
Proc IFIP v 2 p 368 1965
- 24 H N CANTRELL A L ELLISON
Multiprogramming system performance measurement and analysis
Proc SJCC p 213 1968
- 25 W CHANG D J WONG
Analysis of real time multiprogramming
JACM 12 4 p 581 1965
- 26 B PETERS
Security considerations in a multi-programmed computer system
Proc SJCC p 283 1967
- 27 D P GAVER JR
Probability models for multiprogramming computer systems
JACM 14 3 p 423 1967

An algorithm for finding a solution of simultaneous nonlinear equations

by R. H. HARDAWAY

Collins Radio Company
Dallas, Texas

INTRODUCTION

In many practical problems the need for a solution of a set of simultaneous nonlinear algebraic equations arises. The problems will vary greatly from one discipline to another, but the basic mathematical formulation remains the same. A general digital computer solution for all sets of simultaneous nonlinear equations does not seem to exist at the present time; however, several recent techniques make the solution of certain systems more feasible than in the past.

The algorithm described here is a slight variation of one of the methods described by C. G. Broyden¹ in "A Class of Methods for Solving Nonlinear Simultaneous Equations." This modified version of Newton's method converges quadratically for a convex space. It includes Broyden's technique of approximating the initial Jacobian and then updating its inverse at each step rather than recomputing and reinverting the Jacobian at each iteration. A procedure is given which helped to circumvent the difficulty of an initially singular Jacobian in several test cases.

The examples given include applications in several engineering fields. A simple hydraulic network and the equivalent nonlinear resistive network are given to show the identical mathematical formulation. Applications to the stress analysis of a cable, to the analysis of a hydraulic network, to optimal control problems, to the determination of nonlinear stability domains and to statistical modeling are mentioned as examples of usage.

Statement of the problem

Let a system of n nonlinear equations in n unknowns be given as

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= 0 \\ f_2(x_1, x_2, \dots, x_n) &= 0 \\ &\cdot \\ &\cdot \\ f_n(x_1, x_2, \dots, x_n) &= 0 \end{aligned} \quad (1)$$

This may be represented more concisely in vector notation as

$$f(x) = 0 \quad (2)$$

where x is a column vector of independent variables and f is a column vector of functions.

A solution of the system of n equations is a vector x which satisfies each f_i in the system simultaneously.

Newton's method

In Newton's method for a one dimensional case, the iterative procedure is given by

$$x_{i+1} = x_i - f(x_i)/f'(x_i) \quad (3)$$

The need for a good initial estimate, the computation of the derivative, and failure to find multiple roots are usually cited as major disadvantages of the method. However, from a "good" initial estimate, convergence of the method has been frequently proven and has been shown to be quadratic.

For an n dimensional system we will expand this notation. The initial limitations and the advantage of quadratic convergence are maintained. For proofs on the convergence, see Henrici's *Elements of Numerical Analysis*.²

Rather than a single independent variable, we have an n dimensional vector of independent variables x . An n dimensional vector of functions f replaces the single function equation and the Jacobian replaces the derivative of the function. The Jacobian is defined as an $(n \times n)$ matrix composed of the partial derivatives of the functions with respect to the various components of the x vector. A general term of the Jacobian J may be denoted as $a_{i,j} = \partial f_i / \partial x_j$, where f_i and x_j are components of the f and x vectors, respectively. Since in equation (3) the derivative appears in the denominator we must consider the inverse of the Jacobian and evaluate it as x_i . This will be denoted as J_i^{-1} .

Therefore, for an n dimensional case, Newton's method may be rewritten as

$$x_{i+1} = x_i - J_i^{-1}f(x_i) . \tag{4}$$

The obvious difficulties of the method lie in choosing an initial vector x_0 and in computing and evaluating the inverse Jacobian. Despite the oversimplification, it will be assumed that enough knowledge of the system exists to enable one to make a "good" initial guess for x_0 . The Jacobian is considerably more difficult. With the method that was presented by Broyden not only can an initial approximation to the Jacobian be used, but also at each iteration the inverse Jacobian may be updated rather than recomputed.

Broyden's variation of Newton's method

Notation

- x_i i 'th approximation to the solution
- $f_i = f(x_i)$ set of functions evaluated at x_i
- J_i or J_i^{-1} Jacobian or its inverse evaluated at x_i
- A_i or A_i^{-1} i 'th approximation to Jacobian or its inverse evaluated at x_i
- t scalar chosen to prevent divergence
- y_i the difference in f_{i+1} and f_i
 $y_i = f_{i+1} - f_i$
- p_i the negative product of A_i^{-1} and f_i
 $p_i = -A_i^{-1}f_i$
- p_i^T the transpose of p_i

Method

It is assumed that an initial approximation of the Jacobian A_0 exists. The iterative procedure seeks to find a better approximation as it also seeks to find the solution of the system. In this process the function vector f will tend to zero and indicate the convergence of the system.

If we let $p_i = -A_i^{-1}f_i$ as given above, then equation (4) becomes

$$x_{i+1} = x_i + t_i p_i \tag{5}$$

where t_i is a scalar chosen to prevent the divergence of the iterative procedure. The value of t_i is chosen so that the Euclidean norm of f_{i+1} is less than or equal to the Euclidean norm of f_i ; hence, convergence is not ensured but divergence is prevented. A complete discussion of the backward differencing method used is given in a following paragraph.

In order to define the modification method for the inverse Jacobian, we let

$$x' = x_{i+1} = x_i + t_i p_i . \tag{6}$$

Then the vector of functions of $f(x')$ may be considered a function of a single variable t . The first derivative of f with respect to t will exist since the Jacobian has previously been assumed to exist.

It follows that

$$\frac{df}{dt} = \frac{\partial f}{\partial x'} \frac{\partial x'}{\partial t_i} = \frac{\partial f}{\partial x'} p_i = J p_i . \tag{7}$$

When we determine df/dt , we have established a necessary condition on the Jacobian.

To approximate df/dt , Broyden suggests a differencing method where each component of f may be expanded as a Taylor series about s as follows:

$$f_i = f(t_i - s) = f_{i+1} - s \frac{df}{dt} - \dots \tag{8}$$

Disregarding the higher terms, an approximate expression for df/dt is

$$\frac{df}{dt} \approx \frac{f_{i+1} - f_i}{s} = \frac{y_i}{s} . \tag{9}$$

The choice of s is such that the approximation to the derivative is as accurate as possible, but care must be taken that the rounding error introduced in the division is not significant. Since we have chosen Broyden's method of full step reduction, t_i is set equal to s and (9) becomes

$$y_i = t_i \frac{df}{dt} . \tag{10}$$

If we now combine equations (7) and (10) we have another necessary condition which the Jacobian will satisfy,

$$y_i = t_i J p_i . \tag{11}$$

Since A_i , an approximation to the initial Jacobian, exists, we are seeking a better approximation and J is replaced in equation (11) with A_{i+1} ,

$$y_i = t_i A_{i+1} p_i . \tag{12}$$

This equation gives the relationship between the change in function vector f and the change in the x vector in the direction p_i . Since we have no knowledge of chang-

es in any direction other than p_i , we will assume that there is no change in the function vector f in any direction orthogonal to p_i . Using this assumption and equation (12), A_{i+1} can be determined uniquely and is expressed as follows:

$$A_{i+1} = A_i - \frac{[y_i - t_i A_i p_i] p_i^T}{t_i p_i^T p_i} \quad (13)$$

Since we actually need the inverse of A_{i+1} , Broyden uses a modification given by Householder³ which enables us to obtain a modification formula for the inverse from the information we already have. Householder's formula is

$$(A + xy^T)^{-1} = A^{-1} - \frac{A^{-1}xy^T A^{-1}}{1 + y^T A^{-1}x} \quad (14)$$

where A and $(A + xy^T)$ are nonsingular matrices and x and y are vectors all of order n . Therefore,

$$A_{i+1}^{-1} = A_i^{-1} + \frac{[t_i p_i - A_i^{-1} y_i] p_i^T A_i^{-1}}{p_i^T A_i^{-1} y_i} \quad (15)$$

is the modification we have been seeking.

This method of updating the inverse of an initial approximate Jacobian was shown by Broyden to give a better approximation as i increases if the terms omitted in the Taylor series expansion are small. Since $s = t_i$ is always chosen to be less than or equal to 1, this condition is satisfied.

With this improved approximation of the inverse a new p_i is computed and the iteration is repeated.

As x_i approaches the solution, the convergence becomes quadratic as Henrici⁸ has shown. The function vector tends to zero and the Jacobian tends to the actual Jacobian evaluated at the solution vector. Any one of several methods can be used to determine the accuracy of a solution.

Since the computation of partial derivatives has been simplified by the approximation procedure, this method presents advantages over those which require explicit evaluation of partial derivatives.

The step size at each iteration is such that the norm is reduced rather than minimized. The time spent in evaluating the set of functions repeatedly and the storage required to save various vectors to determine the minimum, negate the advantage of norm minimization. This method combines the use of initial approximations with an iteration procedure that is computationally simple, to produce an efficient algorithm.

Computational procedure

Details on the implementation of this algorithm are

given in the next paragraphs. The general flow chart in Figure 1 summarizes the procedure.

The initial vector

The initial vector x_0 may be selected arbitrarily or from knowledge of the system, With a nonlinear resistive network, for example, the elemental values could

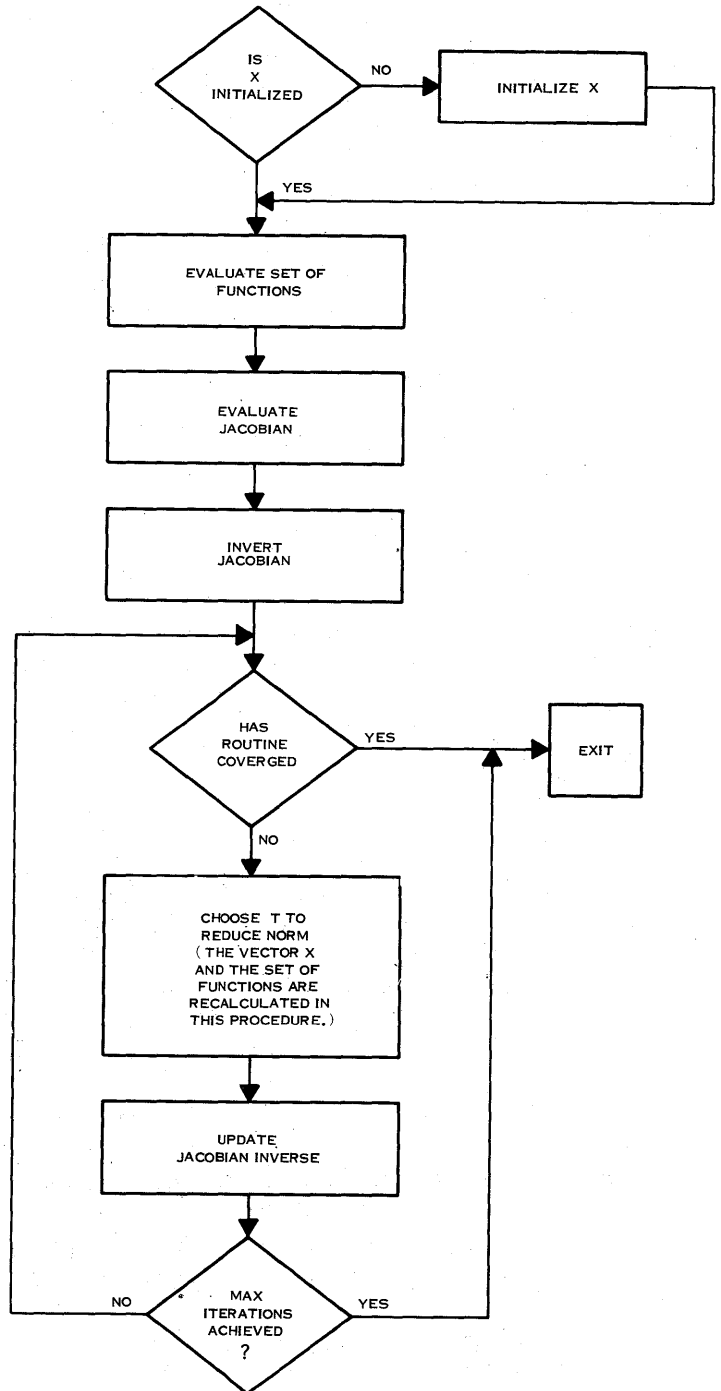


FIGURE 1—Flow chart

be used. If no knowledge at all exists, a unity vector is generated as x_0 .

The Jacobian

Since the initial Jacobian may be approximated, several methods are available to compute this matrix. The simplest is to let the inverse equal a given value and not attempt to compute or invert the Jacobian. So much valuable information is lost that, despite the simplicity, this method is not chosen.

At the opposite extreme is the possibility of computing all the necessary partial derivatives, evaluating the Jacobian at x_0 , and then inverting the matrix. The evaluation of the partial derivatives is generally very laborious so this method was not considered. Further, the assumption that A_0 may be approximated makes this degree of precision unnecessary.

The third alternative is to approximate the partial derivatives. This will provide better information than the first case and give less computational difficulty than the second case.

Since any term of A may be denoted as $a_{ij} = \partial f_j / \partial x_i$, we have

$$a_{ij} = \lim_{h \rightarrow 0} \frac{f_j(x_i + h) - f_j(x_i)}{h} \quad (16)$$

where f_j and x_i are components of f and x . For purpose of evaluating A_0 , a relatively small value of h is selected and equation (16) used to generate each element.

For the test program, a value for h was computed as .001 of x_i .

The inverse Jacobian

When the Jacobian has been evaluated, the next step is to invert it. If the Jacobian is nonsingular, the inversion is performed and the iteration proceeds. A standard Gaussian inversion routine is used.

If, however, the Jacobian is singular, the inverse does not exist. The first solution given in the previous section is one method of sidestepping this problem.

During the testing of the algorithm an attempt was made to determine an optimum arbitrary matrix. One choice was to use the results that had been stored in the inverse matrix by the Gaussian elimination procedure at the time the singularity was detected. This will give a reasonably good approximation for some entries in the inverse. At the next step, the modification procedure will improve the initial approximate inverse. This is the matrix that was used if the Jacobian was singular.

Selection of the Scalar t_i

The Euclidean norm of the vector f_i should approach zero as a solution is reached. When t_i is selected so that the norm of f_i is a nonincreasing function of i , the divergence is prevented.

The first guess for t_i is always +1. If this satisfies the condition that the Euclidean norm of $f_{i+1}(x_i + t_i p_i)$ is nonincreasing with respect to the norm of f_i , then $t_i = 1$ is used. If not, then a quadratic minimization procedure similar to one given by Broyden¹ is used. Ten attempts are made to find a good value of t_i . At that point the final value of t_i is used and the corresponding modifications are made on the inverse Jacobian and the f and x vectors. Obviously, the Euclidean norm may not be decreased but at the next step the directional change in the correction vector will result in norm reduction with relative ease.

In the process of selecting t_i , f_{i+1} , x_{i+1} and the norm of f_{i+1} are all computed. These values are all saved for future use in the computational procedure.

Convergence

The necessary degree of accuracy should be determined by the application and specified for each case. The norm of the function vector can be used as a convergence criterion; the absolute value of each element in f can be checked to see how closely it approaches zero; or a comparison between the norm of p_i and the norm of x_i may be used. This last method implies that if the norm of the vector p_i which is used to correct the solution x_i is less than some epsilon times the norm of the solution, then convergence is already attained. Since $p_i = -A_i^{-1}f_i$, an implicit test is made on f_i .

This last criterion presents an advantage because the prediction vector as well as the function vector is considered, so this was the criterion selected.

If the iterative procedure is not converging, provision must be made to terminate the problem. A value equal to $2n^2$, where n is the order of the system, is computed. The maximum of this value or 50 is used as an upper limit on the number of iterations allowed. Termination is enforced when this number of iterations has been attained whether or not a solution has been found.

Modification of inverse Jacobian

If the convergence criterion is not satisfied, then the inverse Jacobian is modified according to equation (15). The new values of x_i and f_i that are stored as x_{i+1} and f_{i+1} are placed in the appropriate vectors.

Continuing the iterative process

The value of i is set equal to $i+1$. If the maximum number of iterations will not be exceeded, the iteration is repeated from the evaluation of p_i .

The subprogram

The procedure described was programmed in generalized subroutine form using variable dimensioning and making use of the option of an external subroutine to evaluate the set of functions. The external subroutine can be varied from one application to the next without affecting the main subroutine that performs all the other invariate calculations.

As a second option, the subroutine that evaluates the initial approximation to the Jacobian is declared external. The usual approximation is given as follows,

$$a_{ij} = \frac{f_j(x_i + h) - f_j(x_i)}{h} \quad (17)$$

where $h = (.001) (x_i)$. This approximation has been programmed into a subroutine.

However, in any specific case if a better method of approximation exists, a subroutine which evaluates this approximation may be declared external and replace the first approximation subroutine. Results using the first approximation were so satisfactory that this second option was not utilized in the cases discussed here.

Timing and accuracy

The method was programmed in Fortran V on the Univac 1108 for single precision input and output. TABLE I, which follows, reflects the order of the system, the accuracy of the result, the number of iterations and the time required on the Univac 1108 for ten sample problems. Following the table the defining equations are given for each system.

Order N	Accuracy	Iterations	Time in sec.
1.	3	10 ⁻⁶	19 .02
2.	3	10 ⁻⁶	11 .01
3.	2	10 ⁻⁵	15 .01
4.	2	10 ⁻⁵	11 .01
5.	2	10 ⁻⁷	21 .23
6.	4	10 ⁻⁶	7 .02
7.	5	10 ⁻⁷	10 .02
8.	10	10 ⁻⁷	13 .09
9.	20	10 ⁻⁷	17 .40
10.	30	10 ⁻⁷	18 .90

TABLE I—Timing and accuracy of the subprogram

The defining equations, the initial vector, and the solution obtained in each test case are given as follows.

1. $f_1 = X_1^2 + X_2^2 + X_3^2 - 5$
 $f_2 = X_1 + X_2 - 1$
 $f_3 = X_1 + X_3 - 3$

$$X_o = \begin{bmatrix} -\sqrt{5} \\ 1 + \sqrt{5} \\ 3 + \sqrt{5} \end{bmatrix} \quad X = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \quad (18)$$

F. H. Deist and L. Sefor⁴.

2. same as 1.

$$X_o = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad X = \begin{bmatrix} 1.66667 \\ -.66666 \\ 1.33333 \end{bmatrix}$$

3. $f_1 = X_1^2 + X_2^2 - 1.0$
 $f_2 = 0.75 X_{13} - X_2 + 0.9$

$$X_o = \begin{bmatrix} -.4 \\ -.1 \end{bmatrix} \quad X = \begin{bmatrix} -.98170 \\ .19042 \end{bmatrix} \quad (19)$$

V. A. Matveev⁵.

4. same as 3.

$$X_o = \begin{bmatrix} 1.3 \\ -.3 \end{bmatrix} \quad X = \begin{bmatrix} .35697 \\ .93412 \end{bmatrix}$$

5. $f_1 = 10(X_2 - X_1^2)$
 $f_2 = 1 - X_1$

$$X_o = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad X = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad (20)$$

H. H. Rosenbrock⁶.

6. $f_1 = 20X_1 - \cos^2 X_2 + X_3 - \sin X_3 - 37$
 $f_2 = \cos 2X_1 + 20X_2 + \log(1 + X_4^2) + 5$
 $f_3 = \sin(X_1 + X_2) - X_2 + 19X_3 + \arctan X_3 - 12$
 $f_4 = 2 \tanh X_2 + \exp(-2X_3^2 + .5) + 21 X_4$

$$X_o = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad X = \begin{bmatrix} 1.896513 \\ -.210251 \\ .542087 \\ .023885 \end{bmatrix} \quad (21)$$

O. G. Mancino⁷.

7., 8., 9., 10. Are all defined by the same general equations

$$\begin{aligned}
 f_1 &= -(3 - .5X_1) X_1 + 2X_2 - 1 \\
 f_i &= X_{i-1} - (3 - .5X_i) X_i + 2X_{i+1} - 1 \\
 i &= 2, \dots, n - 1 \\
 f_n &= X_{n-1} - (3 - .5X_n) X_n - 1
 \end{aligned}$$

$$X_0 = \begin{bmatrix} -1 \\ \cdot \\ \cdot \\ \cdot \\ -1 \end{bmatrix} \tag{22}$$

C. G. Broyden.¹

Application of the subprogram

Equivalent systems

To demonstrate the equivalent mathematical formulation of a set of defining equations for different systems, a very simple example of a dual application will be given.

Consider the nonlinear resistive network given in Figure 2.

Let the voltage drop across each resistor be $V_i = R_i I_i^k$. Then from Kirchoff's laws the system may be completely described by the following three equations and $I_1, I_2,$ and I_3 may be determined uniquely.

$$f = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} R_1 I_1^k + R_3 I_3^k - E \\ -I_1 + I_2 + I_3 \\ -R_2 I_2^k + R_3 I_3^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{23}$$

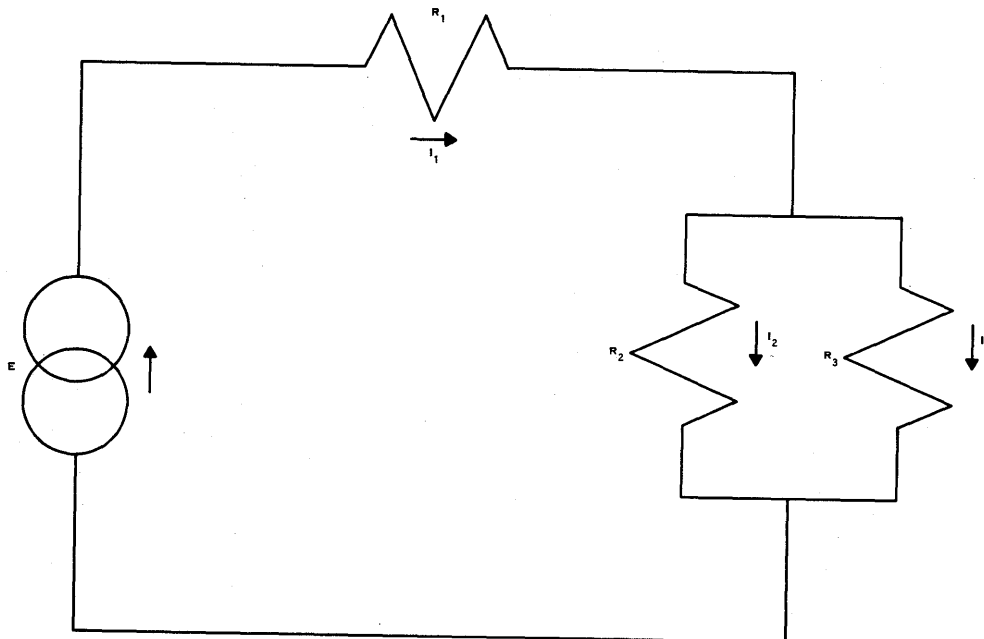


FIGURE 2—A nonlinear resistive network

As a second example, consider the hydraulic network in Figure 3.

Let the pressure drop across each member be expressed as $P_i = (\alpha L_i / D_i^{4.86}) Q_i^{1.85} = B_i Q_i^{1.85}$, where α is a constant, L_i is the length, D_i is the diameter, and Q_i is the flow. The flow through each member may be determined from the following equations which describe the system.

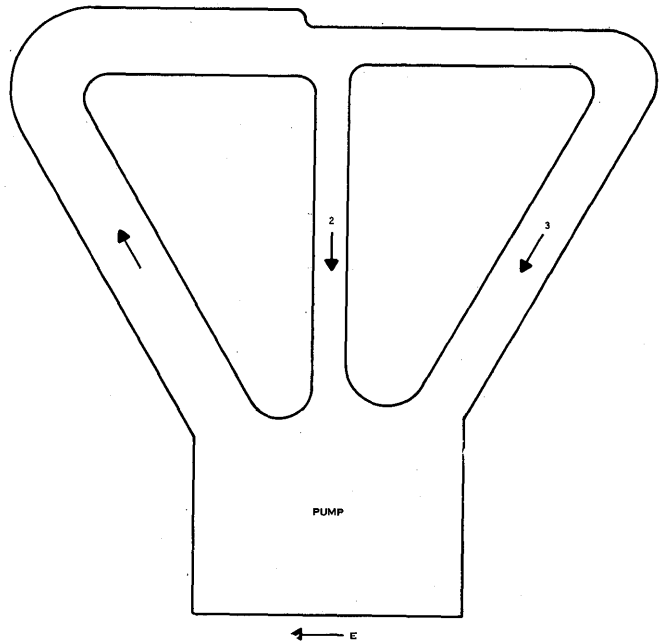


FIGURE 3—A nonlinear hydraulic network

$$f = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} B_1 Q_1^{1.85} + B_2 Q_2^{1.85} - E \\ - Q_1 + Q_2 + Q_3 \\ - B_2 Q_2^{1.85} + B_3 Q_3^{1.85} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (24)$$

Equations (23) and (24) are identical in form although they originated from different sources. The solution to either example may be determined by solving the same set of nonlinear equations.

From this second example an additional feature of the subprogram may be pointed out. The subroutine that evaluates the set of functions may be dependent on other calculations for the elements in the expressions. When the determination of a single element becomes involved, a subroutine may be used for this calculation. Finally, depending on the states of the system, control may be transferred to various segments of the subprogram and the appropriate operation performed. The versatility of a particular subprogram may be greatly increased in this manner.

Cable tension program

An application to which this method of solution was applied was the stress analysis of a suspended cable. Analysis for a uniformly loaded cable does not give an accurate picture of what actually happens under environmental conditions such as wind and ice or for concentrated loads.

The cable in Figure 3 is suspended between two points, A and B. The stress conditions may be considered forces acting on the cable at certain positions and are represented by the k weights.

By writing the static equilibrium equations for each elemental segment, a set of equations for expressing the

change in each direction as a result of each load is obtained. From these equations for each segment, the total stress on the cable in each direction may be determined.

The position of B calculated by this method will not coincide with the actual position of B. A set of nonlinear equations expresses the change in the position of B. As this change is minimized, the calculated tension on the cable approaches the actual tension.

The loads on the cable may represent any environmental conditions either singly or multiply. Analysis of the stress performed in this way gives a much more accurate solution than the assumption of a uniformly loaded cable which was previously used.

This problem is a good example of the extended capabilities that are available with the external subroutine to compute the function evaluations for the nonlinear solution subprogram. The external subroutine which we will call CNTRL acts as a monitor to determine what operations will be performed in a second subroutine, CABLE. The CABLE subroutine reads the data, computes the function evaluations and calculates the final state tension. CABLE, in turn, calls a second subroutine ELEM to evaluate the components of the equations which are evaluated in CABLE. The complexity of the equations and of the individual elements is such that this approach greatly simplifies the programming.

Applications to functional optimization

Finding a minimum of a functional of several variables is a frequently encountered problem. Many optimal control problems fall within this category; so with an appropriate set of equations for the problem this subprogram may be used to find a solution.

Both a minimization problem and an optimal control problem will probably have constraints on the solution. In a control problem both initial and terminal constraints on the state and costate variables and inequality constraints on the state and control variables may be expressed as a penalty function in the formulation of the problem. Lasdon⁸ and others have derived a method of approach involving the conjugate gradient technique which may be adopted to the given subprogram.

The determination of stability domains for nonlinear dynamical systems also involves functional optimization. As in the optimal control problem, provision must be made for the constraints that operate on the given system.

The second method of Liapunov may be used as a theoretical basis for stability determination.

A nonlinear dynamical model may be expressed as the following n -dimensional system of autonomous state differential equations,

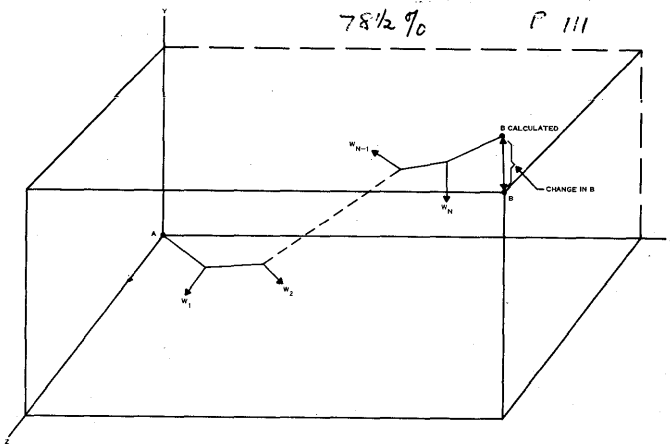


FIGURE 4—Suspended cable under stress

$$\dot{x} = Ax + f(x) = g(x), g(0) = 0 \quad (25)$$

where $f(x)$ contains all the nonlinear terms.

The method is based on choosing a quadratic Liapunov function V which yields the largest estimate of the domain of attraction for the system given in equation (25).

Figure 5 shows the projection in two dimensions of the quadratic Liapunov function.

To find the optimal or largest stability domain, one needs to maximize the area of the ellipse represented by $V(x) = C$ subject to the conditions $\dot{V}(x) \leq 0$ and $x \neq 0$. A complete discussion of this problem is given by G. R. Geiss.⁹

Hydraulics network

The analysis of a hydraulic cooling system is a specific example of how the subprogram may be used in network analysis.

The temperature drop across any heat exchanger may be calculated if we know the corresponding pressure drop. If we assume that the positions of the valves are held stationary, then from the conservation equations that describe the system, one can determine the pressure drop across each heat exchanger. With the known environmental conditions, the external temperature in the vicinity of each heat exchanger may ultimately be determined.

On the other hand, if a specific external temperature is desired, the position of each valve may be calculated

to give the proper flow through each member to cause an appropriate pressure drop. A very general program may be written to analyze a very general network. The elements and their connections may be read into the computer along with the various parameters that are necessary. The complete process of analyzing and solving the network is supervised by a program which will eventually use the subprogram for solution of simultaneous nonlinear equations.

As networks become more complex, this approach will greatly reduce the time spent in tedious calculations.

Application to statistical modeling

Many statisticians are involved in constructing models on the basis of experimental data. The model may then be used to draw conclusions and predict future outcomes. Frequently these models are nonlinear and will result in nonlinear equations. After a model has been developed, its accuracy must be constantly verified.

Using the equations of the model and the equations that describe the data the validity of the model may be determined. Since the model is assumed to be nonlinear, the resulting analysis will involve simultaneous nonlinear equations.

Other statistical applications that may require the solution of a set of simultaneous nonlinear equations are nonlinear regression analysis, testing likelihood estimator functions and survival time analysis. The many fields that use decision theory would then have the same applications.

As a simple illustration of the preceding application, suppose that it has been observed that in a sample of two hundred persons, twenty-two possess a certain genetic characteristic. Suppose, further, that the characteristic is inherited according to a hypothesis which predicts that one-eighth of those sampled can be expected to possess the characteristic. The model would be a frequency function that would enable the observer to infer future outcomes and detect disagreements with his theory. The solution of an appropriate set of nonlinear equations will express the relationship between the model and outcome. By this type of analysis the hypothesis may be accepted or rejected or reformulated.

CONCLUSIONS

The algorithm presented here can be programmed for a digital computer with relative ease. The speed of computation, the number of iterations and the accuracy of the solution compare very favorably with other methods in current use. Since the information from each preceding iteration can be used to modify the

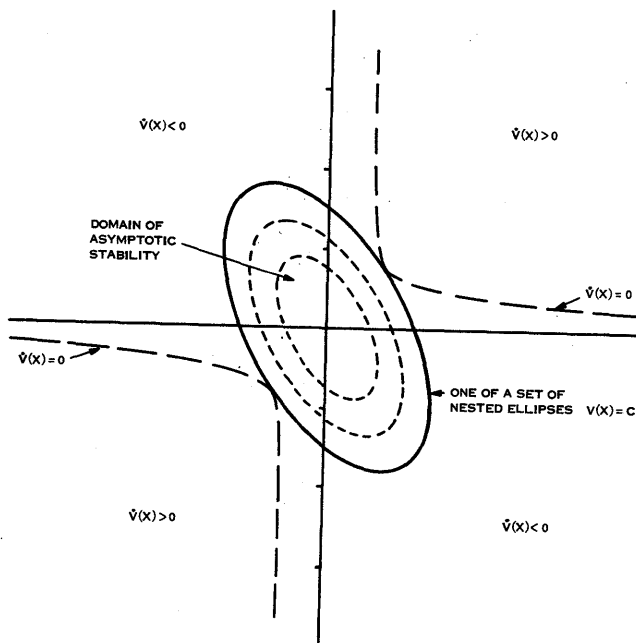


FIGURE 5—Two dimensional portrayal of region of stability

inverse Jacobian, the time involved and the complexity of operations in each iteration are minimized. In this way the inversion of the Jacobian at each iteration is avoided.

The examples presented demonstrate usages in mathematics, statistics, and several engineering fields, and these examples do not begin to exhaust the application areas. The usages are so numerous that it would seem desirable to have this type program widely available particularly in any software library designed for application purposes.

REFERENCES

- 1 C G BROYDEN
A class of methods of solving nonlinear simultaneous equations
Mathematics of Computation Vol 19 No 92 Oct 1965
- 2 P K HENRICI
Elements of Numerical Analysis
John Wiley & Sons Inc New York 1964
- 3 A S HOUSEHOLDER
Principles of Numerical Analysis
McGraw-Hill New York
1953
- 4 F H DEIST L SEFOR
Solution of systems of nonlinear equations by parameter variation
The Computer Journal May 1967
- 5 V A MATVEEV
Method of approximate solution of systems of nonlinear equations
Defense Documentation Center AD 637 966 June 1966
- 6 H H ROSENBROCK
An automatic method for finding the greatest or least value of a function
The Computer Journal Vol 3 1960 f
- 7 O G MANCINO
Resolution by iteration of some nonlinear systems
Journal of ACM Vol 14 1967
- 8 L S LASDON S J MITTER A D WAREN
The conjugate gradient method for optimal control problems
IEEE Transactions on Automatic Control Vol AC-12 April 1967
- 9 G R GEISS J V ABBATE
Study on determining stability domains for nonlinear dynamical systems
Grumman Research Department Report RE-282 Feb 1967

An economical method for calculating the discrete Fourier transform

by R. YAVNE

Raytheon Company
Bedford, Massachusetts

INTRODUCTION

With the advent of digital computers it became possible to compute the Discrete Fourier Transform for a large number of input points in relatively reasonable times. However, for certain uses a demand developed to compute the Discrete Fourier Transform in a very short time or even in real time. Also, a demand developed for computing the Fourier Transform for a very large number of input points. These demands resulted in a requirement for computing the Fourier Transform in the fastest time possible. A very economical way for computing the Fourier Transform was developed a few years ago and is known as the Cooley-Tukey Algorithm. This article describes another algorithm for computing the Discrete Fourier Transform where the required number of additions and subtractions is the same as in the Cooley-Tukey Algorithm; but the required number of multiplications is only one half of that in the Cooley-Tukey Algorithm.

The discrete Fourier transform and the Cooley-Tukey algorithm

The Discrete Fourier Transform may be expressed by the following equation:

$$F_n = \sum_{k=0}^{N-1} f_k e^{-j \frac{2\pi knT}{NT}} = \sum_{k=0}^{N-1} f_k e^{-j \frac{2\pi kn}{N}} \quad \text{for } n=0, 1, 2, \dots, N-1 \quad (1)$$

In this equation, f_k is the k th equally spaced discrete input point of the time function to be transformed. N is the number of samples entering the transformation. F_n is the n th term (the n th filter) of the Discrete Fourier Transform. The symbol T represents the time interval between input samples. Each term has a frequency bandwidth of $\frac{2\pi}{NT}$ and is centered at the fre-

quency $\frac{2\pi n}{NT}$.

The Cooley-Tukey Algorithm is a very economical way of numerically computing equation (1). (It uses a small number of additions and multiplications.) This is especially true when N is a power of 2 only. To develop this Algorithm for $N = 2^m$ we first note that:

$$e^{-j \left(\pi n + \frac{2\pi kn}{N} \right)} = (-1)^n e^{-j \frac{2\pi kn}{N}} \quad (2)$$

Therefore equation (1) can now be rewritten as:

$$F_n = \sum_{k=0}^{N-1} f_k e^{-j \frac{2\pi kn}{N}} = \sum_{k=0}^{\frac{N}{2}-1} \left[f_k + (-1)^n f_{\frac{N}{2}+k} \right] e^{-j \frac{2\pi kn}{N}} \quad \text{for } n=0, 1, \dots, N-1 \quad (3)$$

Let

$$\sigma_k = f_k + f_{\frac{N}{2}+k} \quad (4)$$

and

$$\delta_k = f_k - f_{\frac{N}{2}+k} \quad (5)$$

For n even we can write $n = 2l$ and obtain:

$$F_{2l} = \sum_{k=0}^{\frac{N}{2}-1} \sigma_k e^{-j \frac{2\pi k 2l}{N}} = \sum_{k=0}^{\frac{N}{2}-1} \sigma_k e^{-j \frac{2\pi k l}{N/2}} \quad \text{for } l = 0, 1, \dots, \frac{N}{2}-1 \quad (6)$$

Equation (6) is identical in form to equation (1) but it has half the number of input points and double the time interval between two points, since only the even numbers of the sample points were used. As a result of this, the σ 's may be treated as a new set of $\frac{N}{2}$ input points

for a Discrete Fourier Transform.

For n odd $= 2\ell + 1$ we get:

$$NF_{2\ell+1} = \sum_{k=0}^{\frac{N}{2}-1} \delta_k e^{-j \frac{2\pi k(2\ell+1)}{N}} = \sum_{k=0}^{\frac{N}{2}-1} \left(\delta_k e^{-j \frac{2\pi k}{N}} \right) e^{-j \frac{2\pi k \ell}{N/2}}$$

for $\ell = 0, 1, \dots, \frac{N}{2} - 1$ (7)

After the δ_k 's are multiplied by $e^{-j \frac{2\pi k}{N}}$ they can also be treated as a set of new input points to a Discrete Fourier Transform but, with half the number of points and double the time interval between samples. This first step which divides the input points into σ 's and δ 's and multiplies the δ 's by the appropriate $e^{-j \frac{2\pi k}{N}}$ is shown in Figure 1 in a flow chart form for eight points.

After this, the process is repeated with two groups, each containing $N/2$ input points. The next repetition will be with four groups, each containing $N/4$ input points. This continues until $N = 2^v$ groups are obtained, each containing $N/N = 1$ input point. These last input points are the desired NF_n 's of the Fourier Transform. A complete flow chart of eight input points is shown in Figure 2.

As seen in Figure 2, the output is not in the regular numerical order. The transformed order may be obtained by expressing the numbers 0, 1, 2, ... $N-1$ in a binary system, using v digits for each number, and reversing the bit order of each number. The new binary number will show the order of the output. As a result, we may say that the output is in a bit reversed order. The bit reversal for $N = 8 = 2^3$, is shown in Figure 3.

Another method for obtaining the desired output order is as follows: start with zero. Divide N by two and add this to the zero to obtain the second number. Divide $\frac{N}{2}$ by two and add this to the zero and the

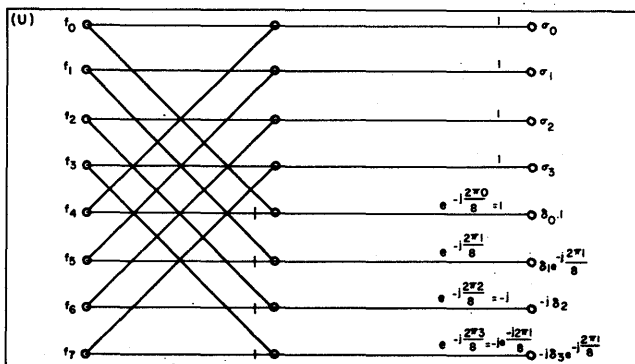


FIGURE 1—Conversion of a set of 8 input points into 2 new sets of 4 input points each

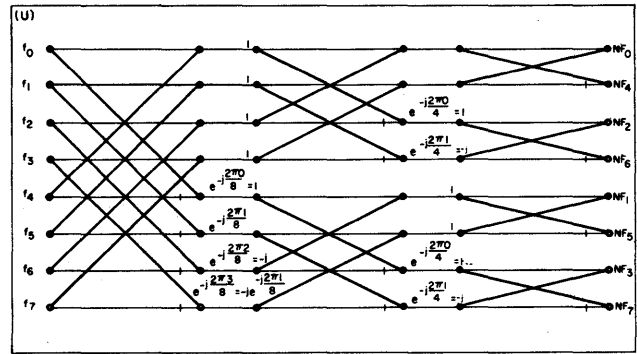


FIGURE 2—Complete flow chart for the Cooley-Tukey Fourier transform for 8 input points

DECIMAL REPRESENTATION OF NUMERICAL ORDER	BINARY REPRESENTATION OF NUMERICAL ORDER	BINARY BIT REVERSAL	DECIMAL OUTPUT
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

FIGURE 3—Bit reversal method for 8 input points

$(0 + \frac{N}{2})$ to obtain two more numbers. Divide $\frac{N}{4}$ by two and add to the four generated numbers to obtain four more numbers. Continue until N/N is obtained, which is also added to all the previously generated numbers. This completes the process.

A demonstration of this method for $N = 8$, and $N = 16$ is shown in Figure 4.

It is worth noting that in the bit reversed order, the numbers in the case N appear as every other number is the case $2N$.

There are many variations of the Cooley-Tukey Algorithm. However, all require either the same or more arithmetical operations as the method described previously and shown in the flow chart in Figure 2. This paper, however, is mainly concerned with a new algorithm, which uses the same number of additions, but only half as many multiplications as needed by the Cooley-Tukey Algorithm.

The number of additions and multiplications required for the Cooley-Tukey Algorithm can be obtained by inspection of Figure 2. They are as follows:

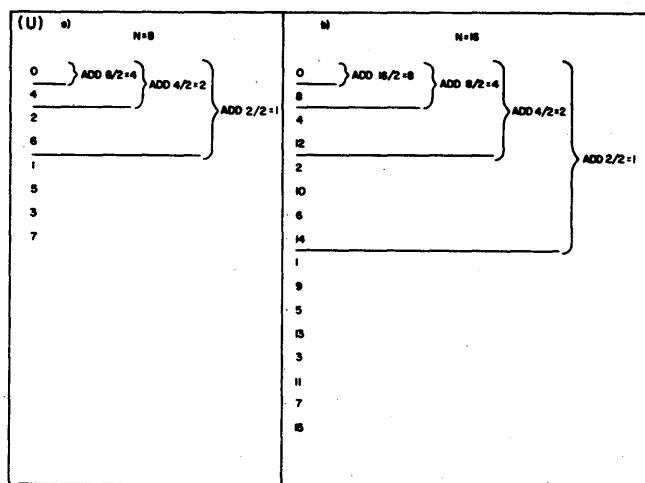


FIGURE 4—Method for generating the bit reversal order

Number of complex additions and subtractions:

$$\text{Additions} = N \text{Log}_2 N = N\nu \tag{8}$$

Number of complex multiplications:

$$\text{Multiplications} = \frac{N}{2} (\nu - 1)$$

Deleting the multiplications $e^{-j \frac{2\pi 0}{N}} = 1$; and

$e^{-j \frac{2\pi N/4}{N}} = -j$, which are not computed by multiplying, we obtain:

$$\text{Multiplications} = \frac{N}{2} (\nu - 1) - 2\left(\frac{N}{2} - 1\right) = \frac{N}{2} (\nu - 3) + 2 \tag{10}$$

Expressing equations (8) and (10) in terms of real additions and multiplications we have:

$$\text{Real additions} = 2N\nu + 2 \left[\frac{N}{2} (\nu - 3) + 2 \right] = 3N (\nu - 1) + 4 \tag{11}$$

and real multiplications = $2N (\nu - 3) + 8$

The extra increase in the number of additions is based on the fact that a complex multiplication is made up of four real multiplications and two real additions.

The amount of savings in operations is easily observed when compared with the straightforward pro-

cessing of equation (1) which requires multiplications and additions each of the order of N^2 .

As shown above, the Cooley-Tukey Algorithm is economical in the required number of add and multiply operations. However, this is not yet optimum. The search for a more economical way of solving equation (1) results in a new algorithm which is more economical than the Cooley-Tukey Algorithm. The required number of real additions is the same in both algorithms; however, the required number of real multiplications has been cut in half to:

$$\text{Real multiplications} = N (\nu - 3) + 4 \tag{13}$$

This algorithm is believed to be optimum as far as the number of required operations is concerned; however, it is more complicated, and therefore may result in a more difficult control problem when it is processed by a digital computer.

The new algorithm

The analysis of the new algorithm is restricted to numbers of inputs, N , which are powers of 2, namely, $N = 2^\nu$. The start of the algorithm is the same as in the Cooley-Tukey Algorithm. The input points are divided into σ 's (sums) and δ 's (differences). The σ 's are again treated as a new set of input points, as shown by equation (6). However, the δ 's are not multiplied, as in equation (7), but are processed as shown by the following equation:

$$\begin{aligned} N^{\ell} F_{2\ell+1} &= \sum_{k=0}^{\frac{N}{2}-1} \delta_k e^{-j \frac{2\pi k(2\ell+1)}{N}} = \left[\delta_0 + (-1)^\ell (-j) \delta_{\frac{N}{4}} \right] \\ &+ \sum_{k=1}^{\frac{N}{2}-1} \left\{ \left[\delta_k + (-1)^\ell (-j) \delta_{\frac{N}{4}-k} \right] - \left[\delta_{\frac{N}{2}-k} - (-1)^\ell (-j) \delta_{\frac{N}{4}+k} \right] \right\} \\ &\cos \frac{2\pi k(2\ell+1)}{N} = \left[\delta_0 + (-1)^\ell (-j) \delta_{\frac{N}{4}} \right] + \sum_{k=1}^{\frac{N}{2}-1} \left[(\delta_k - \delta_{\frac{N}{2}-k}) \right. \\ &\left. + (-1)^\ell (-j) (\delta_{\frac{N}{4}-k} + \delta_{\frac{N}{4}+k}) \right] \cos \frac{2\pi k(2\ell+1)}{N} \end{aligned} \tag{14}$$

This equation is valid for $\ell = 0, 1, 2, \dots, \left(\frac{N}{2} - 1\right)$.

Equation (14) was derived with the aid of the following equations:

$$e^{-j \frac{2\pi k(2\ell+1)}{N}} = \cos \frac{2\pi k(2\ell+1)}{N} - j \sin \frac{2\pi k(2\ell+1)}{N} \tag{15}$$

$$\cos \frac{2\pi k(2\ell+1)}{N} = (-1)^\ell \sin \frac{2\pi \left(\frac{N}{4} - k\right)(2\ell+1)}{N} = (-1)^\ell \sin \frac{2\pi \left(\frac{N}{4} + k\right)(2\ell+1)}{N} \tag{16}$$

$$\cos \frac{2\pi k(2\ell+1)}{N} = \cos \frac{2\pi \left(\frac{N}{2} - k\right)(2\ell+1)}{N} \tag{17}$$

Equation (14) actually requires as many multiplications as equation (7). However, the complex multipliers of equation (7) are replaced by real cosine multipliers in equation (14). A multiplication of a complex number by a complex number requires, in general, four real multiplications and two real additions. A multiplication of a complex number by a real number requires only two real multiplications and no real additions. The saving in the number of real multiplications holds for the new algorithm, but the saving in additions does not hold since the new processing requires exactly as many new additions as has been saved by the real cosine multipliers. In equation (14) it is seen that if $\ell = 2m$ is even, then the term $(-1)^\ell$ is equal to $+1$, and if $\ell = 2m + 1$ is odd, then the term $(-1)^\ell$ is equal to -1 . Therefore, we may break up equation (14) into the following two equations:

For ℓ even we have:

$$NF_{4m+1} = (\delta_0 - j \delta_{\frac{N}{4}}) + \sum_{k=1}^{\frac{N}{4}-1} \left[(\delta_k - \delta_{\frac{N}{2}-k}) - j (\delta_{\frac{N}{4}-k} + \delta_{\frac{N}{4}+k}) \right] \cos \frac{2\pi k(4m+1)}{N} \quad (18)$$

and for ℓ odd we have:

$$NF_{4m+3} = (\delta_0 + j \delta_{\frac{N}{4}}) + \sum_{k=1}^{\frac{N}{4}-1} \left[(\delta_k - \delta_{\frac{N}{2}-k}) + j (\delta_{\frac{N}{4}-k} + \delta_{\frac{N}{4}+k}) \right] \cos \frac{2\pi k(4m+3)}{N} \quad (19)$$

Both equations hold for $m = 0, 1, 2, \dots, (\frac{N}{4} - 1)$

If we reverse the order in equation (19) we obtain:

$$NF_{4M+3} = (\delta_0 + j \delta_{\frac{N}{4}}) + \sum_{k=1}^{\frac{N}{4}-1} \left[(\delta_k - \delta_{\frac{N}{2}-k}) + j (\delta_{\frac{N}{4}-k} + \delta_{\frac{N}{4}+k}) \right] \cos \frac{2\pi k(4M+3)}{N} = (\delta_0 + j \delta_{\frac{N}{4}}) + \sum_{k=1}^{\frac{N}{4}-1} \left[(\delta_k - \delta_{\frac{N}{2}-k}) + j (\delta_{\frac{N}{4}-k} + \delta_{\frac{N}{4}+k}) \right] \cos \frac{2\pi k(4m+1)}{N} \quad (20)$$

For $M = (\frac{N}{4} - 1), (\frac{N}{4} - 2), (\frac{N}{4} - 3), \dots, (\frac{N}{4} - \frac{N}{4})$

and $m = 0, 1, 2, \dots, (\frac{N}{4} - 1)$

since:

$$\cos \frac{2\pi k \left[4(\frac{N}{4} - m) + 3 \right]}{N} = \cos \frac{2\pi k(3 - 4m)}{N} = \cos \frac{2\pi k \left[4(\frac{N}{4} - 1) + 1 \right]}{N} \quad (20a)$$

To simplify the expression of equations (18) and (20) we introduce the following notation:

$$(\delta_0 - j \delta_{\frac{N}{4}}) = P_0 \quad (21)$$

$$\left[(\delta_k - \delta_{\frac{N}{2}-k}) - j (\delta_{\frac{N}{4}-k} + \delta_{\frac{N}{4}+k}) \right] = P_k \quad (22)$$

$$(\delta_0 + j \delta_{\frac{N}{4}}) = Q_0 \quad (23)$$

$$\left[(\delta_k - \delta_{\frac{N}{2}-k}) + j (\delta_{\frac{N}{4}-k} + \delta_{\frac{N}{4}+k}) \right] = Q_k \quad (24)$$

$$\cos \frac{2\pi k(4m+1)}{N} = C_k(4m+1) \quad (25)$$

With these notations we obtain for equations (18) and (20):

$$NF_{4m+1} = P_0 + \sum_{k=1}^{\frac{N}{4}-1} P_k C_k(4m+1) = \sum_{k=0}^{\frac{N}{4}-1} P_k C_k(4m+1) \quad (26)$$

for $m = 0, 1, 2, \dots, (\frac{N}{4} - 1)$

$$NF_{4M+3} = Q_0 + \sum_{k=1}^{\frac{N}{4}-1} Q_k C_k(4m+1) = \sum_{k=0}^{\frac{N}{4}-1} Q_k C_k(4m+1) \quad (27)$$

for $M = (\frac{N}{4} - 1), (\frac{N}{4} - 2), (\frac{N}{4} - 3) \dots 2, 1, 0$
and $m = 0, 1, 2, \dots, (\frac{N}{4} - 3), (\frac{N}{4} - 2), (\frac{N}{4} - 1)$

Equations (26) and (27) will require identical processing procedures. However, the outputs for the odd ℓ 's will reverse their order as compared to the even ℓ 's (which will appear in the bit reversal order, as will be shown later in this paper).

A flow chart for a sixteen point input case for processing the δ 's up to the P_k and Q_k terms is shown in Figure 5.

Equations (26) and (27) are of the same form as equation (1) but with real cosine multipliers instead of the complex exponential multipliers of equation (1). Unfortunately, the method of processing which leads to the Cooley-Tukey Algorithm is not applicable to equations (26) and (27). The reason for this is that an exponential multiplier can be split in a manner shown in the following equation:

$$e^{-j(a+b)} = e^{-ja} \cdot e^{-jb} \quad (28)$$

This does not hold for a cosine multiplier, since in general:

$$\cos(a+b) \neq \cos a \cdot \cos b \quad (29)$$

A new processing algorithm, which has the same order

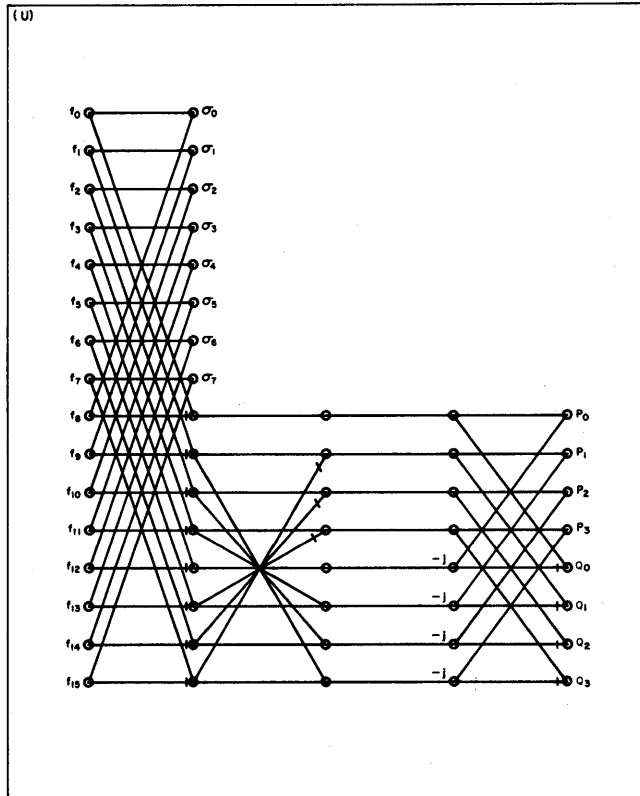


FIGURE 5—Generation of the P and Q set from a set of 16 input points

of efficiency as the one used in the Cooley-Tukey Algorithm, had to be designed for the cosine multipliers.

Since the P_k 's and Q_k 's require identical processing, only the P_k 's need be discussed. First, we multiply the P_k 's by the $C_k(4m + 1)$'s. Figure 6 show the multiplication process of the P's for the case $N = 16$. To ob-

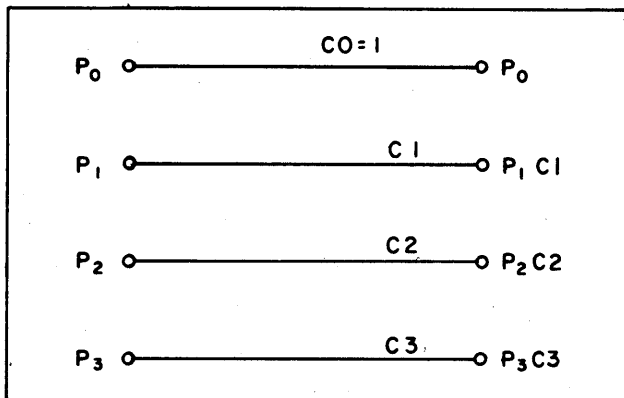


FIGURE 6—First multiplier set for the P set for the case $N = 16$

tain the NF_{4m+1} output term, add all the multiplied P_k terms. The order of obtaining the output terms depends upon the order of selecting the m 's. For simplicity, we start with $m = 0$. The first output term will therefore be NF_1 . This output term also is the first odd term in the bit reversal order. The next output term in the bit reversal order is $NF_{\frac{N}{2}+1}$. The calculation of this

output term is related to the calculation of the term NF_1 , and therefore will be computed next. To obtain the $NF_{\frac{N}{2}+1}$ output term we must subtract the sum of all the odd multiplied P_k terms. This is easily seen from equation (26) which results in the following equation:

$$\begin{aligned}
 NF_{\frac{N}{2}+4m+1} &= \sum_{k=0}^{\frac{N}{4}-1} P_k C_k (\frac{N}{2} + 4m+1) = \sum_{k=0}^{\frac{N}{4}-1} P_k \cos 2\pi k \frac{N}{2} + \frac{4m+1}{N} \\
 &= \sum_{k=0}^{\frac{N}{4}-1} (-1)^k P_k C_k (4m+1) \tag{30}
 \end{aligned}$$

To make best use of equation (30) we place a constraint on the method of adding, that is, the even multiplied P_k terms should be added separately from the odd P_k terms. At the end, the sum of the even terms and the odd terms will yield NF_1 , and the difference of the even terms and the odd terms will yield $NF_{\frac{N}{2}+1}$. One method of adding

the $(\frac{N}{4} - 1)$ terms, which fulfills the above/mentioned constraint, is as follows: First, add p_0 and the middle term $P_{\frac{N}{8}} C(\frac{N}{8})$; next add $P_k C_k$ and $P_{\frac{N}{4}-k} C(\frac{N}{4} - k)$ for $k = 1, 2, 3, \dots, (\frac{N}{8} - 1)$. The output is $\frac{N}{8}$ terms added by pairs. The reason for selecting this method of addition becomes clear from the following analysis. To simplify further analysis we introduce the new notation

$$P_k C \gamma k = \gamma U_k \tag{31}$$

with this notation, the first process of addition yields two results ${}_1A_1$ and ${}_1B_{1,k}$ where:

$${}_1A_1 = [{}_1U_0 + {}_1U_{\frac{N}{8}}] \tag{32}$$

and:

$${}_1B_{1,k} = [{}_1U_k + {}_1U_{\frac{N}{4}-k}] \text{ for } k = 1, 2, 3, \dots, (\frac{N}{8} - 1) \tag{33}$$

We repeat the same process of addition with ${}_1A_1$ and the

$1B_{1,k}$'s, which consist of $\frac{N}{8}$ terms, and obtain:

$$1A_2 = [1A_1 + 1B_{1, \frac{N}{8}}] = [1U_0 + 1U_{\frac{N}{8}} + 1U_{\frac{N}{16}} + 1U_{\frac{3N}{16}}] \quad (34)$$

$$1B_{2,k} = [1B_{1,k} + 1B_{1, \frac{N}{8} - k}] = [1U_k + 1U_{\frac{N}{4} - k} + 1U_{\frac{N}{8} - k} + 1U_{\frac{N}{8} + k}] \text{ for } k = 1, 2, 3, \dots, (\frac{N}{16} - 1) \quad (35)$$

After the i th repetition we have:

$$1A_i = [1U_0 + 1U_{\frac{N}{8}} + 1U_{\frac{N}{16}} + 1U_{\frac{3N}{16}} + 1U_{\frac{N}{32}} + 1U_{\frac{3N}{32}} + 1U_{\frac{5N}{32}} + 1U_{\frac{7N}{32}} + \dots + 1U_{\frac{N}{2^{i+2}}} + 1U_{\frac{3N}{2^{i+2}}} + 1U_{\frac{5N}{2^{i+2}}} + \dots + 1U_{\frac{(2^i - 1)N}{2^{i+2}}}] \quad (36)$$

$$1B_{i,k} = [1U_k + 1U_{\frac{N}{4} - k} + 1U_{\frac{N}{8} - k} + 1U_{\frac{N}{8} + k} + 1U_{\frac{N}{16} - k} + 1U_{\frac{N}{16} + k} + 1U_{\frac{3N}{16} - k} + 1U_{\frac{3N}{16} + k} + \dots + 1U_{\frac{N}{2^{i+1}} - k} + 1U_{\frac{N}{2^{i+1}} + k} + 1U_{\frac{3N}{2^{i+1}} - k} + 1U_{\frac{3N}{2^{i+1}} + k} + \dots + 1U_{\frac{(2^{i-1} - 1)N}{2^{i+1}} - k} + 1U_{\frac{(2^{i-1} - 1)N}{2^{i+1}} + k}] \quad (37)$$

for $k = 1, 2, 3, \dots, (\frac{N}{2^{i+2}} - 1)$

After $(\nu-3)$ repetitions we have:

$$1A_{\nu-3} = [1U_0 + 1U_{\frac{N}{8}} + 1U_{\frac{N}{16}} + 1U_{\frac{3N}{16}} + \dots + 1U_2 + 1U_6 + 1U_{10} + \dots + 1U_{2(\frac{N}{8} - 1)}] \quad (38)$$

$$1B_{\nu-3,k} = 1B_{\nu-3,1} = [1U_k + 1U_{\frac{N}{4} - k} + 1U_{\frac{N}{8} - k} + 1U_{\frac{N}{8} + k} + \dots + 1U_{4-k} + 1U_{4+k} + 1U_{12-k} + 1U_{12+k} + 1U_{20-k} + 1U_{20+k} + \dots + 1U_{4(\frac{N}{16} - 1) - k} + 1U_{4(\frac{N}{16} - 1) + k}] \text{ for } k = 1 = (\frac{N}{2^{\nu-3+2}} - 1) \quad (39)$$

A simple inspection of equations (38) and (39) shows that $A_{\nu-3}$ contains all the even U terms and $B_{\nu-3,1}$ contains all the odd U terms. Adding and subtracting yields:

$$NF_1 = 1A_{\nu-3} + 1B_{\nu-3,1} \quad (40)$$

$$\frac{NF_{\frac{N}{2} + 1}}{2} = 1A_{\nu-3} - 1B_{\nu-3,1} \quad (41)$$

A flow chart for adding the U's is shown in Figure 7 for the case of $N = 64$.

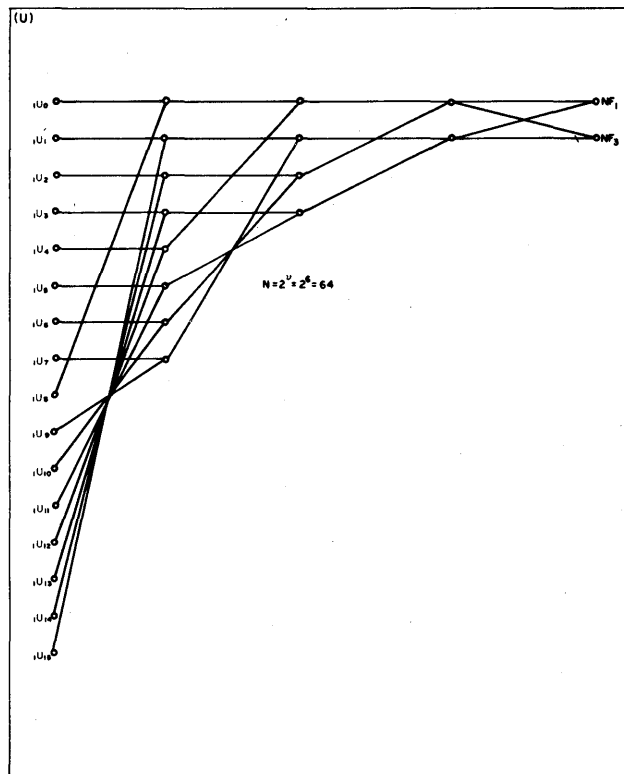


FIGURE 7—Addition of the “U” set for the case $N = 64$

It is naturally not enough to be able to compute two output terms. It should be possible, in the adding procedure, to produce new terms so that at the end of the summation all the output terms will be available. To accomplish this, each time we add terms, we also subtract them. It will be shown that the difference terms can be used to produce new terms for computing the other output terms. The principle of producing the new terms is based on the following formula:

$$2 \cos a \cos \beta = \cos(a+\beta) + \cos(a-\beta) \quad (42)$$

Before using this formula return to the first step in the addition process, as given in equations (32) and (33). The difference terms are produced as follows:

$$1D_1 = [1U_0 - 1U_{\frac{N}{8}}] = [P_0 - P_{\frac{N}{8}} C \frac{N}{8}] = [P_0 + P_{\frac{N}{8}} C \frac{3N}{8}] = 3A_1 \quad (43)$$

$$1E_{1,k} = [1U_k - 1U_{\frac{N}{4} - k}] \text{ for } k = 1, 2, 3, \dots, (\frac{N}{8} - 1) \quad (44)$$

Multiply equation (44) by $(-2C2k)$, and using equation (42) obtain:

$$\begin{aligned}
 \left[{}_1U_k - {}_1U_{\frac{N}{4}-k} \right] (-2C2k) &= -P_k C3k + P_{\frac{N}{4}-k} C\left(\frac{N}{4}-3k\right) - P_k Ck \\
 + P_{\frac{N}{4}-k} C\left(\frac{N}{4}+k\right) &= -P_k C3k - P_{\frac{N}{4}-k} C3\left(\frac{N}{4}-k\right) - P_k Ck \\
 - P_{\frac{N}{4}-k} C\left(\frac{N}{4}-k\right) &= -\left[{}_3U_k + {}_3U_{\frac{N}{4}-k} \right] - \left[{}_1U_k + {}_1U_{\frac{N}{4}-k} \right] \\
 &= -{}_3B_{1,k} - {}_1B_{1,k}; \text{ for } k = 1, 2, 3, \dots, \left(\frac{N}{8}-1\right). \quad (45)
 \end{aligned}$$

The terms ${}_1B_{1,k}$ were already computed. Adding them to equation (45) we obtain:

$${}_1E_{1,k} (-2C2k) + {}_1B_{1,k} = -{}_3B_{1,k}; \text{ for } k = 1, 2, 3, \dots, \left(\frac{N}{8}-1\right). \quad (46)$$

The expressions ${}_3A_1$ and $-{}_3B_{1,k}$ are the result of the first step in the previously discussed addition process, except that the ${}_3B_{1,k}$'s have a minus sign in front of them. We may continue:

$${}_3A_2 = \left[{}_3A_1 - \left(-{}_3B_{1, \frac{N}{16}} \right) \right] \quad (46a)$$

and:

$$-{}_3B_{2,k} = \left[-{}_3B_{1,k} + \left(-{}_3B_{1, \frac{N}{8}-k} \right) \right] \quad (46b)$$

for $K = 1, 2, 3, \dots, \left(\frac{N}{16}-1\right)$.

continuing this process we obtain the terms ${}_2A_{v-3}$ and $-{}_3B_{v-3,1}$; but, ${}_2A_{v-3} = {}_{N-3}A_{v-3}$, and $-{}_3B_{v-3,1} = -{}_{N-3}B_{v-3,1}$, since

$$C_{\eta k} = \cosine \frac{2\pi \eta k}{N} = \cosine \frac{2\pi k(N-\eta)}{N} = C(N-\eta)k \quad (47)$$

Adding the terms ${}_{N-3}A_{v-3}$ and $-{}_{N-3}B_{v-3,1}$ we obtain the output term $NF_{\frac{N+N-3}{2}}$ which is equal to $NF_{\frac{N-3}{2}}$.

Subtracting the term $-{}_{N-3}B_{v-3,1}$ from ${}_{N-3}A_{v-3}$ we obtain the output term NF_{N-3} . We cannot obtain the output terms $NF_{\frac{N+3}{2}}$ and NF_3 because these output terms are constructed by the Q expressions and not by the P's.

After completing the first step of adding and subtracting, and after multiplying the subtracted terms by the appropriate cosines we will have two sets of terms, those prefixed by ones (${}_1A_1$ and the ${}_1B_{1,k}$'s) and those by threes (${}_3A_1$ and the $-{}_3B_{1,k}$'s). At this point we repeat the addition and subtraction process for each of the sets to obtain ${}_1A_2$ and the ${}_1B_{2,k}$'s and ${}_1D_2$ and the ${}_1E_{2,k}$'s. We also obtain ${}_3A_2$ and the $-{}_3B_{2,k}$'s and ${}_3D_2$ and the $-{}_3E_{2,k}$'s. Figure 8 is a flow chart for $N = 64$, which shows the first two steps of the addition and subtraction process discussed above, and also the processing of equation (46). Notice that the results after processing equation (46) are in reverse order and there-

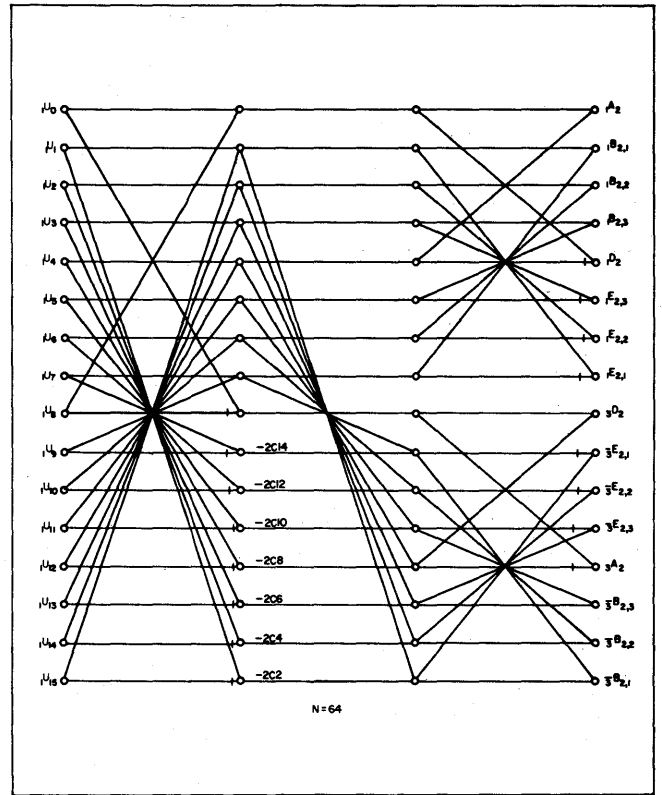


FIGURE 8—Processing of the "U" set into A,B,E, and D sets of the second order

fore the appropriate second step of the addition and subtraction process had to be reversed.

The described method of processing is general and applies also for the case where m is not equal to zero. In this case, the appropriate multipliers would be $-2C2K(4m + 1)$ as seen from the following equation:

$$\begin{aligned}
 \left[P_k Ck(4m+1) - P_{\frac{N}{4}-k} C\left(\frac{N}{4}-k\right)(4m+1) \right] \left[-2C2k(4m+1) \right] + P_k Ck(4m+1) \\
 + P_{\frac{N}{4}-k} C\left(\frac{N}{4}-k\right)(4m+1) &= -P_k C3k(4m+1) - P_{\frac{N}{4}-k} C3\left(\frac{N}{4}-k\right)(4m+1) \\
 &= -P_k C(N-3)k(4m+1) - P_{\frac{N}{4}-k} C(N-3)\left(\frac{N}{4}-k\right)(4m+1) \quad (48)
 \end{aligned}$$

for $k = 0, 1, 2, \dots, \left(\frac{N}{8}-1\right)$

Actually two new pairs of data can be obtained by multiplying by $2C(2^{2r+1}k)(4m + 1)$ for any positive integer of r. However only the multiplication by $2C2k(4m + 1)$ results in two pairs of data where one pair is already available and can be subtracted to obtain the new pair.

The subtraction terms in the second step of the data processing are similar to the addition terms given in equations (34) and (35), and are as follows:

$$\begin{aligned}
 {}_1P_2 &= [{}_1U_0 + {}_1U_{\frac{N}{8}} - {}_1U_{\frac{N}{16}} - {}_1U_{\frac{3N}{16}}] = [{}_7U_0 + {}_7U_{\frac{N}{8}} + {}_7U_{\frac{N}{16}} + {}_7U_{\frac{3N}{16}}] \\
 &= {}_7A_2 \tag{49}
 \end{aligned}$$

We also have:

$${}_1E_{2,k} = [{}_1B_{1,k} - {}_1B_{1,\frac{N}{8}-k}] = [{}_1U_k + {}_1U_{\frac{N}{4}-k} - {}_1U_{\frac{N}{8}-k} - {}_1U_{\frac{N}{8}+k}] \tag{50}$$

For $k = 1, 2, 3, \dots, (\frac{N}{16} - 1)$

We multiply equation (50) by $+(2C4k)$ and obtain:

$$\begin{aligned}
 {}_1E_{2,k}(2C4k) &= [{}_5U_k + {}_5U_{\frac{N}{4}-k} + {}_5U_{\frac{N}{8}-k} + {}_5U_{\frac{N}{8}+k}] + [{}_3U_k + {}_3U_{\frac{N}{4}-k} \\
 &+ {}_3U_{\frac{N}{8}-k} + {}_3U_{\frac{N}{8}+k}] = {}_5B_{2,k} + {}_3B_{2,k} \text{ For} \\
 k &= 1, 2, \dots, (\frac{N}{16} - 1)
 \end{aligned} \tag{51}$$

The terms $-{}_3B_{2,k}$ have already been computed, and by adding them to equation (51) we obtain:

$${}_1E_{2,k}(2C4k) + (-{}_3B_{2,k}) = {}_5B_{2,k} \text{ For } k = 1, 2, \dots, (\frac{N}{16} - 1) \tag{52}$$

From equation (46)_a and (46)_b we have:

$$\begin{aligned}
 {}_3D_2 &= [{}_3A_1 + (-{}_3B_{1,\frac{N}{16}})] = [{}_3U_0 + {}_3U_{\frac{N}{8}} - {}_3U_{\frac{N}{16}} - {}_3U_{\frac{3N}{16}}] \\
 &= [{}_5U_0 + {}_5U_{\frac{N}{8}} + {}_5U_{\frac{N}{16}} + {}_5U_{\frac{3N}{16}}] = {}_5A_2
 \end{aligned} \tag{53}$$

and

$$\begin{aligned}
 -{}_3E_{2,k} &= [-{}_3B_{1,k} + (-{}_3B_{1,\frac{N}{8}-k})] = - [{}_3U_k + {}_3U_{\frac{N}{4}-k} - {}_3U_{\frac{N}{8}-k} \\
 &- {}_3U_{\frac{N}{8}+k}] \text{ For } k = 1, 2, \dots, (\frac{N}{16} - 1)
 \end{aligned} \tag{54}$$

Multiplying equation (54) by $(2C4k)$ we obtain:

$$\begin{aligned}
 -{}_3E_{2,k}(2C4k) &= - [{}_7U_k + {}_7U_{\frac{N}{4}-k} + {}_7U_{\frac{N}{8}-k} + {}_7U_{\frac{N}{8}+k}] \\
 &- [{}_1U_k + {}_1U_{\frac{N}{4}-k} + {}_1U_{\frac{N}{8}-k} + {}_1U_{\frac{N}{8}+k}] = -{}_7B_{2,k} - {}_1B_{2,k} \tag{55}
 \end{aligned}$$

For $k = 1, 2, \dots, (\frac{N}{16} - 1)$

The terms ${}_1B_{2,k}$ have already been computed, and by adding them to equation (55) we obtain:

$$-{}_3E_{2,k}(2C4k) + {}_1B_{2,k} = -{}_7B_{2,k} \tag{56}$$

Completing the addition process for the two sets we will obtain: $({}_5A_{v-3} \pm {}_5B_{v-3})$ which are equal to the respective output terms $NF_{\frac{N}{2}}^5$ and NF_{N-7}^5 . We will also obtain ${}_7A_{v-3} \pm (-{}_7B_{v-3})$ which are equal to the

respective output terms $NF_{\frac{N}{2}}^7$ and NF_{N-7}^7 (see equation (47)).

In the second step of processing we see that the data in the two sets get interchanged. The first set results in ${}_7A_2$, and, ${}_5B_{2,k}$, and the second set in ${}_5A_2$, and $-{}_7B_{2,k}$. To simplify the processing, we exchange the places of the ${}_1E_{2,k}$ and the $-{}_3E_{2,k}$ terms before they are multiplied by the $2C4K$ terms. Figure 9 is a flow chart for the case $N = 64$, showing the second step of addition and subtraction, the exchange of terms, and the multiplication by the $2C4K$ terms.

Inspection of Figure 9 shows that from this point the processing becomes repetitious. We started with two sets of data $({}_1A_1; {}_1B_{1,k})$ and $({}_3A_1; {}_3B_{1,k})$, and finished with double the amount, namely four sets of data, one step further in the processing. We ended with $({}_1A_2; {}_1B_{2,k})$, $({}_7A_2; -{}_7B_{2,k})$, $({}_5A_2; {}_5B_{2,k})$ and $({}_3A_2; -{}_3B_{2,k})$. The sets with a prefix equal to one, modulo four, have plus B_k 's, and appear in the chart with larger k 's going down the column. The sets with a prefix equal to 3, modulo four, have minus B_k 's, and appear in the chart with larger k 's going up the column. Figure 10 shows the next

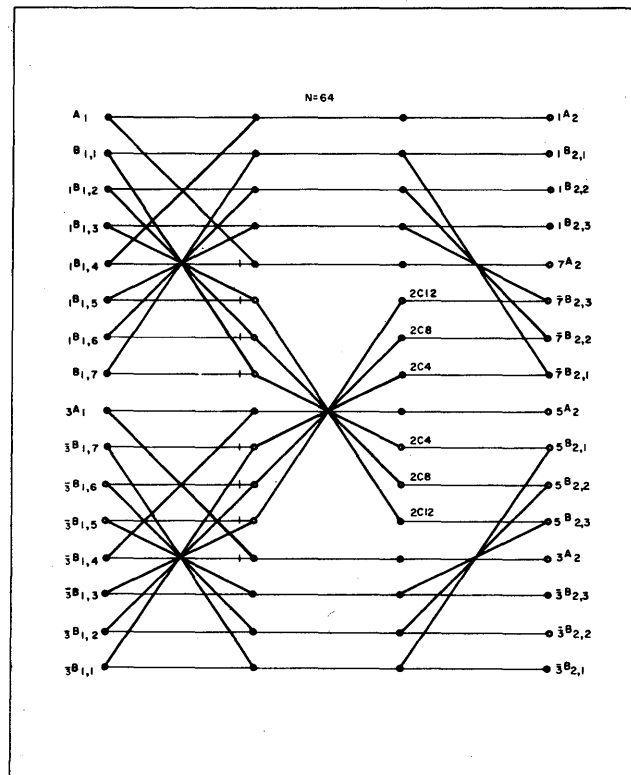


FIGURE 9—Processing of 2 sets of A and B of the first order into 4 sets of A and B of the second order

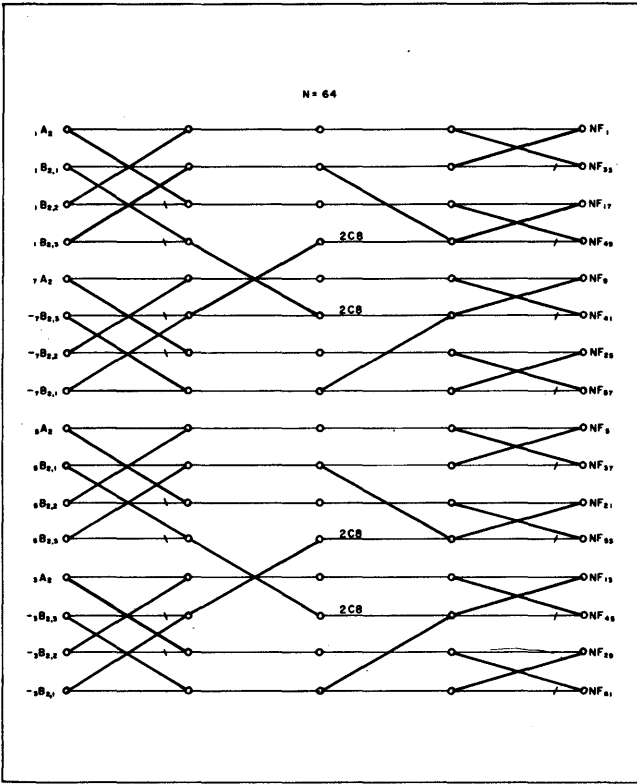
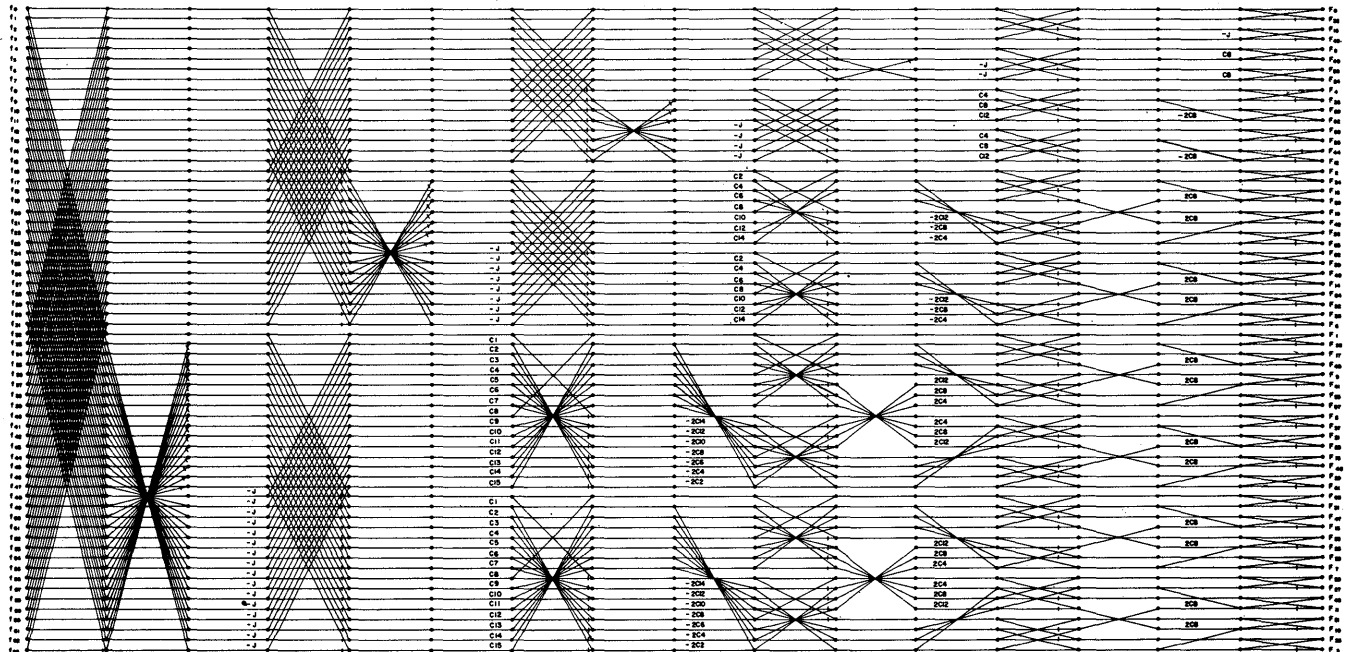


FIGURE 10—Processing of 4 sets of A and B of the second order into 16 final outputs for the case N = 64

step in the processing for $N = 64$ and also the last step and the final outputs.

Inspection of Figure 10 shows that the output terms based on the P's occupy the first half of the odd numbered output terms in the bit reversed order for $N = 64$. Reviewing the method of processing it is seen that we began with the set of data $(1U_0; 1U_k)$ for $k = 1, 2, 3, \dots, (\frac{N}{4} - 1)$. This set may be renamed as $(1A_0; 1B_{0,k})$. After addition, subtraction, and multiplication by $(-2C2k)$ we arrived at the two sets $(1A_1; 1B_{1,k})$ and $(3A_1; -3B_{1,k})$. From this point the method became repetitive. We added and subtracted the two sets of data; we exchanged positions of the two neighboring E's and multiplied each set by $(2C4k)$. This resulted in four sets of data, namely, $(1A_2; 1B_{2,k})$, $(3A_2; -3B_{2,k})$, $(5A_2; 5B_{2,k})$ and $(7A_2; -7B_{2,k})$. This process may be continued until the final outputs are obtained, as shown in Figure 10. The author proved that this continuation is always possible. The proof is based on a full induction method. However, for brevity the proof is not included in this paper. The author also proved that the final result will be in the bit reversed order. This proof is also based on a full induction approach. A complete flow chart for this process is shown in Figure 11 for the Case $N = 64$. It is seen from Figure 11 that the flow chart pattern is repeated every three columns. The first nine columns do

FIGURE 11—Flow chart for the discrete fourier transform for the case N = 64



- 1) THE FLOW IS FROM LEFT TO RIGHT
- 2) UNSHARDED LINES HAVE A GAIN OF ONE
- 3) SHARDED LINES HAVE A GAIN OF MINUS ONE
- 4) $2C = \cos(\frac{2\pi}{N})$
- 5) $-j = -\sqrt{-1}$

not show certain details of the pattern because the columns of the flow chart are terminated at the Nth input point, while the missing details in the first nine columns should appear after the Nth input point.

It should also be noted that when using this algorithm the data processing can be decomposed into two identical channels, each using real numbers only. The real parts of the input points would go into the first channel and the imaginary parts of the input points would go to the second channel. The data processing would be identical in both channels. A multiplication by $-j$ would mean a multiplication of the data of the first channel by -1 and a transfer of these data (after the multiplication by -1) to the identical places in the second channel, while the data from the second channel are transferred to the identical vacated places in the first channel.

The number of operations required for this algorithm can now be easily established by inspection. The number of multipliers is the same in this algorithm and in the Cooley-Tukey Algorithm. However, since the multipliers in the new algorithm are all real, the actual number of real multiplications is cut in half and is therefore (based on equation (12)) equal to:

$$\text{Real multiplications} = N(\nu-3)+4 \quad (57)$$

Equation (57) is, naturally, the same as equation (13).

Also, in general, every complex by a complex multiplication requires two real additions, while no additions are required for a complex by a real multiplication. This will result in a saving of $N(\nu-3) + 4$ real additions in the new algorithm. However, the new algorithm requires extra addition operations, which do not appear in the Cooley-Tukey Algorithm. Inspection of Figure 11 shows that there is one extra complex addition with every multiplication by cosine of two times an angle. There are no extra additions required for a multiplication by cosine of one times the angle. However, Figure 11 shows that there is an extra addition and subtraction processing step prior to the cosine one time angle multiplications which requires exactly the same number of complex additions and subtractions as there are cosine one time the angle multiplications. Therefore, the total number of extra complex additions and subtractions is equal to the number of the complex by real multiplications. This exactly wipes out the saving of the $N(\nu-3) + 4$ real additions mentioned previously. The total number of real additions required for the new algorithm will therefore be equal to the number required by the Cooley-Tukey Algorithm. This number is given in equation (11) and is rewritten here:

$$\text{Real adds} = 3N(\nu-1)+4 \quad (58)$$

We have thus far treated the case where the input data was complex. Considerable simplifications and saving in operations is achieved when the input data is real.

We first observe that in any case the multipliers are real cosine numbers. Therefore, even in the complex input case, the data processing can be divided into two parallel channels. The real part of the input data will be the input to the first channel and the imaginary part of the input data will be the input to the other channel. The data processing in each channel will be the same and will be real. The only difference between the two channels will be in the interpretation of a multiplication by $-j$. In the first channel the data is multiplied by -1 , and transferred to the identical place in the second channel. In the second channel, a multiplication by $-j$ will mean that the data are transferred to the same place in the first channel (without a multiplication by -1). When the input data are real, the second channel stays empty in the beginning. Only when the data processing reaches the first set of multiplication by $-j$ will data be transferred to the second channel. There will be no data to be transferred from the second to the first channel. After each set of $-j$ multiplications, there is an addition and subtraction operation as shown in Figures 5 and 11. If no data are transferred from the second channel to the first, the addition and subtraction would result in the Q's being equal to the respective P's. Since the Q's and the P's have identical data processing, and no more multiplications by $-j$, this means that two identical sets of output terms will be obtained. It is therefore clear that we can process only the P's and leave the spaces for the Q's empty. Physically this could be achieved by eliminating the diagonal lines in the addition and subtraction process which follows the $-j$ multiplications.

Now, instead of transferring the data from the first channel to the second, the data are left where they are, since the $-j$ lines are also the empty Q lines. We will now have a new interpretation for a multiplication by $-j$. It will mean that the numbers become second channel numbers. (That the Q lines become second channel lines.) The required multiplication by minus one will be correctly performed since the processing of the Q's requires a multiplication by minus one. The final result of this analysis is that one channel can process the Fourier Transform when the input data are real. The two channels can therefore process two independent sets of real input data with a saving in the number of operations as compared to the complex data input case. A complete flow chart for processing eight complex and

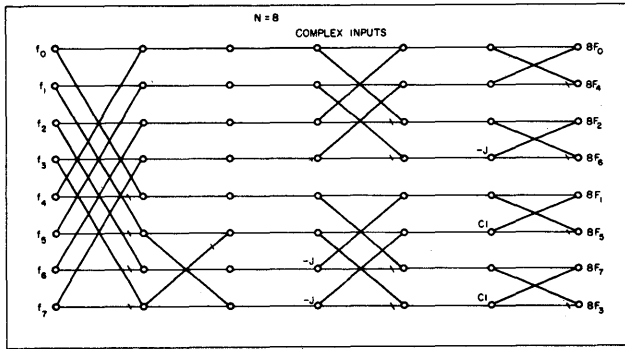


FIGURE 12—Flow chart for the new data processing method of the Fourier transform for the case N = 8

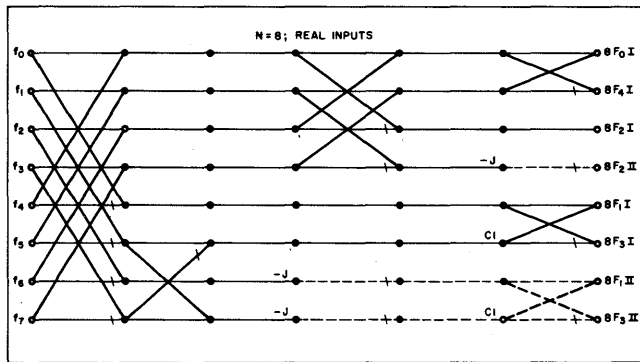


FIGURE 13—Flow chart of the new processing method for the case of 8 real inputs

eight real input points is shown in Figures 12 and 13, respectively.

In Figure 13 we changed the output numbers which are larger than four to output numbers smaller than four by use of the following formula:

$$F_K = F_{N-K} \tag{59}$$

This formula results from the foldover of the data when only real inputs are used. This can be seen from Figure 12, when only real data are used, since $8F_2$, $8F_1$ and $8F_6$, will be equal to $8F_6$, $8F_7$ and $8F_3$ respectively due to the Q's being equal to the P's.

To compute the required number of operations, start with the addition and subtraction operations. From Figures 12 and 13 we see that after every $-j$ multiplication there is a saving of two additional operations. The number of $-j$ multiplications is easily seen, by inspection

of Figure 11, to be equal to $(\frac{N}{2} - 1)$. Therefore, the number of saved addition and subtraction operations is equal to:

$$2. (\frac{N}{2} - 1) = N - 2 \tag{60}$$

Equation (58) shows the total number of real additions and subtractions for the complex case. For real inputs we will have half the number of additions and subtractions. Combining this with the savings given by equation (60), the total number of additions and subtractions in the real input case is:

Real additions and subtractions

$$\text{Real additions and subtractions} = \frac{3}{2} N(\nu-1)+2-N+2 = \frac{N}{2} (3\nu-5)+4 \tag{61}$$

The number of multiplication operations for the real input case is the same as for the complex case. The number of multiplication operations for the complex case is given by equation (10). Therefore, the total number of real multiplications for the real input case is equal to:

$$\text{Real multiplications} = \frac{N}{2} (\nu-3)+2. \tag{62}$$

Presently, the described method for computing the Discrete Fourier Transform is limited to numbers of data points which are powers of two. It is the belief of the author that the method can be extended for any number of data points. This belief is based on similar types of calculations for the special cases $N = 6$ and $N = 12$. The author also believes that the described method requires, in general, the minimum number of additions and multiplications for both the complex and the real case. This is also based on analyzing the cases for small values of N.

Since there are many variations of the Cooley-Tukey Algorithm, the author hopes that other, more streamlined, variations will be developed for this algorithm. Such streamlining should reduce the problems in the control logic when this algorithm will be programmed for computation by a digital computer. In any case, this algorithm should result in considerable saving in time when extended calculations with a large number of data points are required.

The method for the real case was developed together with Mr. Bertram Goldstone, a friend of the author who works at Raytheon Company, Bedford, Massachusetts.

Non-linear interactive stepwise regression analysis in a time-sharing environment

by LETA ROBERTS EDWIN*

Systronics, Incorporated
Ann Arbor, Michigan

and

ALLAN I. EDWIN

University of Michigan,
Ann Arbor, Michigan

INTRODUCTION

Time-sharing techniques have created an environment that is capable of supporting a sophisticated generation of mathematical programs. This paper discusses one such program for non-linear stepwise regression analysis in an interactive mode. The principal improvements of this implementation over standard regression programs are the inclusion of the programmer in the control loop (thereby allowing on-line modification of the stepwise regression analysis and non-linear term selection procedures) and the ability of the program to modify itself, i.e., learn, from the results of previous passes of the analysis over the data.

Regression analysis is a powerful technique for summarizing (modeling) collections of data. The great speed of digital computers now makes it a practical tool as well. Basically, this form of modeling formulates a regression equation which optimally fits data sets that have typically been gathered via experimentation. Since the resulting equation concisely summarizes the characteristics or patterns of the data, it can be used to predict future (causal) relationships or to describe an entire past process (random), based upon the information sampled.

In order to fully appreciate the flexibility and increased utility of the interactive program described in this paper, one must be aware of the historical development of mathematical programs designed for general use. In order to fill the ever increasing demand for spe-

cial purpose programs not supplied by the manufacturer or developed "in house" by or for the user, it has usually been necessary to utilize existing batch processing programs whose limited flexibility often diminished their usefulness in particular applications.

For example, batch processing implementations of regression analysis have been complex numerical programs that allow minimal interaction (if any) between the man and the machine. Moreover, even the best of these regression analysis programs were hampered by the profusion of control cards needed to communicate with the program, the likelihood and frequency of errors on data cards, and the lengthy turnaround time between runs. Then, once the regression equation had been formulated, a superfluity of printed output had to be scanned in order to evaluate the data and the resulting equation.

Time-sharing, i.e., the ability to allow many users to "simultaneously" share the same computer facilities, has given birth to the "computer utilities." These companies usually provide their subscribers access not only to the computer hardware and programming languages, but also to a library of application programs.

For one such computer utility the author has created a flexible general purpose program for non-linear stepwise regression analysis in the form of an independent subsystem. The design of the program (called COMSTAT) reflects advantages in the following areas:

1. Tailoring the program to the user
2. Implementing the statistical method
3. Integrating the subsystem into the time-sharing environment.

*Formerly with Com-Share, Incorporated, Ann Arbor, Michigan

Tailoring the program to the user

COMSTAT allows the user to become an active participant in the program's activity. At critical points in its execution, the program allows the user, via the teletypewriter, to exert control over the program's flow and output. COMSTAT provides for personalized data storage allotment so that on a small data run the user will not be penalized for the large data capabilities also possible in COMSTAT.

As will be seen, the user can insert or delete terms from the equation as the equation is developing. Thus, a scientist may use COMSTAT as an effective tool for model development. He can examine different models on several runs and gain insight into the inter-relationship of the variables of interest. These powerful abilities are available to the scientist who is not a sophisticated programmer, since the program design stresses a simple execution for the user.

Implementing the statistical method

COMSTAT's designers recognized that the large size of the program and the need to store large quantities of data for an analysis would require a sizeable amount of core storage; additional storage was necessary for a square correlation coefficient matrix needed in the stepwise method. Finally, learning techniques required storage to save the significant results of preceding passes for use in future passes. The program, therefore, has utilized the dynamic memory relocation capability of the Com-Share operating system to handle these memory requirements in a most efficient manner.

Integrating the subsystem into the time-sharing environment

By writing COMSTAT as a subsystem, it was possible to have the program simultaneously accessible to many users of the system. Implied in this decision is that COMSTAT, like all other subsystems, would contain a "command dispatcher" which recognizes the user's command and executes the appropriate block of machine instructions.

The capabilities of the Com-Share System substantially influenced the objectives and design of this program. These capabilities include: the size of core storage, data and program storage and accessibility, the extensive on-line editing capabilities available, the virtual elimination of turnaround time, and the memory relocation capabilities.

Mathematical development

Before discussing the implementation of user interaction and learning procedures, it will be useful to pre-

sent a brief-formulation of the classical stepwise regression method.

Statistics

Multiple regression analysis is a modeling technique principally concerned with finding a mathematical relationship of the form:

$$Y = b_0 + a_1 f_1(x_1, x_2, \dots, x_n) + a_2 f_2(x_1, x_2, \dots, x_n) + \dots + a_n f_n(x_1, x_2, \dots, x_n) \quad (1)$$

where b_0 is a constant term to be included, if necessary. $f_i(x_1, x_2, \dots, x_n)$ represents a general term that can be either linear or non-linear, as the examples below show:

- a) $f_1(x_1, x_2, \dots, x_n) = x_1^2 x_7 x_8^3$ — a non-linear term with highest power two
- b) $f_2(x_1, x_2, \dots, x_n) = x_3 x_7$ — a non-linear term with highest power one
- c) $f_3(x_1, x_2, \dots, x_n) = \text{Ln}(x_2)$ — a non-linear term
- d) $f_4(x_1, x_2, \dots, x_n) = x_7$ — a linear term

The method arrives at an equation of the form of Eq. (1) which gives a "best" fit of the data according to the least squares criterion described herein.

The statistics presented here will be developed for the linear case without loss of generality since a substitution can be made to transform a non-linear term into a linear term.

Example:

$Y = a_1 x_1^2 x_3 + \text{Ln}(x_4) + a_3 x_7$
would be transformed according to:

- a) $z_1 = x_1^2 x_3$
- b) $z_2 = \text{Ln}(x_4)$
- c) $z_3 = x_7$

into the linear regression equation:

$$Y = a_1 z_1 + a_2 z_2 + a_3 z_3 \quad \text{which is a linear equation in the new parameters, } z_1, z_2, \text{ and } z_3.$$

The regression equation aims to minimize the sum of the squared residuals, (the deviations between the observed points and the predicted points). If Y_1, Y_2, \dots, Y_m are observed points for m sets of observations

and Y_1', Y_2', \dots, Y_m' are the estimated or predicted values for use in some predicting equation, then what is to be minimized in order to obtain the regression equation is:

$$\sum_{j=1}^m (Y_j - Y_j')^2$$

where: m is the number of observed points.

This minimization is called a least squares criterion since the quantity to be minimized is the sum of the squared deviation; where $(Y_i - Y_i')^2$ is called the squared deviation of the observed value of the dependent variable, Y , from the estimated value of the dependent variable, Y' , for the i^{th} observation.

This type of analysis is used in causal relationships between several independent variables and one dependent variable, i.e., a causal relationship is developed to predict a future effect based upon gathered past observations.

Random processes can also be described by a regression equation, but there is no guarantee for extending the resulting equation to other data sets of random variables. In this case, the regression equation describes a past relationship based upon a sample of the past and nothing can be said about the future in a random process unless more is known about the nature of the process.

Normality and a common variance are assumed for the data so that the F Test for significance can be applied with reliability in term selection. A 2-tailed F Test is used with the level of significance set by the user.

In order to measure the quality of the regression equation developed, certain statistics are computed at each step in the analysis. The Standard Error of Y , also known as the Standard Error Estimate, is computed to estimate that part of the variance in Y left unexplained by the present regression equation. The Coefficient of Determination is the fraction of the total variance in Y which is accounted for by the present regression equation. The Multiple Correlation Coefficient is the positive square root of the Coefficient of Determination, and ranges between 0 and 1. This statistic is preferred over the Coefficient of Determination in estimating the amount of variance in Y explained by the regression equation.

The stepwise process for constructing the multiple regression equation proceeds by inserting the most significant term not included (if its significance exceeds the criterion set by the user). Then it checks all terms in the equation to insure that they still contribute significantly. If one is found that does not meet the inclusion criterion, it is deleted from the equation. This stepwise process continues until there are no longer any

terms significant enough to include, nor do there remain any included terms that should be deleted.

Note that during the stepwise process terms are often included which are not found in the final equation. This is because a combination of terms may be present at a later step that may account for most of that portion of the variance of the dependent variable attributed to an earlier term. The earlier term's remaining contribution to the variance is then insignificant and so the term is removed from the equation. Figure 1 illustrates an example of this sequence.

Useful statistical definitions in conjunction with a flow diagram are included as an appendix. The appendix, together with the references cited should provide the interested reader with sufficient background for further work in this area.

Interaction with the stepwise process

Because COMSTAT operates in a time-sharing environment it becomes practical to allow on-line interaction since the computer is not forced to wait for a user response, but rather can process other tasks during the delay. Thus at each step in the equation development COMSTAT allows the user, if he desires, to insert or delete terms, i.e., the user can modify the equation at any step.

Since the user has had the benefit of inspecting the intermediate statistical results, he may want to try

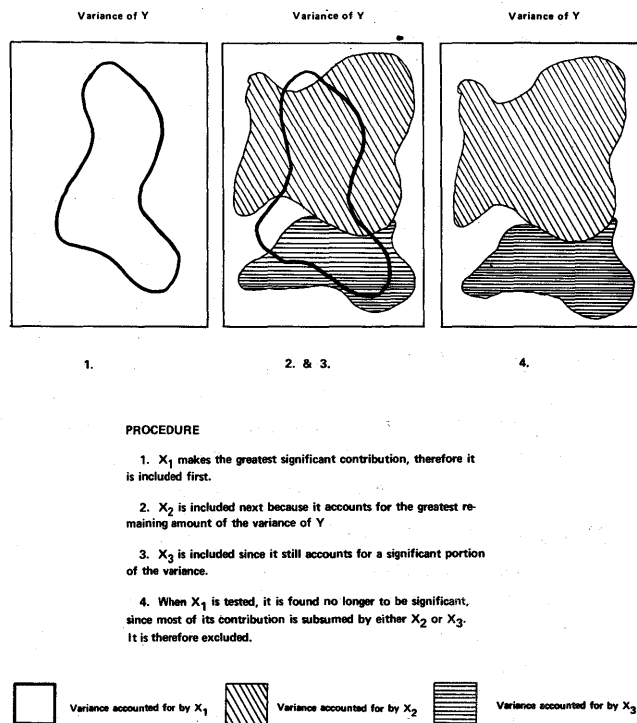


FIGURE 1—Term inclusion-exclusion procedure

some combination of terms not chosen by COMSTAT. Also, since the form of the equation may depend on the order of term selection (particularly in the non-linear case), the ability to steer the course of equation development may allow the user to develop alternate models for his data. It should be noted, however, that if the user adds a term that is not significant on the basis of the existing equation, COMSTAT will reject it. Also, if the user rejects a term that has been chosen, either by COMSTAT or himself, and that term remains significant on the basis of this and subsequent equation modifications, it will be automatically reinserted.

Thus, while the user may modify the development of the equation, he is prevented from including insignificant terms or leaving out terms that must be included. The alternate models developed can be compared and may suggest new and fruitful lines of inquiry.

Learning

In order to accommodate reasonably complex non-linear terms, COMSTAT must have the capacity to select the significant terms from a number of possible non-linear terms that could be too large for analysis in a single run, and perhaps so large as to make complete analysis prohibitively expensive. A method has been incorporated into COMSTAT which will run a series of passes over the data, and analyze sets of non-linear terms chosen from the set of all possible non-linear terms in such a way as to converge on a regression equation. Since this method stores information about the nature and makeup of terms found significant in previous passes for use in the selection of terms for subsequent passes, the method is called learning.

During each pass of regression analysis a new set of non-linear terms, which include the non-linear terms found to be significant in the previous pass, is tried. This insures that the resulting regression equation will be at least as significant as that of the pass before. The additional terms in the set are selected in a manner that takes advantage of knowledge about term characteristics gained from previous passes; terms may also be inserted into the set by the user through interaction.

To explain the learning mechanism more fully, it is necessary to examine the nature of term selection. The relevant characteristics of a non-linear term are:

- 1) the number of interactions of pure variables to be allowed in a general term;
- 2) the actual pure variables which are to interact;
- 3) the functions (powers) of each pure variable selected.

If M were the possible number of pure variables, K the possible number of interactions, and N the number

of possible exponents, then one may write a closed expression for the number of possible non-linear terms:

$$T = \sum_{J=1}^K \binom{M}{J} x N^J$$

$$= \sum_{J=1}^K \left(\frac{M!}{J! (M-J)!} \right) (N^J)$$

In the following example, there are six pure variables. If a maximum of four of these variables, each with an exponent of 1 or -1 , can interact in a term, the number of possible terms would be:

$$T = \frac{6!}{1! 5!} 2^1 + \frac{6!}{2! 4!} 2^2 + \frac{6!}{3! 3!} 2^3 + \frac{6!}{4! 2!} 2^4$$

$$= 6 \times 2 + 15 \times 4 + 20 \times 8 + 15 \times 16$$

$$= 472$$

Since the relevant characteristics listed above form 3 populations each containing an independent finite number of elements, the learning mechanism justifiably utilizes techniques for random sampling from finite populations in generating the non-linear terms used in each pass.

The process for generating a term to be tested in a pass of the regression analysis begins with the selection of the number of interactions that will be included in that term. This selection is from a population that has as members the numerical values of all allowable levels of interaction. Associated with this population is another population which contains as elements the selection probabilities for each corresponding element in the first population.

Example:

Assume the maximum level of interactions is ten. These levels form a set containing 10 elements as shown:

element number:	1	2	3	4	5	6	7	8	9	10
levels of interaction:	1	2	3	4	5	6	7	8	9	10

Set A

For this example let us assume the user has no prior knowledge (as would usually be the case in a first pass over the data). In this case equal probabilities would be assigned to every interaction level, i.e., each element would be assigned a probability of $1/10$ as shown:

element number:	1	2	3	4	5	6	7	8	9	10
probability for corresponding level of interaction:	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$

Set B

Next, a random number between zero and one is generated. Then the term-generating mechanism proceeds to add together the probabilities of elements (starting with the element that corresponds to the lowest level of interaction) until the random number is exceeded. The level of interaction chosen for the term is the level whose probability was the last one added without exceeding the random number.

In the above example, a random number 0.76 was generated. Thus, the probabilities of levels are added until 0.76 is exceeded. The last level added is 7 and so this term will contain seven interactions.

The mechanism then selects the pure variables to be included in the term. This is done in a manner similar to that for selecting the level of interaction. Each variable is assigned a probability of being chosen; with no prior knowledge, the probability for each variable = $\frac{1}{\text{(No. of Variables)}}$. The program generates as many

random numbers as there are interactions, and then utilizes the same summing scheme as outlined above. However, if a variable has already been chosen for inclusion in a term, and it is selected again, the mechanism rejects it and generates a new random number.

Finally, the mechanism must select an exponent for each variable in the term. This procedure is similar to that for the pure variable selection except that here duplication is allowed.

In each pass of the analysis a specified number of terms are selected in this manner. Upon completion of the pass, the results are used to bias the learning arrays, which contain the probabilities associated with the elements of each of the three populations, thereby biasing all future passes.

Example:

In a pass $X_1X_3^{-1}$ was found to be significant and $X_1X_3X_6$ was found to be insignificant. The following probability modifications would be made:

- 1) probability of 2 interactions is increased;
- 2) probability of pure variables X_1 and X_3 are increased;
- 3) probability of exponents 1 and $-\frac{1}{2}$ are increased for the respective variables;
and
- 4) probability of 3 interactions is decreased;

- 5) probability of pure variables X_1 and X_3 and X_6 are decreased;
- 6) probability of exponent 1 is decreased for the respective variables.

The form of these learning arrays and the biasing procedure will be discussed in some detail later in the paper.

COMSTAT's implementation

In programming a subsystem, the programmer must utilize the computer system's inherent capabilities. This section examines the actual implementation of the design. The first part discusses the time-sharing system's contribution to COMSTAT, and the second discusses the software features incorporated into COMSTAT, with examples. Figure 2 summarizes these relationships.

System's capabilities utilized by COMSTAT

The regression analysis subsystem can handle input and output on-line via the teletype or paper tape. However, initially placing the correct input on a disc file can save time for a user during program execution in many input/output tasks. These files may be used for program or data storage and may be in binary or symbolic form.

The storage and retrieval of files created by a user are automatically handled anywhere within the Com-Share System by the EXECUTIVE system. This feature allows a user the flexibility of accessing the same file in any subsystem. Thus, a user is able to create a data file within the text-editor subsystem, QED, and then use it within the regression analysis subsystem.

Ample file protection is provided by the system so that one user may not access another's files unless the file owner has defined the file for public use. This gives a user the flexibility to access files created or used in separate accounts. Practically, this can aid the user by allowing him to be in one account and executing COMSTAT while someone else, in another account, could be simultaneously creating or editing data files to be used in another analysis.

Core memory in the Com-Share System is divided into discrete segments called pages. Each of the 32 pages in memory has a capacity of 2048₁₀ words (cells). Each user is allowed a maximum of eight pages at a time, of which some may belong only to him (private pages) and some may be shared simultaneously by any number of users (shared pages). Although these pages may not be contiguous in core, they appear to be so, to the user, as a result of the virtual memory mapping technique. The pages (private and shared) allocated to the user are

referred to in his map. Referencing for this core memory is provided by pseudo-page numbers.

The dynamic memory relocation capability enables COMSTAT to maneuver the pseudopages within core or in and out of core; the physical process of maneuvering these pseudo pages is referred to hereafter as relabeling.

To make relabeling both practical and economically feasible, pages relabeled out of core are placed on a rapid access device (RAD). This is done because "swap" time between the RAD and core is much faster than when the disc units are accessed.

Since stepwise regression analysis can be functionally broken down into sections with no loss in continuity between sections, the technique lends itself well to implementation on a time-sharing system with dynamic memory relocation capabilities. This statistical analysis requires such capabilities since the quantity of program code alone required to handle this application is sizeable, not to mention the vast quantities of data storage a user may need in a single run.

Also, it is this relabeling technique that makes it possible for the text-editing subsystem, QED, to be incorporated into COMSTAT.

The ability to rerun COMSTAT on the same data, with the same or different subsets of variables, without restarting the entire subsystem is provided by COM-

STAT since it provides for the relabeling of a master data table in and out of core for each run. This allows a user to experiment with the modeling process by specifying different subsets of variables for each run.

In the non-linear case where many passes are required to produce a significant non-linear model, relabeling is essential in handling the added tasks placed on COMSTAT in the term selection process and in saving the learning arrays from pass to pass.

The use of QED, the on-line editing subsystem, within COMSTAT is noteworthy since both are independent subsystems which are usually accessed through the EXECUTIVE system. COMSTAT sets up linking information in a specific area for QED, and then transfers control to an area of code in COMSTAT which relabels COMSTAT's code out of core and onto the RAD, while it relabels QED's program code into core from the RAD. Now, with QED in core, the program location counter continues specifying instructions from the same area as it did before the relabeling had occurred, except that now QED's code will be executed. QED saves the linking information so that it can relabel and restore COMSTAT, without interruption, when QED has completed its job. In addition, QED passes to COMSTAT the edited pages of text or data for direct transfer into COMSTAT's allocated storage area. The effect upon

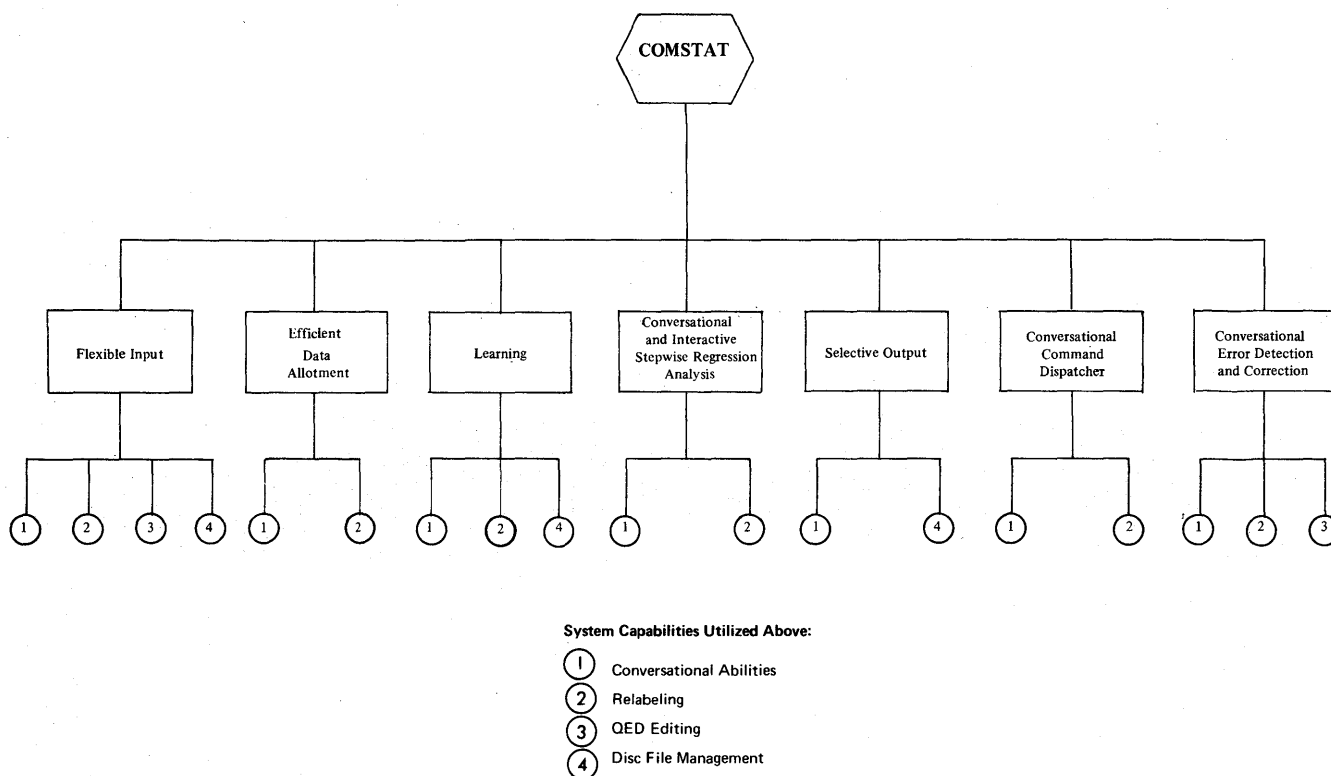


FIGURE 2—Functional representation of COMSTAT with related capabilities

the user at the teletypewriter is only the change of subsystem readiness symbols and commands; response time is continually steady as in normal use.

However, by utilizing QED within COMSTAT, the user is afforded the same range of capabilities as available in the QED subsystem, thereby simplifying and enhancing the input procedure as well as error corrections.

Inherent in a time-sharing environment is the on-line man-machine interaction, "conversational" ability. This permits a wide range of valuable activities as has been discussed.

Software features

In COMSTAT, three pages of shared code are allotted for the program. The program is divided into a control page, an input and learning page, and a statistical analysis and output page.

Depending upon the user's data requirements, COMSTAT will also use 4 to 11 private pages of core storage. The number of data storage pages required can be approximated by:

$$\text{pages} \approx 2 + \left\lceil \frac{(n+1)^2}{1024} \right\rceil + \left\lceil \frac{(n) \times (m)}{1024} \right\rceil$$

where $\lceil \]$ represents the smallest integer not exceeded by the contents.

n = total number of variables

and

m = maximum data set number

Control page

Whenever COMSTAT is used, the control page remains resident in core and commands other pages to be relabeled or swapped into and out of core from the RAD as needed, thus controlling the entire subsystem internally.

This internal control is triggered by the user via a teletype through the use of "commands" to COMSTAT. These commands are translated, in the control page, into internally understood commands by a section of program code referred to as the command dispatcher.

Presently, there are ten such commands. Those commands that are unique to COMSTAT (not standardized system commands), are English words whose meanings are evident.

In a very real sense, this section-by-section command of the program execution in a time-sharing environment gives COMSTAT a "conversational" quality. Furthermore, in the course of executing a command, vital information may be needed from the user. COMSTAT

obtains such information, as needed, through a conversational exchange with the user.

Input and learning page

This page handles all the input activities and learning procedures incorporated into COMSTAT. The full text-editing capabilities of QED are provided in most of the input commands.

Presently, the subsystem is capable of performing analyses on a maximum of 6144₁₀ data entries, or 6 pages of data. The total number of variables in an analysis predetermines the number of data sets allowed. Thus, if an analysis were to be done for 30 variables (29 independent variables plus 1 dependent variable), 153 data sets could be analyzed, for a total of 6120₁₀ data entries.

There are, however, limits to this flexibility in dimensioning. The upper limit presently on the number of variables is 44 (43 independent, 1 dependent). The upper limit on the number of data sets is 1000.

The data requirements for regression analysis vary so widely between users that the large data capacity just described necessitates different use of storage for each user. Conversational dimensioning has been implemented to allow each user to define his specific requirements for data allotment and structure. Likewise, the program sets aside only enough storage to handle the user's data. Thus, the fewer the variables and/or data sets in an analysis, the less storage will be used. This means that fewer page swaps will occur, and connected line time to the computer via the teletypewriter, and central processing unit (CPU) time used will be reduced appropriately. Thus, the user with a small amount of input will not be penalized because the program is also designed to accommodate large data input.

In the input sequence, all errors detected by COMSTAT are communicated to the user, via a printed teletypewriter message. The user is provided with an opportunity at the time of detection to correct the input errors; thereby eliminating the need to correct and resubmit the input, or to completely restart COMSTAT, as would be the case in a batch processing environment.

Also, on this page of code is COMSTAT's learning mechanism which is oriented around 3 learning arrays ("order of interaction" array, "variables entering interaction" array, and the "exponents of the variables" array). It is from these 3 arrays, that COMSTAT develops a specified number of non-linear terms for each pass over the data according to the method outlined.

There are limits set on each learning array. Presently, a limit of 4 is set on the number of variables allowed to interact in any one term. This restricts the "order of

interaction" array to contain 4 entries, (1, 2, 3 and 4 interactions). The maximum number of entries in the "pure variables entering interaction" array is preset by the maximum number of independent variables allowed in any analysis, (43). In the "exponents of the variables" array, the permissible exponents are:

$$\begin{aligned} f(1) &= +1 & f(8) &= -\frac{1}{3} \\ f(2) &= -1 & f(9) &= 3 \\ f(3) &= \frac{1}{2} & f(10) &= -3 \\ f(4) &= -\frac{1}{2} & f(11) &= \frac{1}{4} \\ f(5) &= 2 & f(12) &= -\frac{1}{4} \\ f(6) &= -2 & f(13) &= 4 \\ f(7) &= \frac{1}{3} & f(14) &= -4 \end{aligned}$$

This last array is a doubly dimensioned array of maximum size 602 locations (14 by 43).

In each pass of non-linear regression analysis, a specified number of non-linear terms are chosen by COMSTAT to be tried in that pass. Some of these terms are developed by utilizing the learning arrays and optionally some are specified by the user; the remainder are the terms from the last pass which were found to be significant enough to include in the evolving regression equation.

Terms inserted by the user during the term generation process alter the selection process and will therefore affect the form of the learning arrays for the next pass.

If it is possible to try 10% or more of the possible terms in a single pass, COMSTAT gives the user the option of deciding whether the term generation process is to be done by the learning mechanism or in an ordered manner. This means that whenever the total number of terms could be tried in a "reasonable" number of passes, the learning selection process is often more costly and time consuming than generating all of the terms in an ordered manner and trying a specified number in each pass until all of them have been tested.

The biasing of term characteristics for each term tried in a pass is done by "rewarding" the characteristics of terms found to be significant enough to include

in the present regression equation, and "punishing" the characteristics of insignificant terms.

This "rewarding" is done by multiplying the elements (probabilities) in each learning array which correspond to the respective term characteristics of the significant terms, by the respective learning constants. Likewise, "punishing" term characteristics is done in the same manner as "rewarding," except that each respective array element (probability) is multiplied by the inverse of the learning constants.

<i>Learning Array</i>	<i>Rewarding</i>	<i>Punishing</i>
order of interaction	$\left(\frac{1}{.5}\right)^{1/3}$	$(.5)^{1/3}$
variables entering interaction	$\left(\frac{1}{.5}\right)^{1/3}$	$(.5)^{1/3}$
exponents of variables	$\left(\frac{1}{.5}\right)^{1.5}$	$(.5)^{1.5}$

After all the passes have been made for a non-linear analysis, COMSTAT provides the user with a way to save the learning arrays on a disc file for future use and/or onto the RAD for immediate reuse in a new analysis over the same data.

Statistical analysis and output page

The statistics computed on this page are those outlined in a previous section. In addition, in COMSTAT's stepwise term selection process, the user is provided with the ability to actively interact at each step by specifying a term to include or delete from the present regression equation. This interaction does not harm the stepwise process at all, as explained in the previous part.

Selective statistical output is produced by this page. The user conversationally specifies what output he desires by answering the pertinent questions posed by COMSTAT.

One further point to be noted, which is applicable to all 3 pages of program code, concerns error detection. As part of the natural logic flow of COMSTAT, error traps have been placed strategically throughout the code to assist a user in any difficulties he may encounter while executing. Wherever relevant, detected errors critical to proper program execution are pointed out to the user for correction before processing continues. For example issuing a command out of sequence which will later result in an ambiguous program state is de-

tected whenever possible and noted to the user, failing to supply critical information as input will be flagged as an error, or incorrectly answering any of the conversational COMSTAT questions will be noted followed by the opportunity to reanswer the question.

Execution examples

The ten commands which can be issued to COMSTAT are:

<i>Command</i>	<i>Function</i>
1. TITLE—	Input via QED a title to be used as a heading on output.
2. MAIN—	Input via QED the names of all the variables in the analysis.
3. SUBSET—	Input via QED the names of a specific subset of variables in a run.
4. NUMBER—	Specify the number of variables in the run without naming them. This command does not allow for subsets to be run.
5. DATA—	Input via QED the data sets for all the possible variables.
6. FILE—	Inputs a title, the main variable names, the data, and any number of subsets, from a disc file.
7. RUN—	Begins a run of regression analysis.
8. HELP—	Gives a summary of how to execute COMSTAT and what general conventions are used throughout.
9. QED—	Gives the user complete QED text-editing capabilities.
10. G ^c —	Used to exit from COMSTAT back to EXECUTIVE system.

Any of these commands may be abbreviated by typing 1 or more letters.

In the following examples note that the user-generated type is underlined, while the computer-generated is not. A right parenthesis, [)], in the first position of a line is COMSTAT'S readiness symbol, whereas a star (*) is QED'S readiness symbol. These readiness symbols indicate the respective subsystem's readiness to accept a legal subsystem command from the user. Letters with a superscript c, (for example, G^c) are non-printing and represent a set of characters interpreted by the subsystem to accomplish certain activities. For example, in QED a D^c is interpreted as an indication that the present input or line is complete up to, but not including the D^c.

Figure 3, outlines the simplest complete execution of the subsystem—specifying the three COMSTAT commands NUMBER, DATA and RUN. One can see the

typical COMSTAT-user conversational exchange within the first 2 commands. Two QED commands are (APPEND and EDIT) used internally in the DATA command to input the data and to edit the first line of the input, respectively. The SCO command is a special QED command which is interpreted by QED as the end of all QED tasks and an indication to return to the original subsystem.

The less than sign, (<), on the end of the input is placed there to be interpreted by COMSTAT as an indication that the input sequence is terminated.

The control character, Z^c, within the QED EDIT command is just an indication to QED that the user wishes to copy the line to be edited up to the characters typed immediately following the Z^c, in this instance, up to the first four, (4).

Within the RUN command, many alternative executions can be effected by answering the conversational questions differently. One such execution is outlined in Figure 4. When the intermediate steps are to be printed, the user is given, at each step, the statistics computed.

```

.
.
.
)NUM
NUMBER OF UNNAMED VARIABLES= 3
)DATA
IS THE DATA TO BE WEIGHTED? N
MAXIMUM DATA SET NUMBER= 4
INPUT DATA:
*APPEND
2, 1.3, 1, .998, 3, 1.007, 1.15, 1.4, 7,
.979, 1.2, 1.005< Dc
*1EDIT
2, 1.3, 1, .998, 3, 1.007, 1.15, 1.4, 7,
zc4 2, 1.3, 1, .998, 3, 1.007, 1.15, 1.4, 1,
*SCO
)RUN
.
.
.

```

FIGURE 3—Basic COMSTAT input

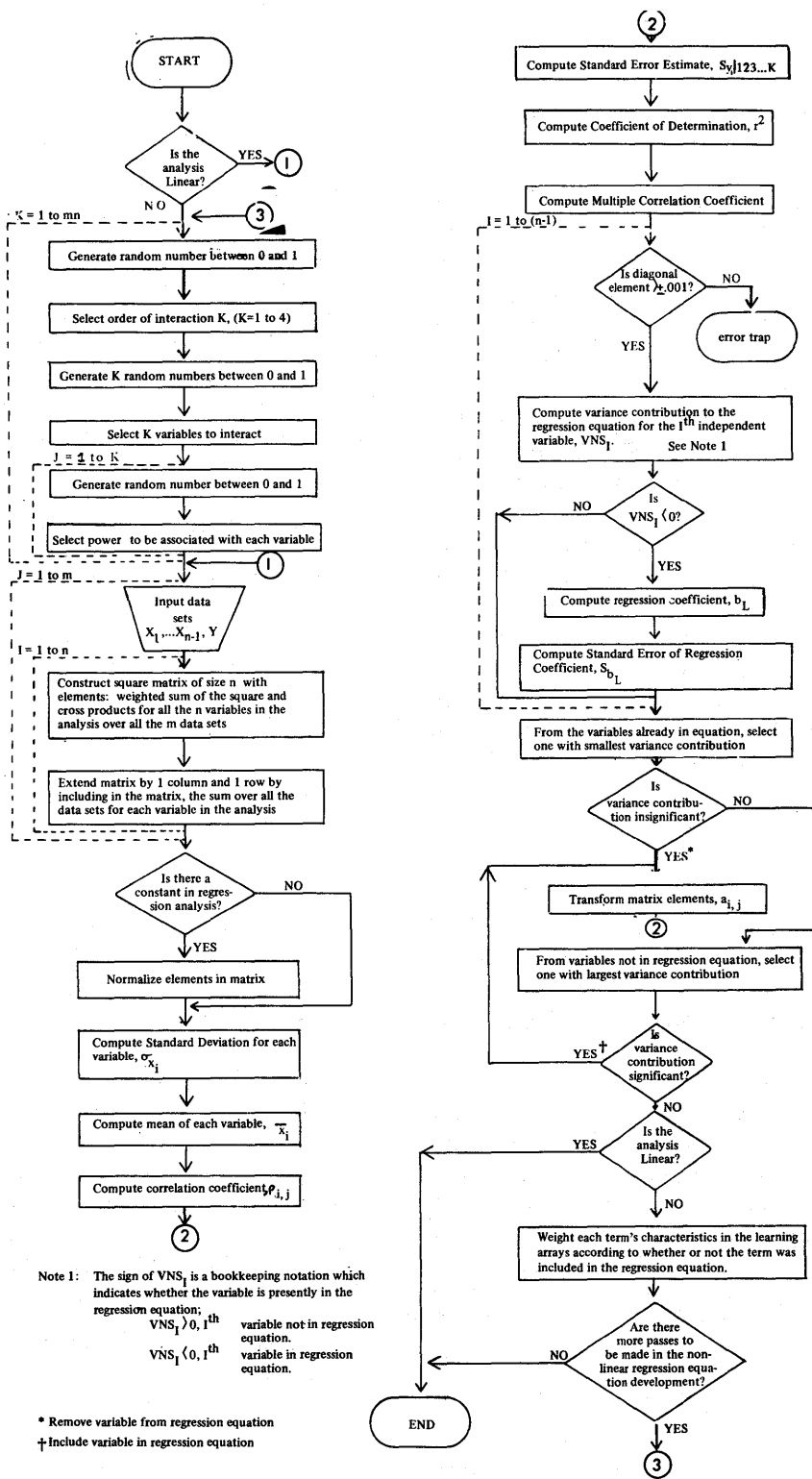
FIGURE 4—Sample interactive execution and output

```

)RUN
IS THIS ANALYSIS LINEAR? Y
IS THERE A CONSTANT TERM IN THE REGRESSION EQUATION? Y
USE STANDARD PROBABILITY LEVELS FOR ENTERING AND DELETING TERMS? Y
PRINT INTERMEDIATE STEPS? Y
USER INTERACTION IN TERM SELECTION? Y
STANDARD ERROR OF Y= 1.223383287
VARIANCE= 1.496666667
STEP NO. 1 :
COMSTATS NEXT CHOICE: INCLUDE A
VARIANCE CONTRIBUTION FOR THIS VARIABLE= 0.996318557
I? N
IN STEP 1 VARIABLE NUMBER 1 (A) ENTERED.
F LEVEL= 811.8977962
STANDARD ERROR OF Y= 0.090911466
MULTIPLE CORRELATION COEFFICIENT= 0.998157581
COEFFICIENT OF DETERMINATION= 0.996318557
CONSTANT TERM= 0.196499239
VARIABLE      COEFFICIENT      STANDARD ERROR OF COEFFICIENT
A              0.95281583       0.040954715
PRINT CORRELATION MATRIX? N
COMSTAT SEES NO FURTHER SIGNIFICANT CHOICES.
I? Y
STEP NO. 2 :
DELETE TERM? N
INCLUDE TERM? Y
NAME OF TERM TO ENTER: B
IN STEP 2 VARIABLE NUMBER 2 (B) ENTERED.
F LEVEL= 10.7681248
STANDARD ERROR OF Y= .089976531
MULTIPLE CORRELATION COEFFICIENT= .998917872
COEFFICIENT OF DETERMINATION= .997214587
CONSTANT TERM= .069542187
VARIABLE      COEFFICIENT      STANDARD ERROR OR COEFFICIENT
A              .732154726       .072342179
B              .147321789       .029432178
PRINT CORRELATION MATRIX? N
STEP NO. 3 :
COMSTATS NEXT CHOICE: DELETE B.
VARIANCE CONTRIBUTION FOR THIS VARIABLE= .56704538
I? N
IN STEP NO. 3 VARIABLE NUMBER 2 (B) DELETED.
F LEVEL= 10.7681248
STANDARD ERROR OF Y= .089976531
MULTIPLE CORRELATION COEFFICIENT= .998917872
COEFFICIENT OF DETERMINATION= .997214589
CONSTANT TERM= .196499239
VARIABLE      COEFFICIENT      STANDARD ERROR OF COEFFICIENT
A              .95281583       .040954715
PRINT CORRELATION MATRIX? N
COMSTAT SEES NO FURTHER SIGNIFICANT CHOICES.
I? N
COMPLETED NO. OF STEPS FOR THIS ANALYSIS= 3
PRINT PREDICTED VS. ACTUAL VALUES? N
RUN ANALYSIS ON THESE DATA SETS WITH NEW SUBSET? N
RUN ANALYSIS ON NEW DATA SETS? N

```

FIGURE 5—Flow diagram for statistical and learning processes



In addition, if the user requested interaction in term selection, he is given an opportunity to interact at each step, as noted in Figure 4 by "I?".

CONCLUSION

Although regression analysis is a powerful and useful tool for constructing mathematical models, many scientists have not utilized the technique. This is perhaps due in part to inadequate statistical knowledge and in part to the complex programming techniques required to employ the batch-processing programs currently available. COMSTAT's statistical capabilities in combination with its conversational execution should interest many experimenters who have not previously used regression analysis techniques.

The availability of intermediate results and the user interaction incorporated into COMSTAT have added a dimension of utility that the experienced user will also find very beneficial in the formation of models. Neither the new user nor the experienced statistician need have much experience before he becomes proficient in the use of COMSTAT.

In summary, we see that the subsystem approach to mathematical programming in a time-sharing environment has great potential. It is hoped that the developmental details presented in this paper will be helpful to programmers in developing future mathematical programs for general use.

APPENDIX

In the interest of completeness, this appendix details the statistical and learning procedures of COMSTAT in the form of a flow chart, Figure 5, with the standard definitions listed separately.

The conventions used throughout the diagram and definitions are:

- (1) n 1 to the number of variables in the analysis. The first $(n-1)$ variables, $(x_1, x_2, x_3, \dots, x_{n-1})$, refer to the independent variables (terms), and the n^{th} variable, x_n , refers to the dependent variable, y .
- (2) m refers to the number of data sets.
- (3) mn refers to the number of non-linear terms to be tried in a pass of regression analysis.
- (4) $x_{i,k} = x_{1,k}, x_{2,k}, \dots, x_{(n-1),k}$ refer to the raw values for the k^{th} data set and the i^{th} variable. $k = 1$ to m , $i = 1$ to n .
- (5) w_k weighting factor for the k^{th} data set. $k = 1$ to m .

- (6) $a_{i,j}$ elements of the correlation coefficient matrix. i and $j = 1$ to n .
- (7) σ_{x_i} the standard deviation of the i^{th} variable. $i = 1$ to n .
- (8) $\bar{x}_i = (x_i \div n)$
 $\bar{x}_n = (x_n \div n)$ the mean value of the i^{th} variable over all the data sets. $i = 1$ to n .
- (9) b_L the regression coefficients for the L variables included in the equation.

The weighting factor, w_k , is assigned to each data set. This allows an experimenter to convey to the program the relative amount of importance (credence), to be given to that data set. Thus data gathered under ideal conditions can be weighted more heavily than data gathered under more trying conditions.

Standard statistical definitions

1. Standard Deviation for i^{th} variable, x_i :

$$\sigma_{x_i} = \sqrt{\frac{m \sum_{t=1}^m (x_{i,t})^2 - (\sum_{t=1}^m x_{i,t})^2}{m(m-1)}} \quad (1)$$

2. Mean of i^{th} variable, x_i :

$$\bar{x}_i = \frac{\sum_{j=1}^m (x_{i,j} \times w_j)}{\sum_{j=1}^m w_j} \quad (2)$$

3. Correlation Coefficient for the i^{th} and j^{th} variables:

$$\rho_{i,j} = \frac{m \sum_{t=1}^m x_{i,t} x_{j,t} - \sum_{t=1}^m x_{i,t} \sum_{t=1}^m x_{j,t}}{\sigma_{x_i} \times \sigma_{x_j}} \quad (3)$$

4. Standard Error Estimate for the k variables (terms) in the regression equation presently:

$$S_{y|123 \dots k} = \sqrt{\frac{1}{(m-k-1)} \sum_{i=1}^m (y_i - y'_i)^2} \quad (4)$$

where: y_i represents the observed points for m sets of observations, and y'_i represents the predicted points from the regression equation.

5. Coefficient of Determination:

$$r^2 = 1 - \left[\frac{(m - k - 1) \times S_y^2 |_{123 \dots k}}{(m - 1) \times S_y^2} \right] \quad (5)$$

where:

$$S_y^2 = \frac{m \sum_{i=1}^m y^2_i - (\sum_{i=1}^m y_i)^2}{m(m - 1)} \quad (6)$$

6. Multiple Correlation Coefficient:

$$= \sqrt{r^2} \quad (7)$$

7. Variance contribution to the regression equation for the i^{th} independent variable:

$$VNS_i = \frac{a_{i,N} \times a_{N,i}}{a_{i,i}} \quad (8)$$

where:

$$N = n \\ i = 1 \text{ to } (n - 1)$$

and $a_{i,j}$ is an element of the correlation coefficient matrix

8. Regression Coefficient:

$$b_L = a_{i,n} \times \frac{\sigma_{zn}}{\sigma_{zi}} \quad (9)$$

where:

$L = 1$ to the number of variables presently included in the regression equation,

$i =$ the index of the variable included,

and $n =$ the dependent variable index.

9. The Standard Error of the Regression Coefficient:

$$S_{bL} = \frac{S_y |_{123 \dots k}}{\sigma_{zi}} \times \sqrt{a_{i,i}} \quad (10)$$

10. Linear transformation on the existing correlation coefficient matrix elements for term to be entered or deleted:

$$a_{i,j} \left\{ \begin{array}{ll} = a_{i,j} - \frac{a_{i,k} \times a_{k,j}}{a_{k,k}} & \text{for: } i \neq k, j \neq k \\ = \frac{a_{k,j}}{a_{k,k}} & \text{for: } i = k, j \neq k \\ = - \frac{a_{i,k}}{a_{k,k}} & \text{for: } i \neq k, j = k \\ = \frac{1}{a_{k,k}} & \text{for: } i = k, j = k \end{array} \right. \quad (11)$$

where k is the index of the variable to be included or deleted from the regression equation.

ACKNOWLEDGMENTS

The authors wish to thank Mrs. Pat Choley and the staff of the Technical Division of Com-Share for their assistance in the preparation and editing of this paper.

BIBLIOGRAPHY

- 1 E L CROW F A DAVIS M W MAXFIELD
Statistics manual
Dover Publications Inc New York 1960 Chapter 6 pp147-194
- 2 J E DALLEMOND
Stepwise regression program in the IBM 704
General Motors Research Staff GMR 199 1958
- 3 M A EFROYMSON
Multiple regression analysis
Mathematical Methods for Digital Computers John Wiley and Sons Inc New York 1962 pp 191-203
- 4 A HALD
Statistical theory with engineering applications
John Wiley and Sons New York 1952
- 5 F B HILDEBRAND
Introduction to numerical analysis
McGraw-Hill Book Company Inc New York 1956
- 6 F H WESTERVELT
Stepwise regression with simple learning on the IBM 704
The University of Michigan Research Institute 1959

Recursive fast Fourier transforms

by G. EPSTEIN

ITT Gilfillan Inc
Los Angeles, California

The development of the Fast Fourier Transform in complex notation has obscured the savings that can be made through the use of recursive properties of trigonometric functions. A disadvantage of the Fast Fourier Transform is that all samples of the function must be stored in memory before processing can start. The computation in the Fast Fourier Transform occurs after the receipt of the last sample of the function; there is no processing of the incoming data prior to this point. Thus if there are N samples of each function, and G different functions (in G "gates" or "channels"), then a total of GN words must be stored in memory.

The recursive approach provides methods whereby these memory requirements may be reduced. These reductions in memory are accompanied by increases in total computation time; the computation time required after the receipt of the last sample, however, decreases. This is accomplished by performing recursive computations on the incoming data as they are being received.

Since savings in memory size must be balanced against increases in total computation time, it is necessary for any given problem to select that recursive method which satisfies the given time and memory requirements. For some problems, it is also important to distribute the computation time in a particular way; e.g., to minimize computation time after receipt of the last sample with bounds on computation time during receipt of samples, as might be the case in a real time spectral analyzer. These considerations are discussed completely for the example case when N is a power of 2.

In January 1958, Gerald Goertzel, in the paper "An Algorithm for the Evaluation of Finite Trigonometric Series" in the American Mathematical Monthly, described an algorithm for reducing the computation time of finite Fourier series. The method which follows makes two essential alterations in this algorithm. First, Goertzel's equations require all samples of the function to be received before the algorithm can be applied. Second, his algorithm does not make use of the

arbitrary factor technique used in the Fast Fourier Transform.

These alterations lead to the following equations. Let the input function be denoted by $f(x)$, so that the spectral components to be obtained are given by

$$(1) \quad A_K = \sum_{x=0}^{N-1} f(x) \cos \frac{2\pi}{N} Kx$$

$$B_K = \sum_{x=0}^{N-1} f(x) \sin \frac{2\pi}{N} Kx,$$

$$K = 0, 1, \dots, N - 1.$$

Let N have a factor p_1 . The method now to be described is called Recursive Method 1. Recursive Method 1 computes recursively on p_1 recursion quantities. Thus this method requires p_1 words of memory. These p_1 recursion quantities are denoted by R_i , $i = 0, 1, \dots, (p_1 - 1)$, and each R_i recurses on the N/p_1 samples $f(p_1x + i)$, $x = 0, 1, \dots, \left(\frac{N}{p_1} - 1\right)$. The recursive equations for each R_i are given by

$$(2) \quad R_i(0) = 0$$

$$R_i(1) = f(i)$$

$$R_i(x) = \left(2 \cos \frac{2\pi p_1 k}{N}\right) R_i(x - 1)$$

$$- R_i(x - 2) + f(p_1x - p_1 + i)$$

$$x = 2, 3, \dots, \left(\frac{N}{p_1} - 1\right); k = 0, 1, \dots, \left(\frac{N}{p_1} - 1\right).$$

Thus in Recursive Method 1 no computation is per-

formed during receipt of the first p_1 samples, and after that one multiplication and two additions are performed for each recursive quantity R_i , $i = 0, 1, \dots, (p_1 - 1)$. This occurs for each value of $k = 0, 1, \dots, \left(\frac{N}{p_1} - 1\right)$.

The values A_K and B_K are computed at the end, after receipt of the last sample, by

$$(3) \quad a_{ki} = \left(\cos \frac{2\pi p_1 k}{N}\right) R_i \left(\frac{N}{p_1} - 1\right) - R_i \left(\frac{N}{p_1} - 2\right) + f(N - p_1 + i)$$

$$b_{ki} = -\left(\sin \frac{2\pi p_1 k}{N}\right) R_i \left(\frac{N}{p_1} - 1\right)$$

and, setting $K = k + \frac{wN}{p_1}$, by

$$(4) \quad A_K = \sum_{i=0}^{p_1-1} \left[a_{ki} \cos \frac{2\pi i K}{N} - b_{ki} \sin \frac{2\pi i K}{N} \right],$$

$$B_K = \sum_{i=0}^{p_1-1} \left[b_{ki} \cos \frac{2\pi i K}{N} + a_{ki} \sin \frac{2\pi i K}{N} \right],$$

$$w = 0, 1, \dots, (p_1 - 1); k = 0, 1, \dots, \left(\frac{N}{p_1} - 1\right).$$

These results follow from the arbitrary factor technique used in the derivation of the Fast Fourier Transform, and from the following recursive property of trinomometric functions:

$$(5) \quad \text{If } S(0) = 0$$

$$S(1) = 1$$

$$S(y) = \left(2 \cos \frac{2\pi p_1 k}{N}\right) S(y - 1) - S(y - 2)$$

$$y = 2, 3, \dots,$$

$$\text{then } \sin \left(\frac{2\pi p_1 k y}{N}\right) = S(y) \sin \left(\frac{2\pi p_1 k}{N}\right)$$

$$\text{and } \cos \left(\frac{2\pi p_1 k y}{N}\right) = S(y) \cos \left(\frac{2\pi p_1 k}{N}\right)$$

-(Sy-1)

The components A_K and B_K for each value of K may be obtained through equations (3) and (4) by $6p_1$ multiplications and $6p_1$ additions. These operations occur after receipt of the last sample of the function, $f(N-1)$.

For each value of K equation (2) requires $N-2p_1$ multiplications and $2(N-2p_1)$ additions. Thus, the total computations using Recursive Method 1 for each value of K are $N + 4p_1$ multiplications and $2(N + p_1)$ additions. If there are G functions to be sampled, the total number of recursion quantities in memory is Gp_1 .

This same technique may be applied if $\left(\frac{N}{p_1}\right)$ has a factor p_2 . This case is called Recursive Method 2, for which there are $p_1 p_2$ recursion quantities stored in memory, and each recursion quantity recurses on $\frac{N}{p_1 p_2}$ samples of the function. The extension of the above equations for this case is obvious. The total number of recursion quantities in memory for Recursive Method 2 increases to $Gp_1 p_2$, and the total computation time decreases from that required by Recursive Method 1, although the computation time required after receipt of the last sample goes up.

Continuation of this process for factors p_1, p_2, \dots generates Recursive Methods 1, 2, ... The selection of the appropriate recursive method depends on the total number of words required for the recursion quantities (given by $Gp_1 p_2$), the total computation time allowable, and the time margin allowed for computation after receipt of the last sample of the function. These considerations will be illustrated completely for the case when N is power of 2, $N = 2^n$.

When $N=2^n$, $p_m = 2$ for $m = 1, 2, \dots, n$. Thus, Recursive Method m for a single function ($G = 1$) requires 2^m recursion quantities to be stored in memory. The computation times for A_K and B_K , $K = 0, 1, \dots, (N-1)$ may be indicated by the number of multiplications that must be performed to obtain these values. These are given in Figure 1, starting with Recursive Method 0, the case where there is no factorization of N ; periodic symmetries are used to obtain the number of multiplications given in this figure.

RECURSIVE METHOD	MAX. NO. OF MULT. DURING RECEIPT OF SAMPLES	MAX. NO. OF MULT. AFTER RECEIPT OF LAST SAMPLE	MAX. NO. OF TOTAL MULTIPLICATIONS
0	$\frac{N^2}{2}$	N	$\frac{N^2}{2} + N$
1	$\frac{N^2}{4}$	$3N$	$\frac{N^2}{4} + 3N$
⋮	⋮	⋮	⋮
m	$\frac{N^2}{2^{m+1}}$	$(2m+1)N$	$\frac{N^2}{2^{m+1}} + (2m+1)N$

FIGURE 1

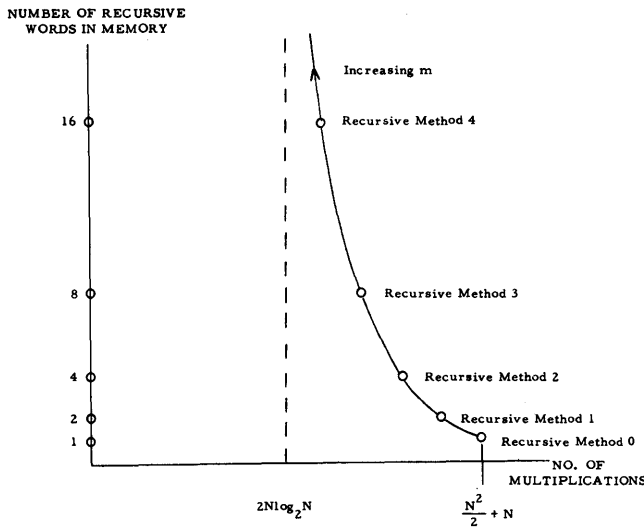


FIGURE 2

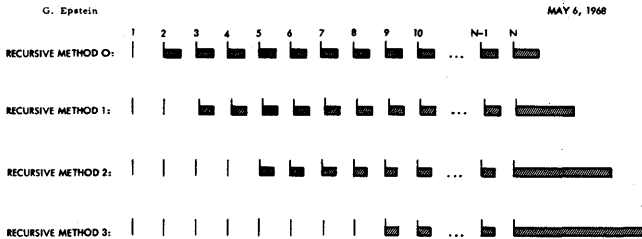


FIGURE 3

It is seen that as m increases, the number of multiplications required after receipt of the last sample increases to the number of multiplications required for the Fast Fourier Transform, while the total number of multiplications, that those required during receipt of the samples, decreases. These results are illustrated graphically in Figure 2.

The distribution of the computation time in these different methods is shown in Figure 3 by the shaded area following the bars, the latter representing the times of sampling.

It should be clear that the actual computation times are less than depicted, due to the existence of trivial multiplications and factorizations. In addition, there are further factorizations in the recursive methods. For example, using the notation

$$(6) \quad {}_kR(x) = 2{}_kR(x-1) \cos k\theta - {}_kR(x-2) + f(x-1) = {}_kA_x - {}_kR(x-2) + f(x-1),$$

expansion and simplification yields

$$(7) \quad {}_sR(n) = \sum_{i=0}^{N-1} a_i(s,n) f(i) + \sum_{i=1}^{[N/4]} \sum_{j=1}^{N-1} {}_iA_j c_{ij}(s,n)$$

for each s in the range $\frac{N}{4} \leq s < \frac{N}{2}$, where $a_i(s,n)$ and $c_{ij}(s,n)$ are integers or zero.

Since (7) does not require any full-length multiplications, it follows that the number of multiplications required during receipt of the input samples may be halved at the cost of increased additions and shifts after receipt of the last sample. That is, $R_i(N-1)$ and $R_i(N-2)$ in (3) may be obtained directly through (7) for each $s \geq \frac{N}{4}$.

ACKNOWLEDGMENT

In addition to the references listed below, this work was founded on discussions with John S. Bailey. The contribution of other personnel at ITT Gilfillan and ITT Electro-Physics Laboratories is acknowledged, and also the fine assistance of Linda S. Saiki in transcribing.

BIBLIOGRAPHY

- 1 G GOERTZEL
An algorithm for the evaluation of finite trigonometric series
American Math Monthly January 1958 p 34
- 2 J W COOLEY J W TUKEY
An algorithm for the machine calculation of complex Fourier series
Math of Comput Volume 19 April 1965 p 297-301

A file management system for a large corporate information system data bank

by HONIEN LIU

Pacific Gas and Electric Company
San Francisco, California

INTRODUCTION

A corporate information system has several characteristics that distinguish it from a conventional "batch-oriented" data processing system.

1. It has a well-planned integrated data base which is logically organized to facilitate the natural information flow in a corporation. By contrast, a conventional batch system is quite often fragmented into "information islands" with rigid boundaries, each designed for a single function.
2. It is a real life computerized model of a corporation. This model will reflect the corporation's performance on a continuous basis. If management wishes to evaluate any segment of the organization, they may obtain an up-to-date picture of that segment upon request.
A conventional batch system usually is a discontinuous record-keeping mechanism, designed for a predetermined cut-off date, with the tabulated results reporting past performance of those organizational segments which had been previously defined.
3. It is a flexible system designed to serve a corporation at all levels of information processing. For instance:

- High-volume, detailed business transactions related to every day business operations will be processed as scheduled.
- Summary reports of management control information (required to speed up control action) will be available on demand.
- Operating standards may be computer monitored with notice of nonstandard performance being swift enough to per-

mit the proper management level to take prompt corrective action.

- Strategic planning information assembled to aid top management in its policy decision making will be available on special request.
- Computer service will be extended to the entire company by telecommunication network thus improving the efficiency and accuracy of the information flow in the organization.

The key factors in the implementation of a corporate information system are the establishment of a corporate data bank and the design of a file management system that will provide for efficient and effective use of this data bank. The remainder of this paper describes the requirements, strategy, design, and implementation of such a file management system.

System requirements

The following requirements are essential for an effective file management system of a corporate information system.

Centralized control of data definitions

Because the elements of the data bank of a corporate information system interact upon each other, a logical intertie must be built to ensure:

- **Consistency of Data Definitions:**
That is, agreement on name, meanings, usage, timing, and scope of each data element definition.
- **A piece of Information Should Be Recorded Only Once:**
This not only allows for efficiency of opera-

tion and reduction of storage requirements, but also is necessary for maintenance simplicity. It will guarantee that once a piece of data is updated, all the users will be provided with identical and current information.

Independence of data from individual programs

The contents of the data bank change as the needs of an organization change. The same data bank will be used by hundreds or probably thousands of programs; the reprogramming cost will be prohibitive if the data and programs are dependent upon each other.

File protection and security

- Security of a File at the Field or Element Level:
Since the data bank is shared among many users, certain sensitive data should be available only to authorized personnel.
- Exclusive Control Over Concurrent Updating:
During concurrent updating of the data bank in the multiprogramming mode, it is possible to destroy previously updated information, thus some preventive procedures must be designed.
- Checkpoint and Restart Facilities:
A large data base usually has long file maintenance runs and it is quite possible that the system will encounter certain fatal input-output errors causing the system to abort the job. An adequate checkpoint and restart facility must be provided to enable the job to skip the error record and achieve normal completion.
For the on-line real-time situation, the system should have a facility to degrade gracefully and still give partial service in the event part of the data bank has been damaged.

Effective use of file space

The data bank of a large corporation may very well require ten to forty billion characters of storage. If we attempt to store all these data on a direct access device with reasonable access speed and reliability, the price could be prohibitive (see *Table 1*).

If we cannot resolve the mass storage space

management problem, economically and technically, we cannot build the kind of data bank we

	IBM-2311 DISK DRIVE	IBM-2314 DISK ARRAY	IBM-2321 DATA CELL
Number of units required for a ten billion character data base*	2,000	60	34
Monthly rental in dollars	\$1,200,000	\$360,000	\$120,000
Sequential loading time in hours	125	30	340

Approximate direct access device requirement for a ten billion character data base using conventional programming technique.

*Assumed net usable capacity after considering overhead of addressing, indexing, etc.

TABLE 1

need. One of our solutions has been to design several data compaction routines which allow us to substantially reduce the amount of storage space required for these data. In *Table 2* an example is given of the amount of savings that can be obtained with data compaction for P. G. and E.'s customer information master file using 8-drive disk units. The storage of the unpacked data would require the equivalent of 90 IBM-2314's. Besides the impracticality of housing these many disk units, their annual rental would approximate \$6,500,000. With well designed data compaction software, we can cut costs to an annual rental of a little over \$500,000; about \$6,000,000 savings in annual rental. However, the question may be raised whether there wouldn't be an offsetting cost in the system time that would be used to compact the data. An examination of *Table 2* shows that it takes 45 hours to sequentially load a file of unpacked data and only four hours to load a file of compacted data; a difference of 41 hours. Our compaction routines will use considerably less time than this, and hence we will receive an overall time advantage as well as reduce storage costs.

A facility to capture management planning and control information

Certain data elements in the corporate data bank have the characteristics of controlling the operation of business activity. Others provide vital information for strategic planning; for instance, information on customer orders and customer usage are used in forecasting demands, in

Record Type	Average Record Size (Bytes)	Storage Requirements (Billions of Bytes) (1)	Number of 2314 Units (2)	Sequential Loading Time (Hours) (3)	Approximate Monthly Rental (4)
Fixed, No Compaction	4500	15.8	90	45	\$540,000
Variable Increments, No Compaction	700	2.5	15	7.5	90,000
Variable Increments, Compaction	400	1.4	8	4	48,000

Notes:

1. Storage requirements based on 3.5 million customers.
2. Storage based on 75% of capacity or 175 million bytes per 2314.
3. Assume 30 minutes for eight drives.
4. Cost per 2314 approximately \$6,000 per month.

TABLE 2

designing facilities, and in constructing plants.

Since planning and control information requirements change from time to time, the system must identify such information and establish adequate relationships among them so that proper summary reports can be prepared without tedious programming efforts.

Provide usage statistics on each data element

This information will detect dead fields or low usage fields versus high activity fields and thus allow for improving the usefulness of the data bank.

Efficient and easy applications programming

The advantage gained by coding in high level languages such as COBOL or PL/1 should be considered for application programming.

Strategy of the system

Separate the input/output (I/O) functions from the application programs

This will reduce the effort required by the application programmer in programming, debugging, and testing the input/output routines as well as providing several other advantages.

Figure 1 shows a conventional program structure in a multiprogramming environment where each application program has its own data and file definitions, I/O buffers, and I/O routines.

If we change our concept in structuring the program to that shown in Figure 2 and centralize all four I/O components in one location, we will achieve the following advantages:

Multitasking:

In Figure 1 the program contains I/O buffers and therefore cannot be re-entrant. Once a task is initialized for this program, variable data contained in the program belong to this task. If a second task begins before the first task is completed, it will destroy the data required by the first task. However, in Figure 2 there are no variable

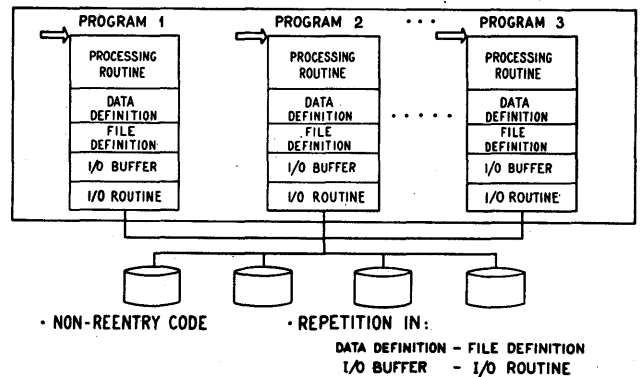


FIGURE 1—Conventional program structure in multi-programming environment

data within the program, and the coding in the program does not alter any portion of the program (i.e., the program is a read-only copy). These programs now may become reentrant modules. Any number of tasks can be initialized in the module without waiting for completion of the previous tasks.

Reduction in Core Memory Requirements:

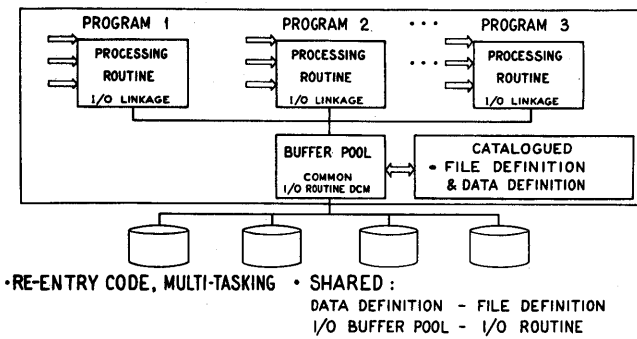


FIGURE 2—Program structure with centralized I/O function in multiprogramming environment

I/O routines can be identical if all the programs share the same data bank.

I/O buffers can be shared by different files.

In general, these four components will represent a high percentage of the core requirements for the application program.

• Speed-Up Program Loading:

In the on-line real-time environment, the program loading activity consumes a good percentage of the time available. It is obviously not economical to load a message processing program of 80,000 bytes merely to process a short message of 20 bytes. Since the sizes of the applications programs are greatly reduced, more programs can be allowed to reside in the core, thus reducing roll-out roll-in time.

The size of the program library will also be reduced thus allowing it to be concentrated in very few cylinders on the disk, resulting in reduced disk arm movement and less I/O time.

• Central Control:

It is necessary to have centralized I/O in order to achieve consistency of data definition, protection/security, usage statistics, mass storage space management, and concurrent updating.

• Flexibility:

Reconstruction and redesign of a file

will have very limited impact on most of the application programs.

There are, however, some disadvantages in separating I/O from the application programs:

- Adequate Software is Not Available: System programs to handle centralized I/O functions are not supplied as standard packages by the computer manufacturer at the present time. Some uncommitted trial packages exist; however, and in general, the performance of these packages is not satisfactory for ultra-high volume systems.

- Nonconventional Usage of High Level Language:

The normal use of high level languages such as COBOL requires that the I/O, file, and data handling routines be within the program. Some programmers accustomed to this method of programming may feel strange if they must change their ways. Documentation, programmer orientation, and training in the new programming concepts are necessary to achieve optimum results.

- Increased CPU Overhead:

The data description compiled within the application program has one definite advantage in that the code is already generated within the program. No further interpretation process is required for the I/O function. Additional CPU time may be required for the centralized I/O routine to interpret the record for the application program. However, the amount of time saved in program loading should more than compensate for this additional CPU interpretation time.

Separate tables and search routines from the application program

To demonstrate the concept of centralized tables and table search routines, compare the following figures:

- Figure 3 shows the fixed table concept with the tables and search routines coded within the applications programs. Its major disadvantages are:

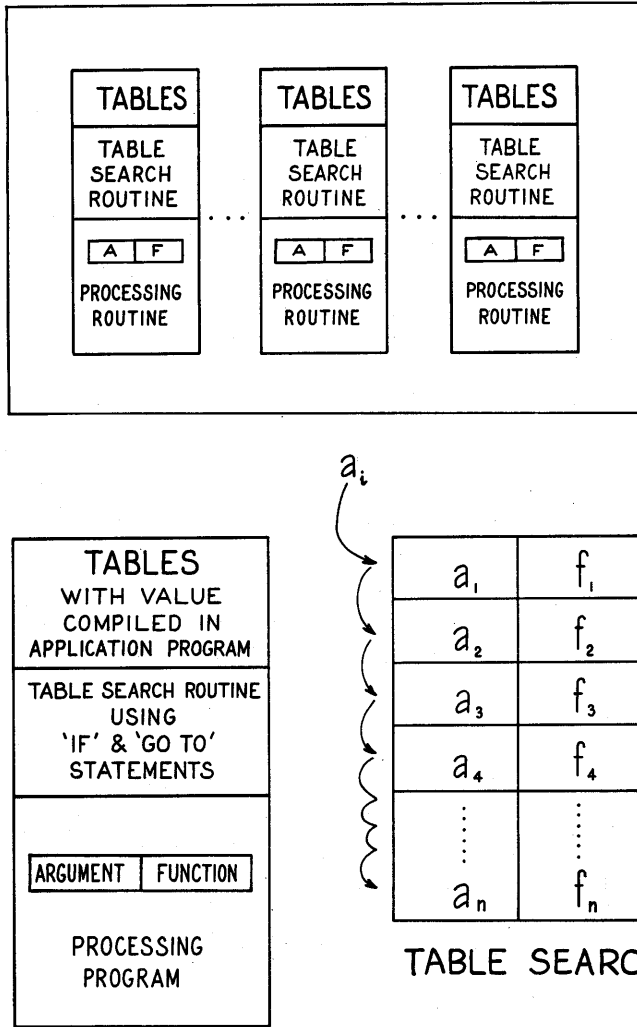


FIGURE 3—Fixed table

- Duplication of tables and search routines in the core.
- Inefficient table search routines.
- Inflexibility of tables; that is, when a table is changed, the program must be recoded and recompiled.

Figure 4 illustrates the dynamic table building concept. Here table space is reserved within the application program; a Build-Table-Routine (BTABLE) is called by the application program and linkage edited with the program. During program execution time, the BTABLE routine will read the table from the on-line code list file. The improvement of this approach over the fixed table concept is that the table is relatively independent of the program. When the table must be modified, if the reserved space

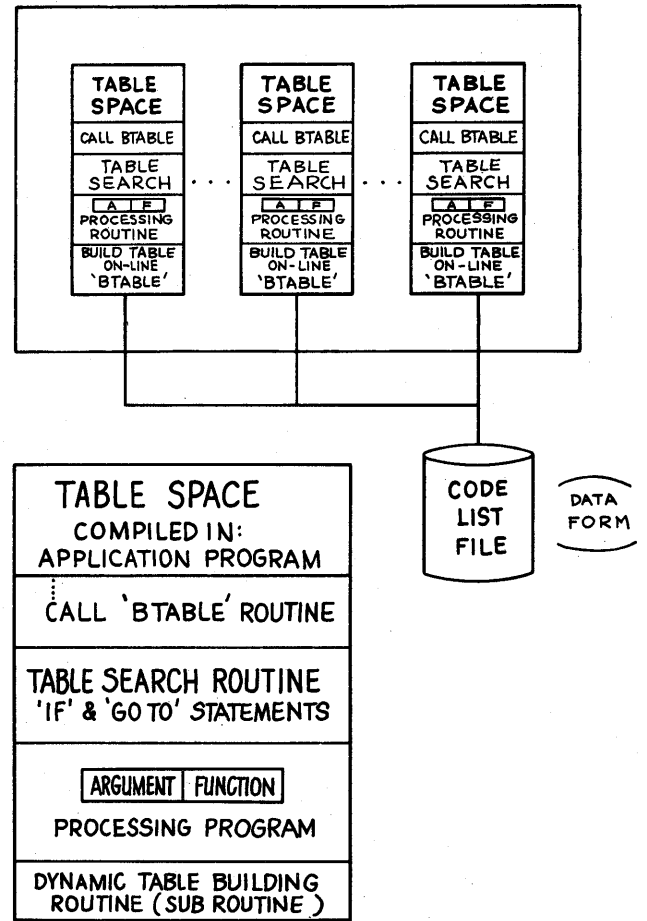


FIGURE 4—Dynamic table building

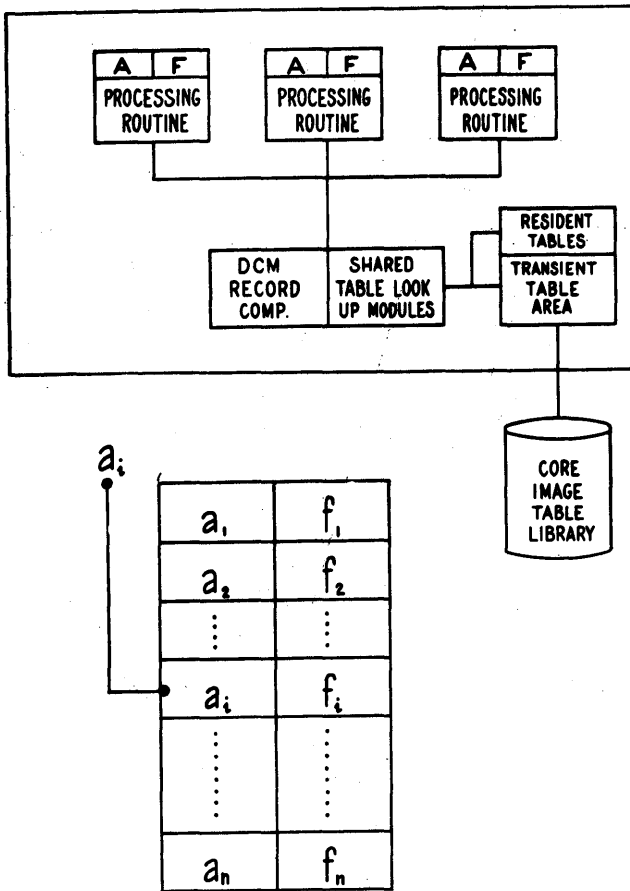
- is not violated, the program is not affected.
- *Figure 5* shows the centralized table concept. Here the tables as well as the table search programs are separated from application programs and collected into one centrally controlled area, offering the following advantages:

Savings of Core Storage:

Core storage utilization is optimized by loading only one copy of the search module or any given table in a multi-programming environment.

Optimum Resource Allocation:

Resource allocation is optimized by dividing the tables into two groups—a high activity group, which resides in the main storage (eliminates frequent loading of these tables from the mass storage device); and a low activity group, which is loaded into



DIRECT TABLE LOOK-UP
(BASE DISPLACEMENT)

FIGURE 5—PG and E centralized table system

a transient area of the main storage only when requested.

Prevent the Creation of Redundant Coding Schemes:

The centralized control of code tables prevents each application project from designing its own coding scheme for the same subject; e.g., an accounting program might use a different construction job code than an engineering program although they both refer to the same physical job. It also guarantees the consistency of code definition since there is only one copy of each table in the system. When this copy is updated, all the users get the latest version of that table.

Efficiency in Searching:

Most high-level languages, such as

COBOL, are not able to use the base and displacement facility of the third generation computer. However, if we can load the beginning address of a table in the base register and then load the displacement of the argument entry into the index register, we can immediately pick up the function and argument from the entry without a search. For large tables, this means a large saving in CPU time.

Ease of Maintenance:

It is obvious that one must store the displacement instead of the argument in the data bank to obtain the above efficiency. This adds another advantage to this approach; if the tables are updated, the data base can remain unchanged.

System design

To design a software package satisfying all the requirements stated previously is quite challenging. The name of the game, however, is not just to meet these requirements but rather to satisfy them in a simple and efficient way. The package we have designed is logically subdivided into four parts:

Data control manager (DCM)

A real-time file management control program:

- It is the central clearing house of all the input/output requests from all the tasks in a multiprogramming environment.
- It performs the centralized input/output buffer pool management.
- It is the intercept point for file protection, security clearance, and exclusive control for concurrent updating.
- It opens and closes all the on-line files, keeps the file and table directories, and captures usage statistics on each data element as well as all the entries on certain resident code tables.
- It interfaces with the Mass Storage Space Manager and the Dynamic Table Manager through the Record Structure Descriptor Table.

Mass storage space manager

A group of data compaction routines driven by the Record Structure Descriptor Table (RSDT) to serve the following functions:

- Centralized Data Definition:

Interpret the contents of the data base to the application programs thus eliminating data descriptions and making the data independent of application programs.

- Provide Various Internal Data Structures to the Data Base:

That is, variable length records each constructed by a combination of the following types of increments.

Fixed length increment with increment bit map:

If increments are not used in certain records, they are not stored in the file. The presence or absence of an increment is signified by a bit switch in the increment "bit map" for that particular record. For instance, if there are four increments in a maximum record, and only the first, second, and fourth are present for a particular record, its bit map would be 1101. The three "1's" indicate increments that are present and the "0" indicates an absent increment (*Figure 6*).

Variable length increment with field bit map:

Similar technique applied to field level.

Variable length increment with field counter:

A group of fields of the same length with a counter to tell how many of

these fields are actually present in this increment.

Utilizing nesting logic, one can construct a great number of different internal record structures; one special example would be the tree structure (*Figure 11*).

- Data Compression:

Information units should be precisely defined to the bit level; thus, a binary variable (switch) should only occupy one bit instead of one byte (eight bits).

Absent data elements should not occupy storage space with blanks.

The variable length record concept can be extended to the field level; e.g., a name field in the personnel file is usually defined as a fixed length of 30 characters so that it can handle the longest name in the file; thus, a short name will occupy a 30 character space padded with unnecessary blanks.

A more efficient coding structure can be utilized to increase the information density; e.g., if a long data field of alphanumeric information is stored in the eight bit EBCDIC code, by simply changing it to a six bit BCD coding scheme 25 percent storage space can be saved.

Tailored routines can be designed to compact frequently occurring data elements for greater saving; e.g., a six-digit "date" can be stored in a two-byte field without loss of information.

Store numerical information in binary form; e.g., a four byte binary field can accommodate nine digits of decimal data.

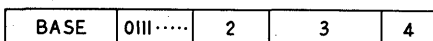
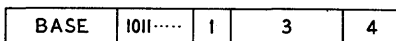
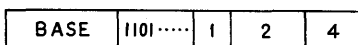
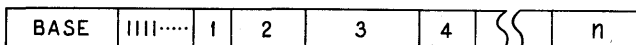


FIGURE 6—Incremental structure

Dynamic table manager

Routines are provided that will load, delete, search, and maintain a centralized table facility for all of the application programs. These routines serve the following functions:

- Reduce the number of tables required since all programs desiring similar information will use the same table.
- Facilitate code table maintenance. If a code

is changed, added, or deleted, only one change need be made to the code table rather than having to change several tables in several applications programs.

- Relieve the applications programmer of writing table search routines, some of which are quite tedious.
- Centralize control of code tables to insure that different applications don't design different coding schemes for the same subject matter.

File maintenance interface

Since batch processing and on-line message processing programs share the same data bank, certain facilities such as the Mass Storage Space Manager and the Dynamic Table Manager must be available for use by a file maintenance run.

The heart of the file management system presented in this paper is the Record Structure Descriptor Table (Figure 7). This table is divided into two major sections, one containing the data coordinates and the other containing control information:

1. *The Data Coordinate Section Shows the Following Attributes:*
 - Before Expansion:

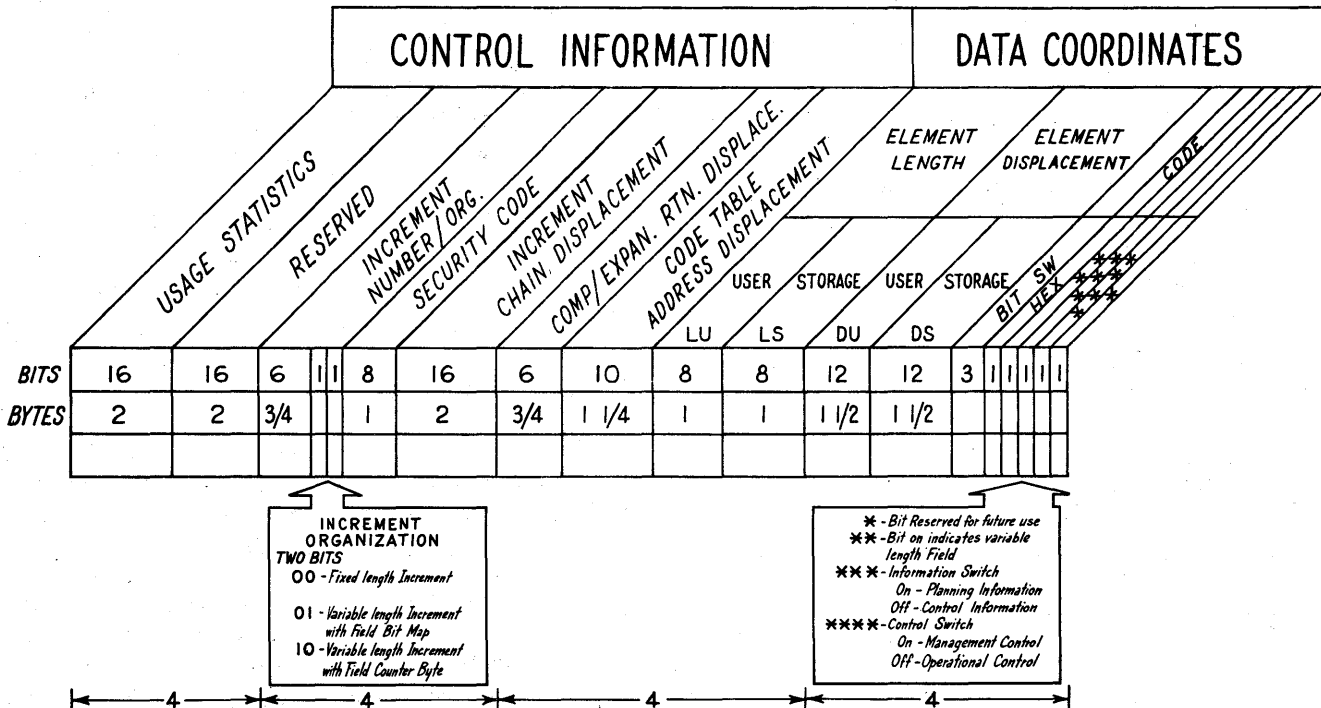
The relative location of the data element when the record is residing in file storage (Ds, Ls) in the compacted form (Figure 8).

- After Expansion:
The relative location of the data element after the record has been expanded and moved into the user area (Du, Lu).
- Units of Information:
The "bit switch" and the "hex" attributes define the units of information at the bit level.
- Variable Length Element Switch:
When data elements are stored in the variable length form, this switch will be turned on.

2. *The Control Information Section Contains:*

- Usage Statistics for Each On-Line Data Element:
Periodically, a system service program will dump these accumulated statistics on a storage device, such as magnetic tape, for future analysis.
- A Security Code Attached to Each Data Element:
Each on-line request for access to the element will have its security code checked

FIGURE 7—Record structure descriptor table



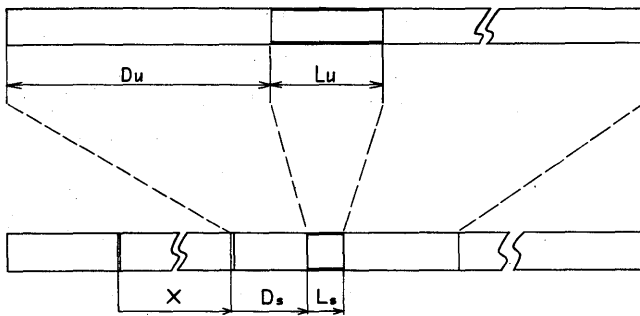


FIGURE 8—Record compaction/expansion

against that of the data element before access is allowed.

- **Compaction, Expansion, or Table Search Routine Displacement:**

This attribute serves as a pointer to the correct routine to interpret the data element (Figure 9).

- **Code Table Address Displacement:**

If the data element is a code that requires interpretation, this attribute will locate the correct code table for the table search routine. If the table is not in the main storage, the control program will load the table into the core. If it is a low activity table, the system will delete the table after the search function is completed thus freeing the core for other usage.

- **Information Switch:**

If a data element is identified as management planning information, this bit switch will be turned on.

- **Control Switch:**

If a data element is identified as management control information, this bit switch will be turned on.

- **The increment number, the increment organization, and the increment chain displacement attributes will provide the ability to access information in the following forms from the data base:**

A record:

If the user desires, he may have the entire record expanded and presented to him.

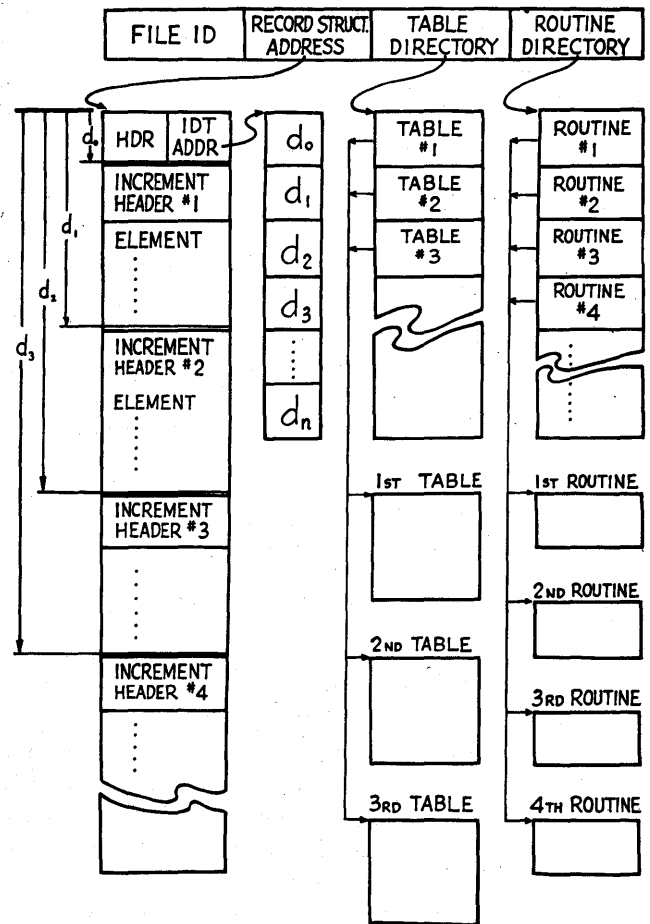


FIGURE 9—File directory entry

An increment:

A logical group of elements within the record. The increment chain displacement attribute indicates the location of the next element in the increment and thus provides the ability to supply a complete logical increment to the user.

Increment by circular ring:

Since the increment chain displacement is constructed in a circular ring fashion (see Figure 10), a user can obtain a logical increment by identifying the class of information desired in the lead column of his report. For example, a user requests a listing of an employment directory from the data base; he may choose to have his employees listed first by name or by telephone number, or by social security number, or by address, or by de-

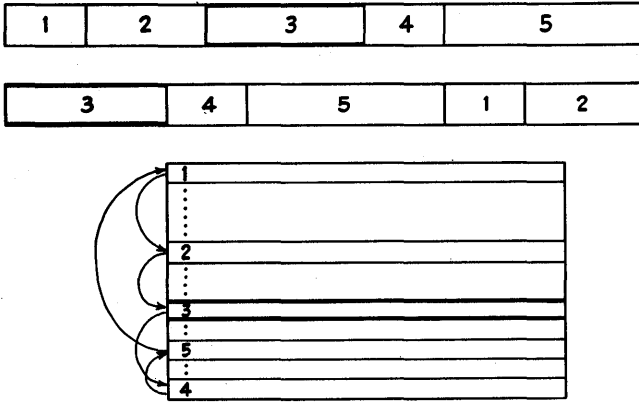


FIGURE 10—Increment chain and circular ring

partment number, etc. without additional programming.

Branch of a tree structure:

By combining different increment organizations in one record, a limited tree structure can be formed with a record (Figure 11); e.g., an inventory record using the material code as the major key has a base increment describing the characteristics of the item followed by a certain number of increments which describe those stores that carry this item, and each store increment in turn will be followed by a certain number of increments which will describe all the local vendor information concerning this item. Furthermore, each vendor increment can have a number of volume and price increments. If we consider this as an internal information tree, a branch of the tree may appear as: list the price information

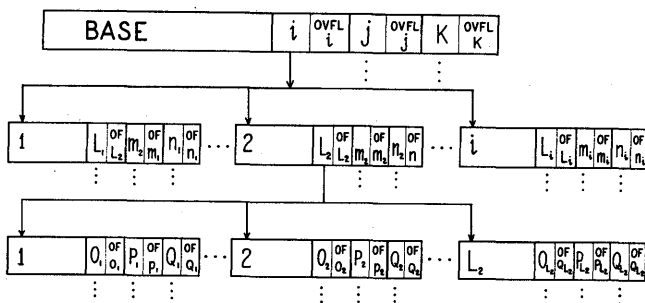


FIGURE 11—Tree structure

on material item (XYZ), store (i), vendor (1), volume (p).

An element:

A unique element control number assigned to each field when the RSDT is created allows the application program to request individual elements and thus free the program from being cluttered with elements that it does not use.

Implementation of the system

The system program package has been written in IBM-360 Operating System Assembly Language (ALC).

1. It is fully interfaced with the IBM-360 operating system, under either MVT (multiprogramming with variable number of tasks) or MFT (multiprogramming with fixed number of tasks). The package is not tied to any particular release of O/S; hence, if a new version is released, there should be little or no effect on this package.
2. The data bank management package takes full advantage of the existing operating system facilities.
3. It is intended to interface with all the operating system supported languages (COBOL interface will be implemented first).
4. The entire package has been designed to be dynamic in nature; that is, all programs are load modules. They are not linkage edited into the application program; thus, the package may be redesigned and improved without any appreciable effect on the application programs.
5. The entire package has been programmed in re-entrant code.
6. The system has been coded in a modular fashion. Each routine was individually coded, tested in detail, subgrouped, and finally all routines were combined together.

To increase the efficiency of the package, the following measures were taken:

Unnecessary saving of registers and store register operations were avoided by branching between modules rather than using subroutine calls.

All modules were carefully ex-

amined to the bit level to insure tight coding. The total coding of the package may use less than one percent of the total main core memory.

7. The hardware anticipated over the next several years includes two large central processors with a million bytes of main memory, supported by smaller satellite computers and a score of multi-drive disk storage units. The system is being designed to support several hundred terminals, most of which are expected to be high speed CRT display units.

APPENDIX 1

Record structure descriptor table

Control information

1. Reserved area—A two byte reserved area for control function; e.g., use it as 16 bit switches each associated with a management report.
2. Usage Statistics
 - A. A two byte binary counter is used to maintain a count of the number of accesses made to each particular element.
 - B. When maximum value (65,535) is exceeded, an overflow routine stores the element control number in an overflow area in the record structure header, adds one to the two byte binary overflow area, and clears the original counter.
3. Increment Number/Organization
 - A. Indicates which increment within a record contains the element described in the table entry.
 - B. Increment number occupies six high order bits (maximum value - 63).
 - C. Increment organization: Two low order bits.
 - 1.) 00 — Indicates fixed length increment.
 - 2.) 01 — Variable length increment with field bit map. The presence or absence of certain fields in an increment depends on the on or off of the related bit switch in the bit map.
 - 3.) 10 — Variable length increment with a field counter byte followed

by a group of fields of same length.

4. Security Code
 - A. Consists of a security or protection code for the element described in this entry.
 - B. Designed to prevent certain information in the files from being obtained by unauthorized individuals.
 - C. Occupies one byte (maximum value—255).
5. Increment Chain Displacement
 - A. Provides a pointer to the next logical element description for the record (or increment).
 - B. Provides record structure independence from data changes in description and/or additions or deletions of elements by allowing a nonsequential physical record structure table organization.
 - C. The chains are constructed in a circular ring fashion; i.e., the last element pointed to the beginning element of an increment, or in the case of single level structure, the last field pointed to the beginning field of a record.
 - D. Occupies two bytes.
6. Compaction/Expansion or Table Look-Up Routine Displacement
 - A. Used by the DCM modules to determine the correct compaction/expansion or table look-up module to be invoked.
 - B. Occupies six bits, maximum 64 routines.
7. Code Table Address Displacement
 - A. Points to an entry in a table directory which identifies the table required to convert the code in the record to the value obtained from the table.
 - B. Occupies ten bits.

Data coordinates

1. Element Length
 - A. Length User — Indicates the length of the element when expanded to user format. Occupies one byte.
 - B. Length Storage — Indicates the length of the element in its compacted format as stored upon the DASD. Occupies one byte.
2. Element Displacement
 - A. User Displacement — Indicates the relative location of the element in relation to the beginning of the record (or increment) as expanded to user format. Oc-

- copies twelve bits.
- B. Storage Displacement — Indicates the relative location of the element in relation to the beginning of the record (or increment) as compacted to storage format. Occupies twelve bits.
 3. Bit Displacement
 - A. Occupies three bits which indicate the relative displacement within the byte of the particular bit switch.
 4. Hex Switch Indicator
 - A. Indicates if data normally contained in $\frac{1}{2}$ byte has been combined at compaction time with similar data.
 - B. Occupies one bit.
 5. Variable Length Field Indicator
 - A. If on identifies the element as a variable length field or if off as a fixed length field.
 - B. Occupies one bit.
 6. Information Switch — Occupies one bit.
 - A. If on identifies the data in this element as *planning information*.
 - B. If off identifies the data in this element as *control information*.
 7. Control Switch — Occupies one bit.
 - A. If on — *management control*.
 - B. If off — *operational control*.
 8. The last bit is not used at this time, but is held in reserve.

ACKNOWLEDGMENT

The author wishes to thank Mr. J. R. Kleespies

for his encouragement and support; Messrs. J. O. Mohn, R. J. Wolfenden, C. E. Jones, and R. St. Germain for their dedicated efforts in design and programming; Mr. W. S. Peck for his careful editing; Mmes. L. Andrews and L. Fiore for their typing; Mr. R. P. Kovach for many stimulating discussions; and Messrs. J. W. Nixon and F. J. Thomason of Haskins & Sells for their invaluable assistance and advice.

REFERENCES

- 1 R V HEAD
Management information systems: A critical appraisal
Datamation May 1967
- 2 A METAXIDES
Data base management
ACM SICBDP December 28 1967
- 3 P J DIXON J SABLE
DM-1—A generalized data management system
Spring Joint Computer Conference 1967
- 4 COBOL extension to handle data base
CODASYL Data Base Task Group January 1968
- 5 *The corporate data file*
EDP Analyzer November 1966
- 6 *Corporate data file design*
EDP Analyzer December 1966
- 7 G SCHECTER
Information retrieval
Thompson Book Company and Academic Press 196
- 8 J MARTIN
Programming real-time computer systems
Prentice Hall 1965
- 9 J R KLEESPIES
A progress report on developing a comprehensive information system
Presented at the Annual Meeting of the Association of Edison Illuminating Companies at White Sulphur Springs West Virginia October 4 1967

Omnibus:

A large data base management system

by ROY P. ALLEN

Industrial Indemnity Company
San Francisco, California

INTRODUCTION

In designing a third-generation data base and a system for interrogating and maintaining it, Industrial Indemnity Company set the following general objectives:

1. The management and retrieval system should be oriented to providing its services in a *simple and straightforward* way to application programs written in a high-level language, such as PL/1 or COBOL.

2. The data base should be *compact*. Its second-generation predecessor was a pair of files occupying together some 35 reels of magnetic tape; but the new data base, which would need to meet significantly expanded data requirements, was to fit onto a single IBM 2321 data cell drive (capacity about 390 million bytes).

3. It should be possible to design, program, and implement the system with a configuration of machines and personnel *commensurate with the anticipated benefits*.

4. Provision should be made for storage, retrieval, and maintenance of *data elements with widely varying space requirements*. Well over half of our policies have no claims at all, but a few policies for very large companies acquire claims in the thousands.

5. Both *sequential and direct accessing*, in a variety of combinations, should be feasible, in order to accommodate file-scanning applications, efficient updating, and an open-ended variety of inquiry and reporting applications.

6. *Retrieval should be swift* enough to maintain rapid response to inquiries coming from several dozen remote inquiry terminals.

7. A very *high level of data integrity* must be maintained, with protection implemented in at least three ways: detection of improper attempts to alter the contents of the data base; prevention of incorrect updating; and recovery from data, program, or machine failures in a minimum of elapsed time and with a high level of

justified confidence in the restored version of the data base.

8. Both the data base structures and its management system should be flexible enough to permit *new and unforeseen data requirements* to be accommodated in the future with a minimum of disturbance to either operational application programs or the data base management system itself.

I would like to describe to you "Omnibus," the system developed to meet these objectives. The description will be presented in five topics: the logical structure of the data; data interfacing and retrieval; protecting data integrity; accommodating changing data requirements; and patterns of retrieval.

Data structure

At the most general level, data managed by Omnibus are clustered by one primary and one secondary identifier. Since Industrial Indemnity is an insurance carrier, our primary identifier is the *insurance policy number*; each data item is related to a single, specific policy. Data that apply to a policy as such, rather than to its individual claims, have no secondary identifier; data relating to specific claims, however, are grouped by *claim number*, the secondary identifier. Each claim is subsidiary to the specific policy on which the claim is made.

The terms "policy" and "claim" apply specifically in the insurance industry, but the philosophy and techniques of Omnibus are applicable for any large data base that can be similarly structured according to one primary identifier and a very small number of subsidiary identifiers. For the sake of concreteness, the application-oriented terms "policy" and "claim" are used throughout this discussion.

The basic unit of data organization in Omnibus is the *field string*; a field string is defined as a group of one or

more fields structured according to a corresponding *Format*, and a *Format* is defined as a fixed pattern of related fields. Since Omnibus can accommodate a variety of data requirements, few or many *Formats* may be defined to the system at a given point in time. *Formats* are identified by *Format numbers* in the range 1-27; *Formats* numbered 1-63 apply to field strings such that each relates to its respective policy, but not to any specific claim; whereas *Formats* numbered 64-127 apply to field strings such that each relates to a specific claim on a specific policy. Each field string carries with it the *Format* number of its corresponding *Format*.

Field strings are logically and physically grouped in *sections*; a *section* is defined as a group of one or more related field strings such that Omnibus is required to treat the group as indivisible in mass storage. Specifically, a *policy section* is defined as the group of field strings relating to a particular insurance policy but not specifically to individual claims, and a *claim section* is defined as the group of field strings relating to a particular claim on a policy. Each policy section consists of one field of *Format 1* (the policy summary *Format*), followed by zero or more field strings (for the same pol-

icy) numbered 2 - 62*; each claim section consists of one field string of *Format 64* (the claim summary *Format*), followed by zero or more field strings (for the same claim) numbered 65-127.

The complete data recorded in the data base for any given insurance policy, then, consist of one policy section followed by zero or more claim sections.

Data are transferred between application programs and Omnibus in either *standard form* or *long form*, where the form is defined as the sequence of field strings that is assumed in such a transfer. *Standard form* consists of one, two, or three field strings: the field string of direct interest is always last; if that field string is numbered greater than 64, it is preceded by the same claim's *Format 64*, or claim summary, field string; in any case the first field string is the policy's *Format 1*, or policy summary, field string. Thus a data item is always accompanied by the most closely related summary data.

Long form consists of a variable number and arrangement of field strings, depending on the amount and complexity of the data recorded for the given policy or claim. In long form, all the field strings in a specified section are transferred, in ascending order by *Format* number; if the specified section is a claim section, the *Format 1* field string of the related policy precedes the claim field strings. An application program receives data in long form when it specifies a *Format* number of zero; otherwise, transfers are in standard form.

This much is all the application programmer needs to know about the data structure in Omnibus.

Each rectangle represents a field string of the *Format* indicated.

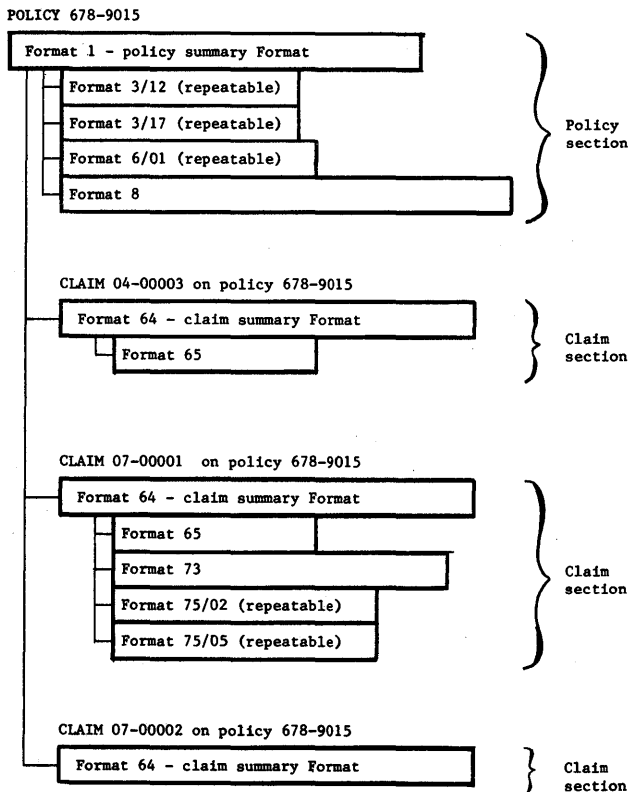
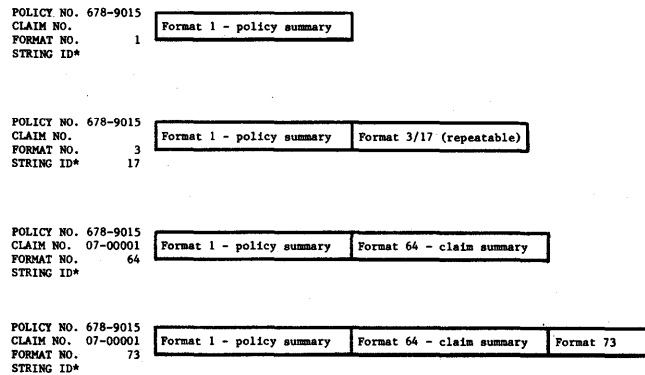


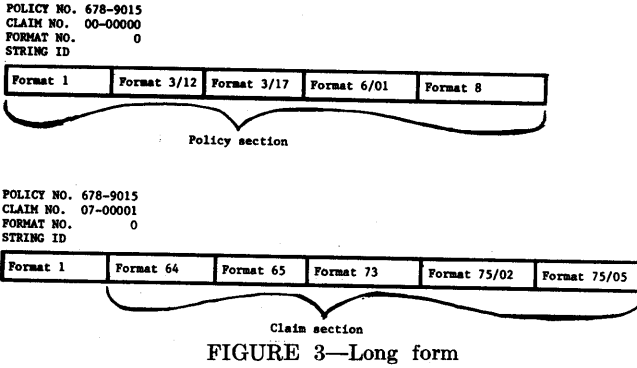
FIGURE 1—Logical structure of a policy's data



*When a *Format* is defined as repeatable (meaning that a given section may contain more than one instance of that *Format*), the instances within a section are distinguished from one another by a 2-digit string identifier. This term has no application to non-repeatable *Formats*.

FIGURE 2—Standard form

**Format 63* applies to field strings used for indexing the claims of "HVC" policies, as described in the following section of this paper; its instances constitute separate sections, called "claim index sections," for Omnibus internal use only.



For instance, even though each policy and claim has only field strings of such Formats as are appropriate to its particular requirements, it is still true that much of the space in most field strings is, in varying degree, wasted: inapplicable fields are empty (blank or zero), amounts are substantially smaller than the maximum for which space is allocated, character fields contain many low-order blanks.

To reduce the amount of comparatively expensive direct-access storage so wasted, Omnibus defines two representations of data for each Format: *expanded* and *condensed*. In *expanded* representation, the number of fields, and the size and usage of each field, for a given Format, is fixed, and applies identically to every field string of that Format. In *condensed* representation of a field string, unused fields are omitted, and noted as such in a bit matrix stored with the data; high-order zeroes

Data interfacing and retrieval

Internally, however, different classes of fields, claims, and policies are accessed in different ways, according to their individual characteristics.

FIGURE 4—Examples of expanded & condensed representation

Field name	Expanded representation	Expanded Length	Condensed representation	Condensed Length	Technique
(Matrix)			11011111, 10011111	2	
1. Policy number	6, 7, 8, 9, 0, 1, 5	7	67, 97, 97, (6789015 ₀)	3	Zoned → binary
2. Insured's name	P, M, JONES STORE	25	11, P, M, JONES STORE	8	Variable-length character string
3. and address	12, BROAD STREET	25	omitted (see matrix)	0	
4. HAMILTON, NEW YORK	12, HAMILTON, NEW YORK	25	0E, 12, BROAD STREET	17	
5. Zip code	1, 5, 0, 4, 5	5	12, HAMILTON, NEW YORK	18	Zoned → packed
6. Agent number	9, 8, 7, 6, 5	5	15, 04, 5+	3	
7. Effective date	12, 25, 66	6	98, 76, 5+	3	Zoned MoDaYr → binary DayYr
8. Expiration date	12, 24, 67	6	8C, 7E, (359-66 ₀)	2	
9. Date cancelled	00, 00, 00	6	8C, 1B, (358-67 ₀)	2	Variable-length packed decimal field
10. Date reinstated	00, 00, 00	6	omitted (see matrix)	0	
11. Premium	00, 00, 45, 13, 5+	5	omitted (see matrix)	0	Zoned → binary
12. No. claims	00, 00, 3+	3	45, 13, 5+	3	
13. Claims amount	00, 00, 43, 20, 0+	5	3+	1	Zoned → binary
14. Binary flags 1	0, 1, 1, 0, 0, 0, 0, 1, 1 ₀	8	43, 20, 0+	3	
15. Binary flags 2	1, 0, 1, 1, 1, 1, 0, 1, 1 ₀	8	61, (01100001 ₂)	1	Zoned → binary
TOTALS		170	BD, (10111101 ₂)	1	
				77	

are dropped from numeric fields; numeric character fields are stored either in two-digits-per-byte form or in binary; and so on. For each Format defined to it, Omnibus uses a *Format map* to specify for itself the relation between the condensed and expanded representations of the fields that make up the field strings of that Format. Each two-byte field descriptor in the map specifies, for one field (or group of similar adjacent fields) of a Format, the particular technique of expanding and condensing to be used, and the conditions under which the field can be omitted in condensed representation. This characteristic alone of Omnibus reduces the storage requirements for our data base by about sixty percent, a savings in mass storage cost that appears to constitute a sufficient cost justification for the design and programming effort.

Policies are recorded in, and retrieved from, direct-access storage in one of three modes, depending on the size of their condensed representations.

(1) A "one-track" policy is one such that its policy section and all its claims (if any) will fit, in condensed representation, on a single direct-access track (on the data cell drive, 2000 bytes or less); these policies have

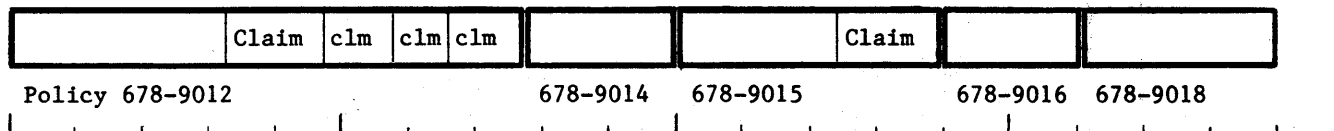
from zero to about 15 claims each, and account for nearly all policies, but scarcely more than half of all claims. A particular section, when requested, is located by Omnibus by simply searching the single logical record for that policy on the single track that contains all the policy's sections.

(2) A "two-track" policy is one that is too large, in condensed representation, to fit onto a single direct-access track, but will fit onto two consecutive tracks without requiring any section to straddle them; these policies have about 15 - 35 claims each, and account for a small percentage of the total number of policies. Omnibus locates a particular section of a two-track policy by checking the last section in the single logical record on the policy's first track to determine which track should contain the requested section, then searching the appropriate track.

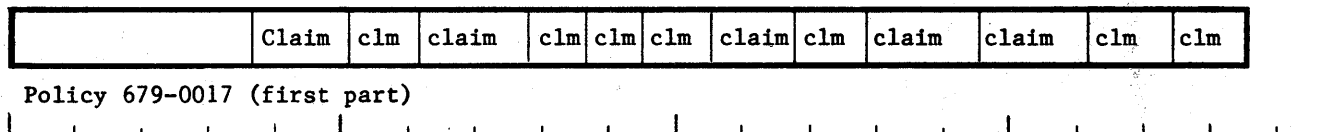
(3) A "high-volume-of-claims," or "HVC," policy is one, the condensed representation of which is too large to fit onto two tracks; these policies have from about 35 to several thousand claims. They represent a relatively small number of policies, but they account for nearly a third of all claims. Two levels of index are

FIGURE 5—Examples of one-track and two-track policies

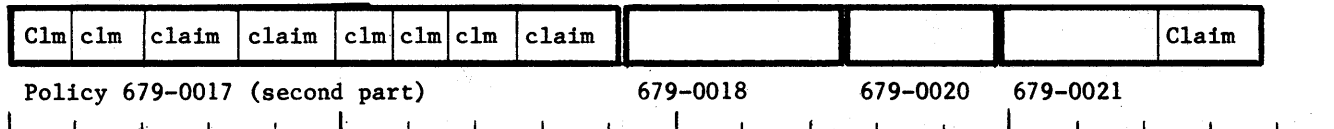
Track 47 - contains five one-track policies



Track 53 - contains the first-track portion of a two-track policy



Track 54 - contains the remainder of the two-track policy's data, and three one-track policies



Each rectangle above represents a section: blanks represent policy sections, and claim sections are marked as such. Each separate logical record is marked off by a heavy black outline; a single logical record contains data for one and only one policy.

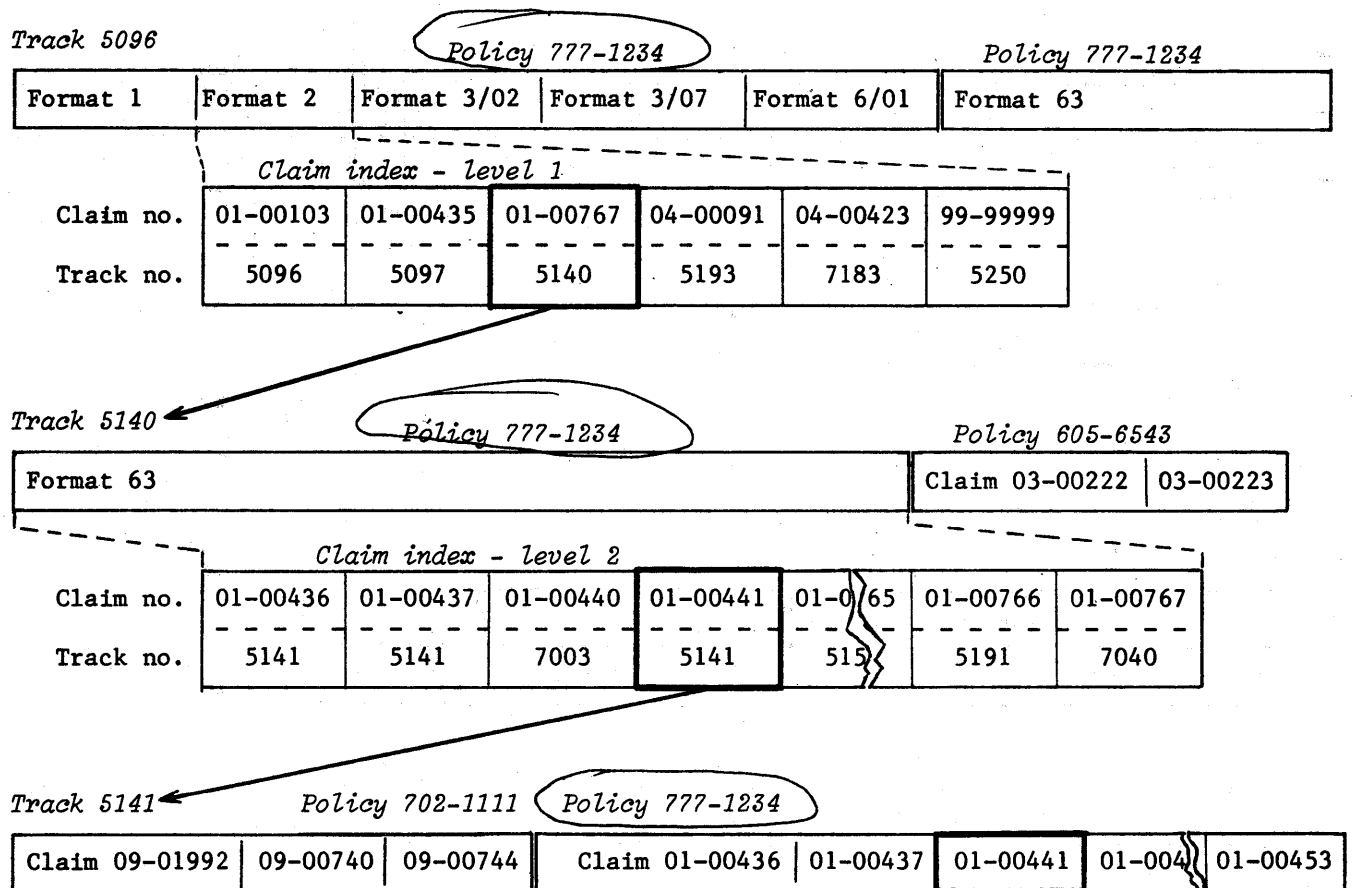
used to proceed from an HVC policy's policy section to a specified claim section. The first level is stored within the policy section itself, in a field string of a special-purpose Format, not accessible to application programs; this index points to the direct-access track(s) on which the second-level index record(s) reside(s), normally either the same track as the policy section or the ones immediately following. A second-level index record for an HVC policy contains one entry for each claim, within its range of claim numbers, recorded for that policy; each entry points to the track on which its claim is currently recorded. Thus, when Omnibus has secured the policy section of an HVC policy, and it seeks a particular claim on that policy, it searches that policy's first-level claim index for the pointer to a second-level claim index whose number range includes the claim number sought. Using that pointer, it secures the appropriate second-level index, and binary-searches it for the entry for the desired claim number, which points to the track on which that claim is recorded. Some effort is expended, both when reorganizing the data base and when maintaining it, to assure that all sections of an HVC

policy are recorded on the same data cell strip if at all possible, in order to avoid excessive seek time.

Physically, the data base is divided into four regions: a *Prime* area, where one-track and two-track policies are stored sequentially by policy number whenever the data base is reorganized, packed tightly into one-track blocks; an *HVC* area, where high-volume-of-claims policies are stored sequentially at reorganization time, with overflow space reserved at the end of each strip in order to minimize the chance of a policy being forced by additions or modifications to straddle a strip; and a pair of *Activity* areas (one each for the Prime and HVC areas) to receive new policies and claims and relocated data, as required, during the cycle between one reorganization and the next.

Whenever updating or adding data causes enlargement of the condensed representation of a section, it is possible that the new, larger, condensed data will no longer fit its old home location. This could occur because a previously unused field comes into use, because a new field string is added, because an amount field comes to have fewer high-order zeroes, or for any of

FIGURE 6—Example of high-volume-of-claims policy.



several other reasons. When this occurs, one general rule obtains: the home track of data separable from that being moved because of its own enlargement always remains the same. For example, if a one-track policy enlarges so that it will no longer fit its old space, Omnibus does not move any data for other policies that may be sharing the same track, but instead moves only the affected policy; if an HVC claim section enlarges so much that it no longer fits, no policy section, claim index section, or other claim section is relocated because of that claim's move. In the case of a two-track policy, both parts are moved together, even though only one may have expanded. After data are moved from a track, the data remaining on that track, if any, are packed together, so that no trace remains on that track of the removed data, and all its space is available for other subsequent use, such as the expansion of some of the remaining data. Any affected pointers in indexes are, of course, updated immediately when data are moved.

In spite of the breaks in physical sequence that may be caused by additions, relocations, and the use of separate regions for HVC and other policies, Omnibus

can find any policy, and therefore any section, fairly swiftly, using a very simple two-level indexing scheme.

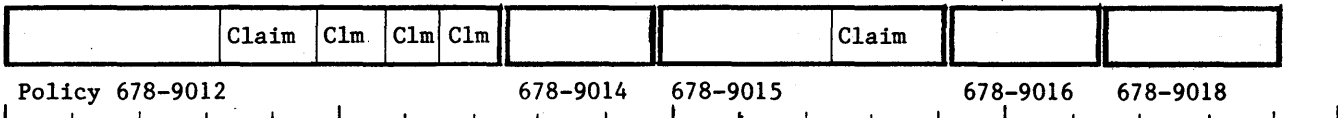
A one-for-one index of policies is maintained on disk, in sequence by policy number; it lists, for each policy in the data base, its policy number, certain bit flags that may obviate the necessity for retrieval, and a pointer to the track in the data base on which the policy section is currently recorded. This index is recorded in fixed-length, keyed blocks of 134 entries each, with each block having as its key the policy number of its last (highest-numbered) entry; four keyed blocks are recorded per track.

To access this index efficiently, a core-resident index of some 1200 entries is used; this index consists of the key, or last policy number, of the last block on each track in the disk-resident index, in ascending order by policy number. The relative position of each entry in the core-resident, or first-level, index implies the relative track number of the corresponding track in the disk-resident, or second-level, policy index. Thus the nineteenth entry in the core-resident index consists of the highest policy number indexed on the nineteenth

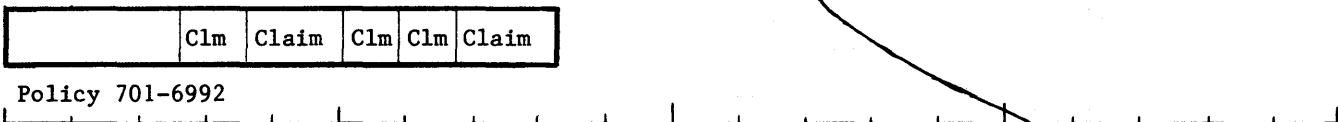
BEFORE

FIGURE 7—Moving a policy because of adding a new claim

Track 47 (Prime area)

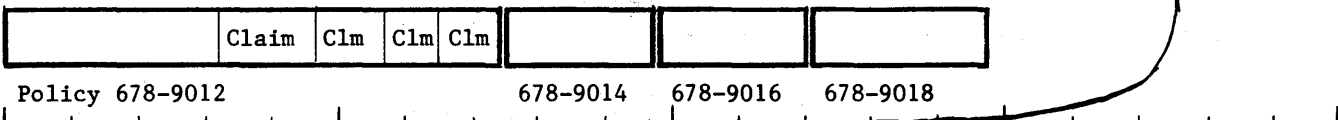


Track 1345 (Prime Activity area)

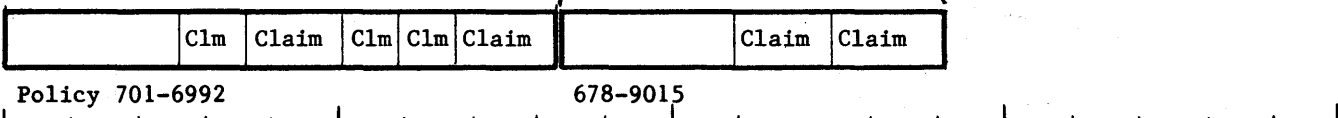


AFTER

Track 47



Track 1345



track of the disk-resident index, the one-thousandth entry in core corresponds to the one-thousandth disk track, and so on. To secure the direct-access address of a particular policy, then, Omnibus performs a binary search of the first-level index for the first entry at least equal to the sought policy number. The relative entry number of the selected entry is used as a relative track number for the second-level index, and the first block on that track having a key at least equal to the sought policy number is read into core. This block, in turn, is binary-searched for the entry matching the sought policy number; the matching entry points directly to the track on which the policy section is currently recorded. Failure to find a matching entry in the second-level index ends the search by establishing that no policy of that number is recorded in the data base.

Thus Omnibus requires, at most, one disk seek and one data cell drive seek to reach a specified policy, and at most one disk seek to find that the specified number is not in the data base. This is significantly swifter than would be the case, for instance, with IBM's Indexed Sequential Access Method applied to a data base this size, which would require at best three or four relatively

time-consuming data cell drive seeks, whether the policy were present or not.

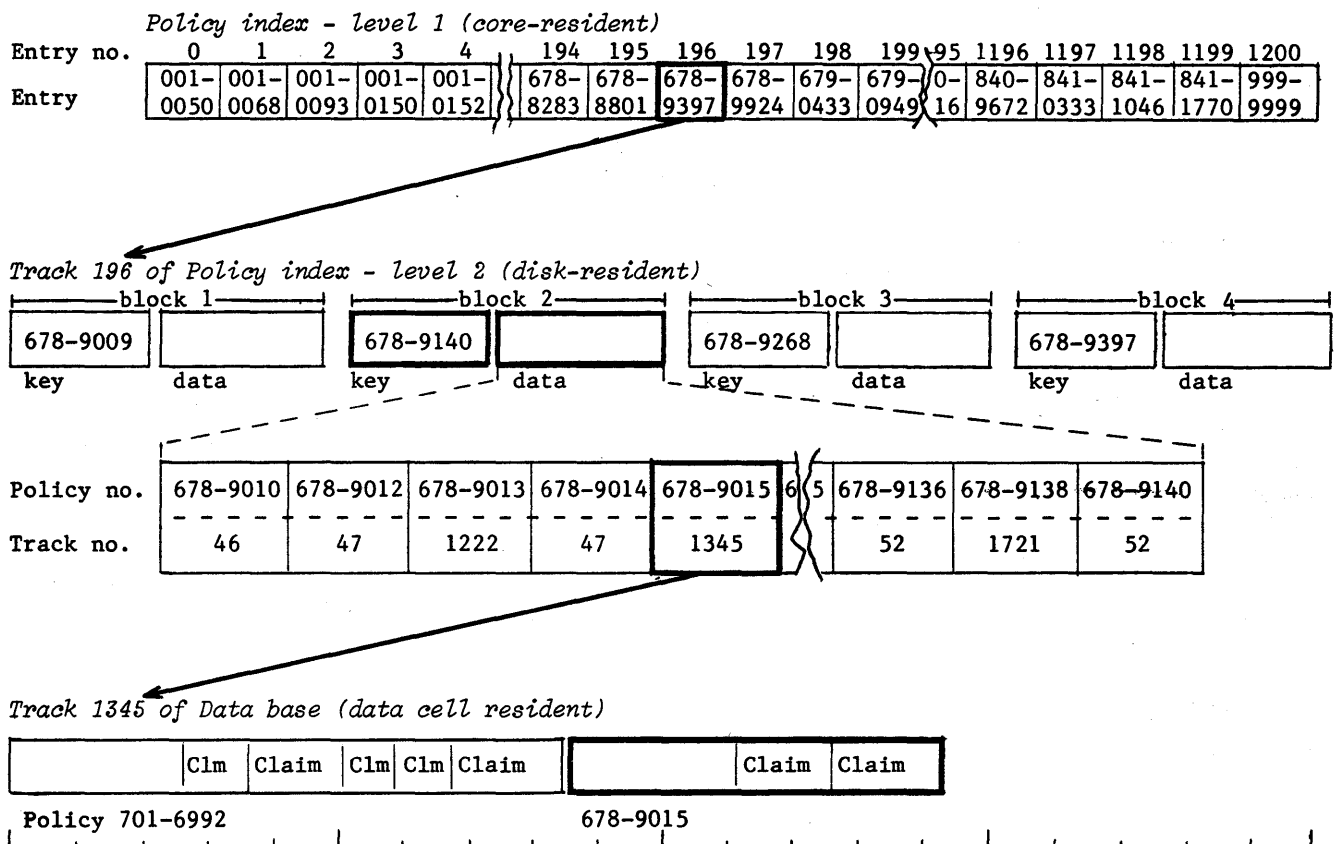
Data integrity

It was mentioned at the outset that Omnibus has as one of its design objectives the maintenance of a high level of data integrity, to be implemented through error detection and prevention schemes, and through a reasonably quick and very reliable means of recovery from damage of one kind or another.

The *detection* requirement is approached in three ways. First, when an application program presents new or modified data for inclusion in the data base, Omnibus' field string condensing routines include checking each field for proper data type. Detection of improper data type, such as non-numeric data in what should be a numeric field, is considered to render the entire request unexecutable. This serves principally as a safeguard against an incompletely debugged updating program, or a program that has not yet been recompiled to reflect a changed Format definition.

Second, Omnibus maintains a set of control totals based on certain key summary fields in the claim and

FIGURE 8—Policy indexing



policy summary field strings. All of these totals, together with a count of policies recorded, are maintained by Omnibus at the volume and data set levels; the claim summary totals are also maintained by Omnibus (not by application programs) in the Format 1 field strings of the corresponding policies. These various totals are used to detect certain kinds of errors, as follows: When a data base updating program opens Omnibus, Omnibus provides to that program a copy of the then-current data base grand totals. As the program prepares each request for the insertion, deletion, or updating of data, it is to compute the effect the change should have on the control totals, then post that effect to its copy of the data base grand totals. On receiving the request, Omnibus computes what the actual effect of the change would be and compares the result to the program's copy of the control totals. If they do not agree, the request is rejected, and the application program is so notified. This procedure assures, essentially, that the application program "knows" what it is doing, and it keeps the program and Omnibus locked in step with each other on an up-to-the-second basis.

Third, as Omnibus actually carries out its output operations, it computes the net effect on volume and data set control totals of the changes it makes to each track; if either the sum of all track changes or the sum of all volume changes caused by a request fails to balance to the expected net change, Omnibus catches itself in the error.

Error prevention methods in Omnibus are oriented to two different types of errors: those caused by faulty application programs, and those caused by mutual interference of concurrently executing programs in a multiprogramming environment.

Two aspects of the prevention of certain kinds of application program errors have already been touched upon, namely, rejecting requests with improper data in one or more fields, and rejecting requests that incorrectly predict control totals. But nothing mentioned so far would affect the situation in which several changes are to be made concerning a single policy or claim, and at least one of them is faulty and detected as such; under most circumstances, it would be desirable to reject all those changes as a package, so that they can be resubmitted later, in corrected form, as a unit. To accomplish this, Omnibus collects all consecutively issued requests for insertion, deleting, or updating relating to a single section into a *queue*, checking each request for validity and control total balance as it is received. The requests so enqueued are not executed until the queue is complete, that is, until all the requests affecting that section have been issued and the program has explicitly requested queue execution. At this point, Omnibus executes all of the enqueued requests if and only

if each of them was valid and in balance; if any of them was detected as being faulty, the entire queue is purged without execution, and the application program is so notified. This removes the difficulty for the application program, or for a person later attempting to correct data errors, of trying to recover from the partial execution of an integrated group of transactions, and it eliminates one way in which conflicting data could come to be recorded in the data base.

Two types of mutual interference by concurrent programs are prevented by Omnibus procedures: (1) If two updating programs run concurrently, each could accidentally undo some of the work of the other by concurrently working on the same track or tracks, and neither would be able to maintain valid control totals. (2) If an updating program and one or more other programs use Omnibus concurrently, the situation could arise of one of the other programs using a pointer (such as an index entry) that was secured before the update program happened to move the data in question to a different track, but used after the move, with the result that the program doesn't find the data where the pointer points.

The former of these situations is prevented very simply: Omnibus permits only one program to have its updating facility in an open state at a time.

Sequence	Update program	Inquiry program
1.	Reads policy 123-4567 from data base track 40.	Idle.
2.	Manipulates data in core, preparing to add a new field string of Format 8.	Receives a request for data concerning policy 123-4567.
3.	Moves policy 123-4567 from track 40 to track 916.	Reads the disk-resident policy index record for policy 123-4567 (points to track 40).
4.	Updates the disk-resident policy index record for policy 123-4567, so that it now points to track 916.	Reads data base track 40, expecting to find policy 123-4567.
5.	Proceed with processing next transaction.	????????????????

4.	Updates the disk-resident policy index as above.	Waits for update program to release exclusive use of the data base.
5.	Proceeds with processing next transaction, until next OMNIBUS request.	Reads track 40, expecting to find policy 123-4567.
6.	Waits until exclusive use of the data base is available again.	Re-examines the policy indexes; finds pointer for policy 123-4567 pointing to track 916.
7.	Waits.	Reads track 916, finds policy 123-4567.

FIGURE 9—"Musical sections"

The latter situation is a bit more complex. It could be avoided by shutting out all other access to the data base during execution of an updating program, but that seemed unduly restrictive. Instead, we assign to the update program a relatively low priority, and we assign a higher priority to inquiry programs. Omnibus secures for the updating program temporary exclusive use of the data base each time it begins accessing a new section; this exclusive use is retained until the updating program is through with that particular section, as indicated by the purging with or without execution of a queue of output requests for the section, or the receipt of a request from the updating program applying to a different section. At that point, exclusive control is released, and all inquiry requests that have accumulated in the meanwhile take shared use of the data base until they have all been carried out; once that is done, exclusive control returns to the updating program. This technique does not make it impossible for an inquiry program to find a pointer before a change of location and use it after, but it does make it possible for Omnibus, once such a situation is encountered (as indicated by data not being found where a pointer pointed), to re-examine the indexes with the assurance that the updating program cannot change the data base or any of its indexes again until after the inquiry program's data are found and secured from their new location.

In spite of these error detection and prevention procedures, however, it is a commonplace that data, program, and hardware failures can and do still occur; means must be at hand to recover reasonably quickly and very reliably from such failures. In a magnetic tape oriented environment, such backup is provided by the "grandfather" system, whereby, after each update of a file, we would still have the tape from before the update, permitting a corrected rerun if needed. In direct-access environments, however, with updating in place, the predecessor of an updated block is lost; in case of an error, either the file must be rebuilt or the individual changes must be reversed. For direct-access files of modest proportions (*e.g.*, one occupying a single disk pack), a simple and only moderately inefficient procedure is to copy the file onto tape before updating it, so that, in the event of a glitch of whatever sort, the file can be restored by simply copying the tape back onto direct-access. With a very large data base, however, this approach would be most costly, in both time and money: to dump the full contents of a data cell drive to tape requires more than three hours and about 22 reels of magnetic tape. A reliable way had to be found to individually reverse individual changes.

Omnibus accomplishes this as follows: each time the contents of a track in the data base are to be altered, we first write onto a magnetic tape file a single block

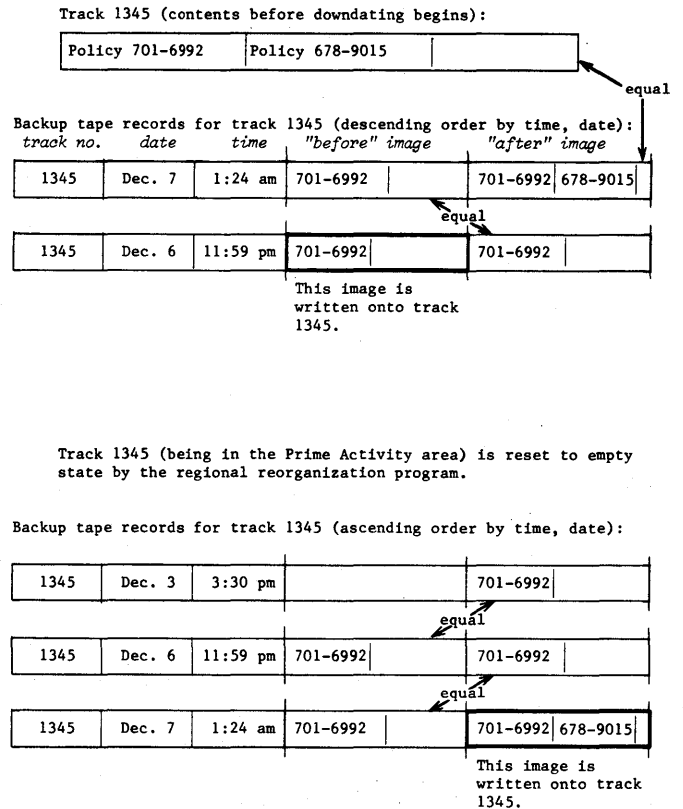


FIGURE 10—Using the backup tape for error recovery

containing the address of the track being changed, the date and time of the change, and an image of the track both before and after the change. If, after executing all or part of an update run, it is discovered that errors have occurred, the run can be reversed by sorting the before-and-after tape into descending sequence by time within date within track address, then passing the sorted result against the updated file, writing onto each affected track the "before" image from its oldest backup record. The presence of both "before" and "after" images provides a check on the validity of the procedure, since the contents of the track before reversal should be, bit for bit, identical with the newest "after" image on the tape. If several days' runs need to be reversed at a time, their backup tapes can be sorted together, and the same procedure is followed. Since the control totals are recorded within the data base, this procedure will effectively restore them as well. The disk-resident policy index is backed up by simply dumping it to tape before each update run; consequently, it can be restored in a conventional manner. This procedure (using the "before-and-after" tape) is both swifter and more reliable than attempting to reverse the updates with another application program: swifter because tracks are restored in physical sequence (thus minimizing access time); and more reliable because it

does not require decisions about which transactions to reverse and how, and because it is carried out by a very simple program that is not sensitive to changing application and data specifications.

Similarly, this method provides backup in the event of hardware problems, such as mangling a strip, scraping a recording surface, or even losing an entire cell. The last preceding data base reorganization is, in this event, rerun, but time is taken only to recreate the affected region, not the entire data base. At this point, all of the data base except the affected region is at current status, and the affected region is at the status it had *after* reorganization, but *before* the intervening updates occurred. All of the backup tapes created by updates since that reorganization are searched for before-and-after records of tracks in the affected region; these records are sorted into ascending sequence by time within date within track address, then passed against the data base. This time, however, the first "before" image should match the initial status of an affected track, and the last "after" image should be written to bring it up-to-date. Again, by proceeding in physical sequence, and by processing only the area that needs restoring, this procedure is relatively swifter than reorganizing the entire data base and then rerunning all intervening updates; and it is more reliable for the reasons mentioned above, not to mention the elimination of the need to use more than one version, in proper sequence, of any updating program that was changed in the meanwhile.

The procedure whereby we step backwards from the current status of the data base to undo bad updates we call "downdating"; the procedure whereby we step forwards to reconstruct a region of the data base we call "unclobbering." Each is a very conservative, self-checking procedure.

Accommodating changing data requirements

In the "normal" data processing environment, once a decision is reached to include additional data in a file's records (perhaps to accommodate a new application or to enrich the reporting vocabulary), a trauma must occur in which each program that uses the file must be revised, recompiled, and (possibly) retested; then a program to convert the records from the old format to the new one must be written, debugged, and run; and steps must be taken to assure that the conversion is total, so that only new versions of programs are run against the file in its new format. The next time a new field or set of fields is required, the same sort of trauma must be endured again. In practice, this has the consequence *either* that improvements to the system are made at considerable cost and some danger, *or* (possibly worse)

that improvements are not made because of the cost and risk involved in revision.

This sort of problem probably cannot be avoided entirely. Omnibus does, however, possess several characteristics that materially increase our flexibility in the face of unanticipated new data requirements.

When a new application arises, it will typically require, not just one or two, but a whole cluster of related new fields for each affected entity. With Omnibus, such a cluster would be defined as a new Format. It would be assigned an available Format number, in the appropriate number range according to whether it should be considered a policy or a claim Format. The expanded and condensed representations of the new Format would be designed, then defined to Omnibus in the form of a new Format map. This would alter neither the condensed nor the expanded representation of any existing field strings, so no operational program would need to be changed simply in order to continue valid operation at the current level. No massive, all-at-once conversion of data base and programs is entailed; programs that recognize, use, insert, and maintain the new data may be put into operation when and as they are ready.

When only a few new fields are to be added, they will most probably be implemented as an addition to some existing Format, rather than as a new Format; this approach requires a little more effort. The map of the affected Format must be updated to include definition of the new fields; but, as when adding a new Format, the data base need not be rebuilt, since it will simply show these new fields as being empty and therefore omitted from the individual condensed field strings. Programs that use field strings of the affected Format do have to be recompiled, in order to reflect the new expanded representation; to minimize this effort, among other reasons, we maintain in direct-access storage a cataloged description of the expanded representation of each Format currently defined, coded in a form suitable for inclusion in source programs written in a high-level language.* Consequently, when a Format is changed, we simply alter the cataloged description of it accordingly, then recompile (without any further change) each of the programs that use field strings of that Format. This technique removes the need for a massive file conversion effort and for individual, manual revision of each affected program.

*In a COBOL shop, these cataloged descriptions would consist of data definition statements for incorporation in the Data Division by means of COPY statements. For PL/1, each would consist of a structure declaration for incorporation by means of a %INCLUDE statement.

Patterns of retrieval

The data base structure used by Omnibus permits considerable flexibility in the patterns of retrieval to be made available to application programs, while requiring linkage maintenance techniques of only moderate quantity and complexity.

First, and most simply, a program may proceed directly to the policy, section, and field string it specifies, or it may sequentially retrieve consecutive instances of a particular (repeatable) Format from a given section, consecutive field strings in a given section, field strings for consecutive claims on a policy, or field strings for consecutive policies within the data base, in any desired mix of directness and sequentiality. Since a large share of our transactions currently carry with them the policy and claim numbers to which they apply, this provides efficient access for most of our current needs.

To provide effective and efficient ways of collating data related in other ways, however, a number of additional *list processing techniques* are feasible when using Omnibus; it is anticipated that they will find extensive application.

Some sections are related in *rings*: that is, each member of a group of related sections contains a pointer to the next section, and to the preceding section, of the

The following policies constitute a six-member ring; they are all policies for the same insured.

<u>Policy number</u>	<u>Next policy "leftwards"</u>	<u>Next policy "rightwards"</u>
678-8930	678-8934	678-9015
678-8934	678-9008	678-8930
678-9008	678-9034	678-8934
678-9034	678-9124	678-9008
678-9124	678-9015	678-9034
678-9015	678-8930	678-9124

In the diagram below, each rectangle represents a policy in the data base.

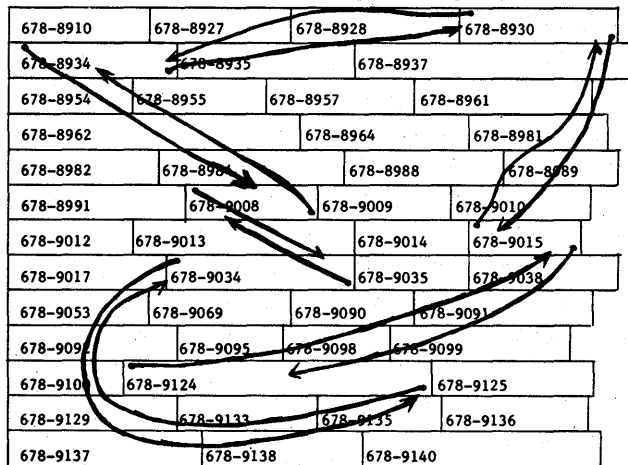


FIGURE 11—Ring structure example

The following policies constitute a six-member simple list; they represent succeeding "generations" of policies on the same risk.

<u>Policy number</u>	<u>In force during year ending</u>	<u>Renewal of policy no.</u>	<u>Renewed by policy no.</u>
678-8927	December, 1964		678-8955
678-8955	December, 1965	678-8927	678-8989
678-8989	December, 1966	678-8955	678-9015
678-9015	December, 1967	678-8989	678-9125
678-9125	December, 1968	678-9015	678-9138
678-9138	December, 1969	678-9125	

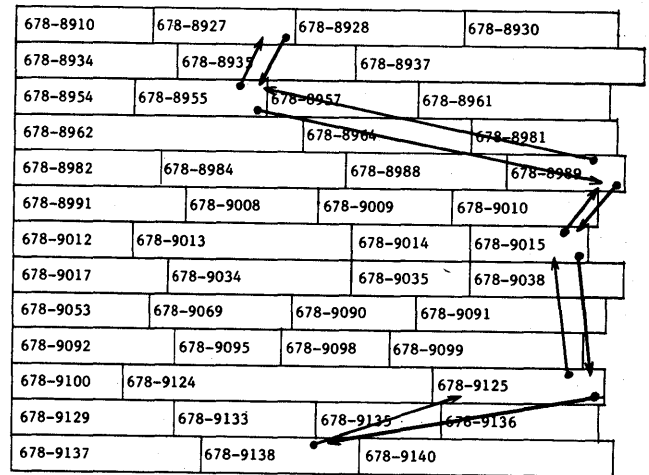


FIGURE 12—Simple list structure example

group; stepping though in either direction from any member of the ring will eventually bring you back to your starting point, having in the meanwhile retrieved every member of the ring in turn. One use for the ring structure is to link together different policies currently in force for the same insured party.

Other sections are related in *simple lists*: that is, they point to one another in a string, but no attempt is made to close the ends of the list together to form a ring. One use for the simple list structure is to point from each generation of coverage on a single risk both backward (to the policy of which it was a renewal) and forward (to the policy, if any, that is a renewal of it); the list terminates at one end with the oldest policy on that risk in the data base, and at the other end with (typically) the policy currently in force.

A wide variety of other patterns of retrieval is to be available to us in an efficient way by means of *inverted lists*, to be maintained on disk. ** Each such list is a string

**The file containing these inverted lists is not managed by Omnibus, and Omnibus does not include the routines for maintaining or using any of these list structures. They are mentioned here to illustrate some of the uses to which a data base managed by Omnibus is susceptible.

To secure data concerning, for example, the effect of the 1969 Duckburg earthquake (catastrophe number 77) on the customers of the Drake Insurance Agency (producer number 69000), retrieve the data at the intersections of the following pair of lists; that is, retrieve the claims* in the first list the policy numbers of which also occur in the second list.

<p>List 1. <i>Claims arising from catastrophe ??</i> <i>(policy no./claim no.)</i></p> <p>678-8930 / 07-00004 678-8937 / 02-00105 678-8937 / 02-00106 678-8937 / 02-00108 678-8981 / 07-00004 678-8981 / 07-00005 678-9010 / 07-00003 678-9015 / 07-00001 678-9034 / 02-00047 678-9099 / 02-00012 678-9124 / 02-01005 678-9124 / 07-00999 678-9124 / 07-01005 678-9136 / 07-00023 678-9136 / 07-00024 700-1234 / 07-00001</p>	<p>List 2. <i>Policies produced by producer number 69000</i> <i>(policy number)</i></p> <p>678-8927 678-9909 678-8935 701-6992 678-8981 702-1111 678-9014 777-1234 678-9015 777-1235 678-9017 777-1236 678-9095 777-1238 678-9124 777-1239 678-9138 678-9140 678-9220 678-9221 678-9227 678-9817 678-9819 678-9820</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Thus, the relevant data consist of the following claims (expressed as policy no. / claim no.):

- 678-8981 / 07-00004
- 678-8981 / 07-00005
- 678-9015 / 07-00001
- 678-9124 / 02-01005
- 678-9124 / 07-00999
- 678-9124 / 07-01005

FIGURE 13—Intersections of inverted lists

of pointers to policies or claims related in some way. For instance, maintaining, for each insurance agency that produces policies for us, a list of all the policies produced for us by that agent, and also maintaining, for each separate disaster (such as a flood, civil disorder, or major fire), a list of all the claims we have received arising from that disaster, would enable us to swiftly find, for a specified agent, a variety of data concerning the disaster's impact on that agent's customers. A very extensive library of such inverted lists is anticipated; they entail no modification of the data base.

What makes such an extensive network of list structures feasible is the swift and direct indexing scheme by policy number used by Omnibus. The pointers in each list of policies are maintained simply as policy numbers, not as direct-access addresses; since the use of the index increases the total elapsed time to find a policy by less than 20% as compared with having the policy's address to start with,* this does not seriously degrade retrieval speed. The use of the policy number (or the policy number/claim number pair) as a pointer has, of course, the desirable consequence that changing the location of a section in mass storage does not require finding and up-

*It causes no increase at all if some other task is concurrently using the data cell drive, so that the one task's index lookup can be carried out during time that would be spent waiting in any case.

dating the related lists. This is a major simplification of the list maintenance task.

SUMMARY

In summary, I should like to review the ways in which the Omnibus system addresses itself to the objectives we defined for it.

1. Retrieval and maintenance functions are *simplified for high-level language application programs* because a group of shared routines, previously written and debugged, provides the index searching, input/output, condensing, expanding, relocating, control, and backup functions required for execution of a simply-specified task, and because use of source program library facilities reduces the burden of keeping up with data base revisions.

2. The data base is made *compact* by the condensing technique for omitting unused fields and portions of fields.

3. The *cost of implementation* is kept in proportion to the benefits secured by means of the condensing technique (which helps minimize hardware costs) and the philosophy of maintaining lists by policy and claim number rather than by actual address (which reduces the complexity of data base maintenance).

4. *Data elements of widely varying space requirements* are accommodated by means of the structuring of sections into variable configurations of field strings, and by the use of three different schemes to store and find policies claims, depending on the space requirements of each individual policy and its claims.

5. A high level of *flexibility with respect to patterns of access* is maintained by providing facilities suitable for use through extensive and varied list-structuring and list-processing techniques, so that the most effective technique can be selected individually for each separate pattern of relation.

6. *Swiftness of retrieval* is achieved by implementing a simple, fast policy indexing structure, and by planning for the use of inverted lists in high-speed mass storage for most list-processing and logical search applications.

7. *Data integrity* is protected by forcing an updating program to keep its control totals in step with those of Omnibus, and by saving the before and after images of each block being changed, as it is changed.

8. *Flexibility in accommodating new data elements* is achieved by means of the open-ended Format structure of the data base, and by the use of a changeable set of Format maps (in combination with the condensing/expanding routines) to define the individual Formats.

Thus Omnibus provides significant advantages with

respect to each of the primary objectives of our large data base management system.

ACKNOWLEDGMENT

The system described in this paper was developed at Industrial Indemnity Company during 1967 and 1968 by a team consisting of Donald Dewey, James Bolen, James Otagiri, Dennis O'Donnell, Albert Holman, and

the author, with assistance from Lee Jensen, of Information Systems Design, Inc.

BIBLIOGRAPHY

A METAXIDES

Data base management

Special Interest Group on Business Data Processing Newsletter
Number 4 February 1968

A design approach to user customized information systems

by ROBERT G. RANGEL

International Business Machines Corporation
Endicott, New York

Complex information systems such as plant or corporation manufacturing information systems, serve a wide range of users and manufacturing areas. These systems are often characterized by the requirement to provide user options in order to maintain the integrity of local procedures. Although there may be a strong effort at standardization throughout the system, the fact remains that the system must serve different levels of manufacturing, from components to end product, and must interface with a variety of personnel, from manufacturing line operator to management. Although a system such as this is normally required to fulfill a general set of needs, the system design cannot ignore local requirements. The end result can be a system consisting of numerous programs, many serving similar functions but each containing some distinctive feature.

This is particularly true in systems which have traditionally been regarded as process or product sensitive. A wide variety of manufacturing control systems, including process control, quality control and testing, are characterized by "special" requirements. Normally, these systems start by serving a limited number of process or product centers. Gradually, over a period of time, additional products and processes are added to the system. The programming burden in such an environment is compounded as the system expands its coverage and as added peculiarities must be served.

This problem can be alleviated by the development of a number of generalized application programs which, as a group, can satisfy the overall system requirement. At the same time, these programs provide, and in fact require, an extensive degree of user (e.g., quality or process engineering) participation to customize the general set

of facilities to his requirements. The programming burden is reduced by minimizing the amount of specialized programming effort and providing the user with the tools to do individual application tailoring. After these tools are developed, systems and programming become less involved with the peculiarities of each new installation. In addition, the user has a much more responsive application. He may now interface directly with the system to input new or updated processing specifications, without going through the inherent delays of design and programming effort.

Since the user now becomes deeply involved in the design of the system and the subsequent processing provided by the system, software support must be provided to control the use of his specifications. This support, in the form of a logic control subsystem, is an integral part of all phases of the overall application system. The logic control subsystem provides: (1) support for programmer specification of allowable options during application program development, (2) processing of option specifications at installation time, and (3) support for controlling the use of the user specifications at execution time.

This logic control subsystem is being developed to control user specifications in a manufacturing quality assurance system which forms a portion of a larger manufacturing information system. The information system consists of a number of local plant sites connected to a central site through an interplant teleprocessing network. Each local system will utilize "in-plant" terminals connected to a dual processor configuration. Application processing at each site will be performed in a System/360 Model 50 or 65 with one background partition and from three to seven teleprocessing partitions. Application programs, written in As-

sembler Language or PL/1, will run under Operating System/360 using Multiprogramming with a Fixed number of Tasks (MFT).

The quality assurance portion of the information system, which will utilize the logic control subsystem, is required to provide a wide range of services in each manufacturing plant. These services vary with the type of product, process and people being supported and also varies in the amount of control and information desired. The system must support both process control and management information type tasks. Process control jobs involve interfacing with manual, semi-automatic and automatic manufacturing processes for the collection and distribution of quality control data. Management information must be supplied to a number of management levels.

Although a general set of needs have been defined by the users, these needs vary within plants and from plant to plant within the corporation. Because of this environment the logic control subsystem is being implemented to allow extensive quality engineer participation in defining processing requirements. The quality engineer will interface directly with the system to specify his requirements. Application programs will then use these specifications under the direction of the logic control subsystem.

The manner of structuring application programs must lend itself to operation in the logic control subsystem environment. It is important to note, however, that this activity takes place under the direction of the user specification. The design principles used in system structuring are as follows:

1. *Modularity*

This is an established concept which is required in the system architecture. Software is developed in small, independent and easily controlled modules, each with a limited function. Modules are designed to be independent of their configuration with other modules. This allows the building of programs by linking modules in different combinations.

2. *Generalization*

Programs and the modules from which they are structured are generalized to permit their use over a wide range of processes. Modules are generalized by coding in a skeletal form. A number of internal parameters which affect the type and logic of processing are left blank within the modules. When these parameters are assigned

values, the module becomes customized to a particular process. Programs are generalized by the inclusion of both required and optional modules. The program becomes customized when the selection is made of the actual modules to be used for a particular manufacturing process or product.

3. *Ability to Customize*

The user is given the ability to customize processing for his particular need. Services are provided to input, store and access user specified options. To a large degree, the user has the ability to construct and modify the system. Logic control information resulting from user specifications is stored on a data set. This information consists of two types of options: (1) internal module parameters which affect the mode of processing within the modules and (2) routing information which specifies the sequence of processing from module to module within the application program.

4. *Execution Control*

Programs are built from independent and generalized modules which require the control information described above for proper execution. This control is provided by a logic control module contained in each application program. This module accesses control information, passes the processing parameters to the application modules within the program, and directs the sequence of execution from module to module. Application programs remain generalized until they are actually used, since the control information required for tailoring is not accessed and used until execution time. The logic control module, therefore, plays a critical role in the execution of any application sequence.

The logic control subsystem, which provides the software support necessary to handle user specifications, is involved with the application system in three separate phases. (See Figure 1.)

Phase 1—System Development

- a. At the time application modules are developed, the programmer's definitions of module parameters are input to the logic control subsystem for storage. These parameters are later assigned values by the user to vary the internal processing of the module. The number of parameters defined

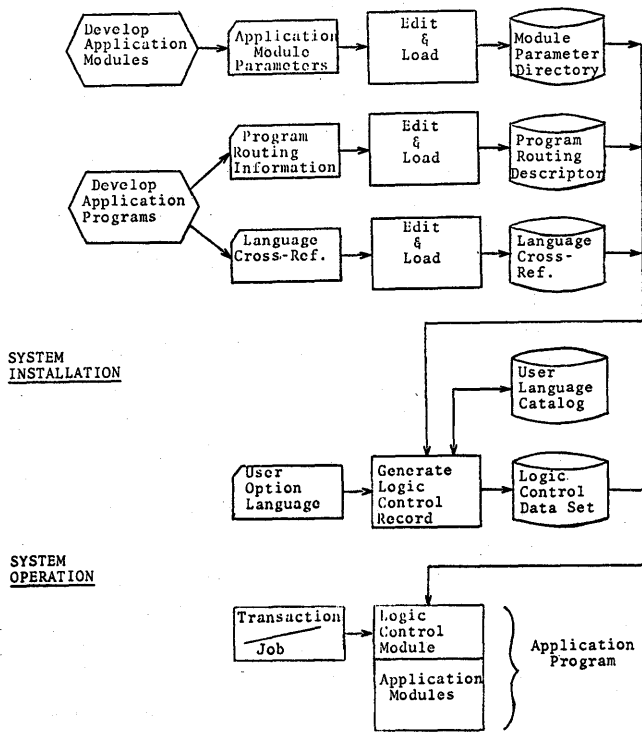


FIGURE 1—Logic control subsystem

eter is used. Parameters are not identified by the program which contains the module. This permits the same parameter information to be used when the module is utilized in more than one program. At the time of user input of processing specifications, the parameter identification is employed to ensure that the user value is assigned to the correct parameter. The parameter ID is also utilized during application program execution to identify the values passed to the modules within the program. The record format of the parameter directory is shown in Figure 2.

Module ID	Parameter ID	Data Type	Parameter Length & Precision	Count	Default Value	Low Value	High Value
						(or)	
						List of Valid Values	

Repeated for each parameter in module

FIGURE 2—Parameter directory record

and stored depends on the degree of module generalization.

The parameter definition information provided by the programmer includes the following—

Module ID—the identification of the module containing the parameter.

Parameter ID—the unique parameter identification within the module. This will be referenced later by a user specification.

Data Type—indication of character, decimal, binary, etc.

Length & precision—indication of the length and precision of the parameter.

Count—the number of values which the user may assign to the parameter.

Range—valid high and low limits for numeric data.

Valid values—valid values for character data.

Default value—value to be assigned to the parameter when no user value is specified.

Parameters are identified by the parameter ID and by the module in which the param-

- b. When the individual modules are combined into application programs, processing sequence information is input to the logic control subsystem. These routing rules indicate the range of choices in which processing is permitted to proceed from one module to the next within a program. The actual sequence is determined by both execution time decision and as the result of user option selection.

A typical program layout is shown in Figure 3. The resulting program load module consists of standard and optional modules which are user selectable. If specified by the user, optional modules will be used in place of or in addition to standard modules.

All modules return completion codes after execution. The completion code is used by the logic control module to determine the next module to execute. At execution time, if an optional module has been specified, control will pass to the optional module rather than the standard module which had originally been in its place.

Each module in a program is assigned a node identification. The node ID is used by

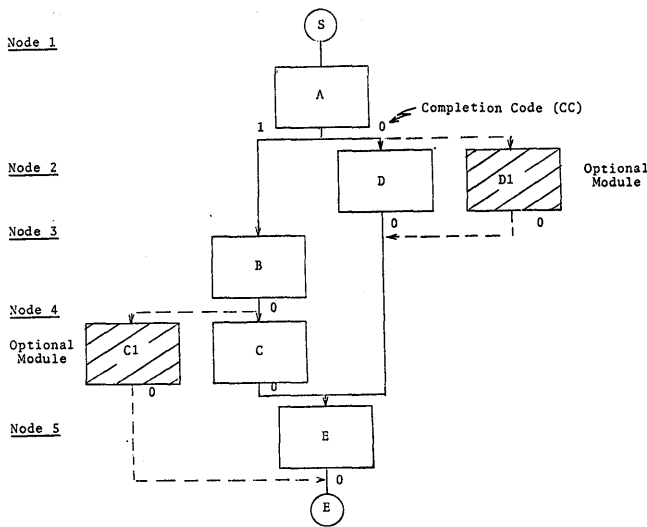


FIGURE 3—Typical program

the programmer to indicate which portions of the standard program may logically be replaced by optional modules.

When one or more option modules can replace an unlike number of standard modules, dummy modules are included in the routing information in order to maintain a one to one correspondence between optional and standard modules. This facilitates module replacement and ensures that information is not included in the logic control record for modules which will not be used during execution.

The programmer routing information input to the system is composed of the following items.

Program ID—program name

Standard Routing:

Node ID—unique node number within the program

Module ID—name of the module which is assigned to the node

Completion Code—completion code returned by the module at execution time

Next Node—next node to gain control for the paired completion code. This also indicates whether or not the calling module expects control to be returned at the completion of execution of the module at the next node.

Optional Routing:

Option ID—a unique option identifier which will be referenced by a user specification.

Node ID—node at which this module will replace a standard module.

Module ID—name of the optional module
Completion Code—same as standard routing

Next Node—same as standard routing

The resulting routing descriptor record shown in Figure 4 is generated for the program illustrated in Figure 3.

- c. The development of the user option language is a joint responsibility of systems and user personnel. This definition is simply a list of user keywords cross-referenced to the module processing parameters and routing options which they affect. A single keyword may be cross-referenced to one option in one module or to a number of options in modules contained in different programs. This allows a user specification

Standard Routing						
Program ID	Node	Module ID	CC	Next Node	CC	Next Node
Prog. X	1	A	0	2	1	3
	2	D	0	5		
	3	B	0	4		
	4	C	0	5		
	5	E	0	End		
Optional Routing						
Option ID	Node	Module ID	CC	Next Node		
01	2	D1	0	5		
02	4	C1	0	5		
	5	Dummy	--	End		

FIGURE 4—Routing descriptor record

to have as broad an affect as desired.

The user takes an active part in defining the keywords which make up his language. This permits him to input option specifications in terminology familiar to him. The user option language is completely independent of application programming since its definition consists merely of a cross-reference. This means that it can be changed and expanded with little effect on the application programs. In addition, the logic control system processing required to handle user specifications is minimized. There is no unique processing required for each keyword specification since all user input is handled through the cross-reference list.

The sequence in which keywords are listed in the cross-reference data set determines the order of keyword processing at user specification time. This is normally a strictly sequential mode of processing. To vary the processing sequence, the programmer may define "instructions" to the logic control subsystem. These instructions are included in the cross-reference data set and allow non-sequential processing at the time user specifications are read into the system. The instructions provide the ability to define parameter subsets and hierarchies.

A conditional branch type instruction causes bypassing of keywords in the cross-reference data set when certain values are specified by the user. For example, "if KEYWORD 1 = VALUE 1, go to KEYWORD 10, otherwise continue with KEYWORD 2." A looping type instruction allows a set of keywords, for instance KEYWORD 1 through KEYWORD 10, to be repeated by the user with different values specified each time.

The language definition and resulting cross-reference data set consist of the following elements as shown in Figure 5.

keyword—user language keyword
 module and parameter ID—list of all parameters in the parameter directory which are affected by this keyword.

(or)

program and routing option ID—list of all routing options in the routing descriptor data set which are affected by this keyword.

loop indicator—indicates the start or end of a group of keywords.

number of loops—indicates the maximum number of times the group of keywords may be repeated by the user.

value and next keyword—indicates the next keyword to be processed if the user inputs the specified value.

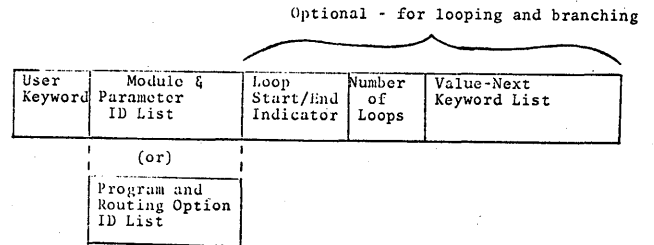


FIGURE 5—Language cross-reference record

Phase 2—System Installation

At local installation of the information system for a specific process or product, the user inputs his local processing options via the user option language. The user specifications, in conjunction with the previously developed data sets, are used to generate logic control records. These records are used to control the processing of the application programs at execution time. The user specifications are also placed in a catalog, to be referenced later when updates are required to a previous specification.

Logic control records, therefore, contain user selected subsets of the options defined in the parameter directory and routing descriptor data sets. These values are selected by keywords which point to the desired options through the cross-reference data set. Values are also selected by default, when the user omits keywords which are not required in the input stream.

Each logic control record is keyed by program ID and user ID. At execution time, inputs which invoke the application program contain the same key, allowing the correct logic control record to be accessed for the particular use of the program.

The resulting logic control record, as shown in Figure 6, contains the following elements.

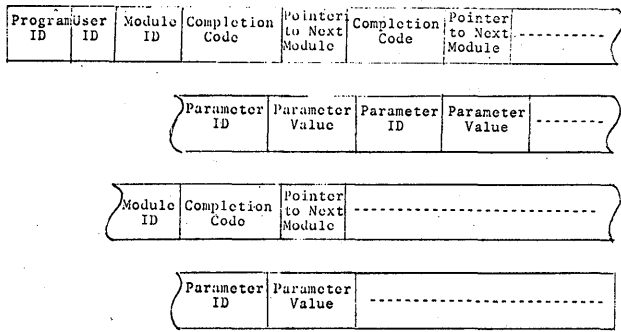


FIGURE 6—Logic control record

Program ID—program identification

User ID—identification of the program user. This element and the program identification form the record key.

Module ID—identification of modules within the program.

Completion code—completion codes returned by the module at completion of execution.

Next module pointer—pointer to the next module to execute for the corresponding completion code.

Parameter ID—unique identification of parameters used by the module

Parameter value—values of the parameters used within the module.

Application modules access the parameters in the logic control record by parameter ID rather than by the parameter position in the record. This simplifies addition and deletion of parameters used by the module, since changes in position do not affect accessing of the parameters. Additionally, parameters are separated by module in the logic control record. Therefore, parameter changes for one module will not affect parameter accessing in other modules of the program.

Phase 3—System Operation

Each application program contains a logic control module which serves as the interface between the logic control data set and the application modules within the program. The control module is a common module which is independent of the application program being controlled. It provides control

in either a background or real-time mode. When the application program is invoked, control is initially passed to the logic control module. The user identification contained in the input is used to access the correct logic control record. The control module determines the first module which will execute, as specified in the control record, and gives it control. At this time the module receives its parameters as contained on the record. The parameter values, in effect, customize the function performed by the module, as previously specified by the user.

When the application module completes processing it returns control to the logic control module and passes back a completion code. This code is matched against all completion codes which may be returned by the module. When a match is found, the pointer to the next module, contained on the logic control record, is used to determine the next module for execution. Control and parameters are passed to this module and the process continues until program completion. The application program layout is illustrated in Figure 7.

The fact that control information is not accessed and used until the program is invoked means that the program remains generalized until execution time. This permits an individual program to serve many users, each with varying requirements. Users do not require individual systems and programming attention, since re-

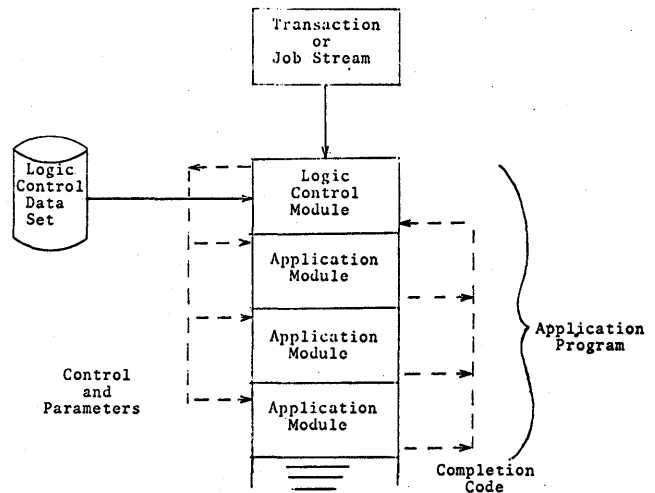


FIGURE 7—Application program layout

quirements are specified through the user option language and stored until needed. The logic control subsystem, which is independent of the ap-

plication being controlled, provides the necessary support to control the specification and utilization of processing options.

B-LINE, Bell line drawing language

by AMALIE J. FRANK

Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

INTRODUCTION

General Description

Over the past few years increasing interest has been shown in the application of digital computers in the graphics arts and publishing industries. Considerable effort has already been made in developing systems for the editing and publishing of text. Early work resulted in the formulation of algorithms for hyphenation and justification, followed by systems for page composition and correction of text stored internally within the computer system. Initially, the output function of these systems was to control a conventional hot-lead typesetting device. More recently, systems have been designed to control the formation of images on the face of a cathode ray tube (CRT). An image thus displayed is captured by a camera aimed at the CRT, and the resulting film is used to produce plates or mats for off-line volume printing. Systems of this type have been successfully implemented and are in full operation, as for example the MACE¹ (Machine-Aided Composing and Editing) system in use at Bell Telephone Laboratories, and the PAGE1² (Page Generation) system developed by the RCA Graphic Systems Division.

Paralleling the need to automate the handling of text, is the need to produce graphic arts quality line drawings with an equal amount of ease and economy. This is of particular concern in the production of technical publications. To date, a number of computer programs have been developed to produce line drawings on a CRT for specific applications, as for example the AUTODRAFT³ system for engineering drafting and design, and the XYMASK⁴ system for generation of integrated circuit masks. In addition, various FORTRAN subroutines have been written, as for example TPLOTT⁵ for drawing graphs, and the graphic subroutine package proposed by the SHARE Standard Graphic Output Language Committee.⁶ At present, a need exists for a common computer language facility aimed at the production of graphic arts quality line drawings. In

order for such language to be useful, it must have certain basic properties.

First of all, this language must be concise and easily learned. It should permit the user to specify the various features of a drawing in the natural order in which they occur to him and in a continuous stream rather than in segmented form in separate statements. For publication purposes, it must give the user direct and ultimate control of every line drawn if he so desires. Yet, where applicable, the user should be able to cause a particular version of a whole superstructure to be generated by the system merely by specifying a few simple options. Toward this end, the language should include the facility to construct higher level statements from the basic language statements. It is envisioned that a set of such "user defined" statements could be developed by an experienced programmer for a particular application. Once defined, such statements could then be used by non-programmers without knowledge of their genesis. Preferably, the language should meet the needs of users of widely varying computer experience. At one end of the scale it should appeal to a user essentially untrained in computer programming for the simple transcription of drawings from a rough draft. At the other end of the scale it should satisfy a user desiring to generate pictures controlled by algorithm at execution time. Drawing on a conditional basis is particularly attractive for applications such as circuit drawings and the production of musical scores. Finally, the implementation of this language should readily accommodate minor changes in syntax dictated by user experience. In addition, it should be designed to run easily on a variety of computers, and hopefully on a variety of terminal CRT systems, such as the Stromberg Carlson 4060 or the RCA Videocomp.

The B-LINE language was designed to meet the requirements indicated above. Initial application of the language is to be made to the production of the illustrations in the Bell System Technical Journal. The remainder of this paper is devoted to a description of the

B-LINE language. The following section summarizes the general features of the language. A later section describes the composition of draw-strings in greater detail. The last section contains a summary of the basic statements.

General features

The B-LINE language includes a set of eight basic statements, which describe the drawing features to the system in textual form. Included is the facility for the user to define other graphic statements, germane to a particular application. In addition, the user may employ any of the FORTRAN IV statements in admixture with the graphic statements, thus supplying the usual arithmetic and control functions. The total input describing a picture is a collection of B-LINE basic, B-LINE user-defined, and FORTRAN statements. In any case, this input need not be a complete FORTRAN program.

The formats of the eight basic statements are shown in illustration 1; they are summarized in the next section. In brief, these statements perform the following functions:

DRAW describes a part of a drawing by means of a draw-string, consisting of a string of elements which indicate a sequence of drawing functions. The effect of *DRAW* is to resolve the draw-string into a set of line segments to be drawn.

DEFINE gives the definition of a graphic variable. It is roughly equivalent to a FORTRAN assignment statement. However, the value of a graphic variable is a

string of elements, which themselves comprise a draw-string.

ERASE causes a set of line segments, which had been previously specified to be drawn, to be deleted.

EXPAND gives the definition of a symbol, which is used in place of a contiguous part of the input describing a picture. It is used to minimize preparation time where it is anticipated that a set of pictures of a similar nature are to be drawn and where certain parts of the input are the same for each member of the set.

OPFORM indicates the structure of a user-defined statement or a graphic function.

SIZEOUT, *BORDER*, and *SIZIN* primarily establish the necessary scaling factors.

The various statements describing a drawing are composed of characters from the *standard character set*. This set consists of 64 symbols, as defined in Appendix A. Each of the graphic statements, basic or user-defined, consists of an optional label, followed by the name of the statement, followed by a set of arguments. Labels and statement names are to conform to FORTRAN standards for labels and variable names respectively. A statement must be followed by a space. Spaces are significant only following a statement name or within a string of characters representing text to be drawn. Other spaces are ignored by the system and may be used freely to give visual separation. At present, input is on punched cards. Two successive statements on the same card are separated by a semi-colon. In each graphic statement there are a fixed number of *initial* arguments. In some statements, such as the *DRAW* statement, the *terminal* argument itself consists of a string of elements, which indicate a sequence of drawing functions. A string of this type is called a *draw-string*. Commas are used to separate initial arguments. A colon is used to separate the last initial argument and a terminal draw-string.

The elements of a draw-string may assume various forms: a coordinate pair, a symbol for a graphic variable, a graphic expression, a text element, a range function call, a graphic subroutine call, or a code for a control operation. These are discussed in detail in section 3 below. The various elements of a draw-string may be freely intermixed, thus permitting the user to specify the various features of a section of the drawing in a continuous stream. Drawing data, such as coordinates defining line segments, names of variables or predefined substructures, text, and control parameters like line thickness and font selection, may flow in the same order as they occur to the user. Concatenation proceeds naturally from the order established in a draw-string, i.e., the last point specified for a draw-string element becomes the concatenation point for the next element, unless superseded where desired. Concatenation points

BASIC GRAPHIC STATEMENTS	
Statement	Arguments
DRAW	NAME, XSCALE, YSCALE: STRING
DEFINE	SYMBOL: STRING
DEFINE	SYMBOL, ORIGIN, XSCALE, YSCALE: STRING
ERASE	TYPE, SKIN, XSCALE, YSCALE, X1, Y1, X2, Y2
EXPAND	SYMBOL, EXPANSION
OPFORM	NAME, RESULT, TYPE1, TYPE2, ...
SIZEOUT	BOTTOM, LEFT, LABEL, PAGE1, PAGE2
BORDER	LOCATION, SCALE, BASE, FIRST, LAST SCALAB, AXILAB, NODRAW
SIZIN	SIZE1, SIZE2

ILLUSTRATION 1

may also be variable. At execution time, a draw-string is resolved into a set of line segments defined in terms of CRT raster coordinates.

Of particular note is the handling of graphic variables. A graphic variable assumes two forms: a string variable or a point variable. A string variable is itself defined as a draw-string, which in turn may contain any of the elements listed above for a draw string, including other variables. A point variable consists of a pair of elements, representing the abscissa and ordinate of a point in the picture. A variable is assigned a symbol, which may be used in various contexts in the initial arguments of the graphic statements, or within a draw-string. A graphic variable may assume various values throughout the course of the drawing process. Thus, a drawing feature may be represented in variable form, and a specific form determined by algorithm at execution time. This is in contrast to describing each part of the drawing explicitly in the input data. Variables are discussed further in the following section.

In many cases, the user needs to arrive at the position of various key points within the drawing, or such factors as the distance between two points, or the slope of a given line. These factors may be calculated manually, albeit somewhat laboriously, by the familiar techniques of analytic geometry. To relieve the user of this burden, the system includes the facility to perform *graphic arithmetic*, which manipulates entities representing points within the picture or scalar quantities. Expressions involving such entities may be used in various contexts in the initial arguments of a graphic statement or within a draw-string. Such expressions are evaluated at execution time by means of the graphic arithmetic facility. This is discussed further in a later section.

For a given application, considerable economy can be effected by incorporating user-defined statements. To do so, the user first determines the actions that are to be performed when the statement is used, and gives a name to and determines the form of the arguments of the statement. The user then writes a subroutine bearing the name of the statement and performing the indicated actions. In the main body of input describing the picture, the user includes an OPFORM statement declaring the name and format of the statement. The user is then free to use the indicated statement in the main body of input. A user-defined statement may be used as a separate graphic statement or as a graphic subroutine call element in a draw-string. Either type of usage results in a call to a subroutine bearing the name of the statement. Such a subroutine may itself be written using the basic graphic statements, other user-defined statements, and any FORTRAN IV statements. The arguments of a user-defined statement may assume

all of the forms permissible under FORTRAN. In addition, they may assume a number of other forms, and in particular the terminal argument may be a draw-string.

A set of user-defined statements have been defined for the production of graphs, and are outlined in illustration 10. User-defined statements are particularly applicable where a class of structures to be drawn can be specified in generalized form by means of a user-defined subroutine. In this case, the statement is defined so that at a particular use of the statement, the arguments supply parameters which cause a particular version of the structure to be drawn.

Implementation of the language is made using a macro-processor written in the BSTRING⁷ language. BSTRING is a string language, with special features added to facilitate macro-generation. Its syntax is somewhat less pleasant than, but more powerful than SNOBOL, in particular permitting arbitrary recursion within statements.

The macro-processor ingests the input describing a picture, and treats each statement as a macro-instruction, which either expands into a sequence of FORTRAN statements, or causes the macro-processor to use the indicated arguments to control the translation of subsequent graphic statements. For the basic graphic statements the code generated consists essentially of a call or a series of calls to a collection of system subroutines. For example, the appearance of a coordinate pair (2, X) as an element of a draw-string of a DRAW statement results in a call to a system subroutine, which performs the following minimal actions. The values of the constant 2 and the variable X are converted to raster coordinates, if required. The converted values are incorporated into an instruction as required by the output CRT device, and the instruction is added to an output stack. The code generated for a

USER DEFINED OPERATIONS FOR THE PRODUCTION OF GRAPHS		
Operation	Arguments	Feature drawn
LNGRID	BORDER, MAJORDIST, MAJORTYPE, MINORQUAN, MINORTYPE	linear grid
LOGRID	BORDER, BASE, MAJORTYPE, (MINORMULT, MINORTYPE)	log grid
SCALAB	BORDER, (LABEL)	scale labels
SCALABA	BORDER, LABEL, INCREMENT	
DENTAL	BORDER, TYPE, DIST, XSCALE, YSCALE, .STRING	incidental grid line, tick, scale label
AXTLAB	BORDER, POSITION: STRING	axis label
AXTLABA	BORDER, DIST: STRING	
LEGEND	CORNER, X, Y, WIDTH, HEIGHT, LINES, BORDER	legend area
LEOTXT	ALIGN: STRING	legend line of text
UNITXT	CORNER, X, Y: STRING	single line of text
DPOINT	TYPE, CURVE, RADIUS, (X, Y)	data point
LEADER	WISE, RADIUS, XSCALE, YSCALE, X1, Y1, X2, Y2, CURVE, X3, Y3, X4, Y4	leader
LDIMEN	SIDE, X1, Y1, X2, Y2: STRING	linear dimension
ADIMEN	SIDE, X0, Y0, X1, Y1, X2, Y2, RADIUS: STRING	angular dimension

ILLUSTRATION 10

user-defined statement consists of a call to the subroutine, which bears the name of the statement. In the case of either a basic or a user-defined statement, a certain amount of in-line code may also be generated to manipulate the arguments prior to calling the indicated subroutine, or to indicate dimensions, perform initializations, provide missing organizational statements, etc. FORTRAN statements appearing in the input are copied directly into output FORTRAN statements.

The total output of the macro-processing phase is a complete FORTRAN program or subprogram, as applicable. The resulting FORTRAN program unit is then translated by a FORTRAN compiler into a machine program. Upon execution, the machine program stores the instructions for the CRT device onto a tape or other storage medium. The tape is used to drive the CRT device off-line. The machine program may also be written to drive the CRT device on-line where applicable. Note that only those user-defined subroutines referenced in a particular run need be bound in the object program for that run.

Draw-strings

A draw-string consists of a string of elements which indicate a sequence of drawing functions. Draw-strings appear as the terminal argument of the DRAW and DEFINE statements. In addition, a draw-string may appear as the terminal argument of a user-defined statement. The elements of a draw-string may assume various forms: a coordinate pair, a symbol for a graphic variable, a graphic expression, a text element, a range function call, a graphic subroutine call, or a code for a control operation. Immediately below is a preliminary discussion of the handling of these elements with respect to origin, scaling, and concatenation. Following this is a description of the various types of draw-string elements.

Drawings are composed with reference to a pair of scaled axes. The rectangular area described by these axes is called the *picture proper*, and the lower left-hand corner of this area is called the *global origin*. The size and placement of this area of the CRT is determined by means of the SIZOUT statement. In general, coordinate values in a draw-string are given with respect to the global origin. In some cases, an alternate origin is used. This is discussed further below.

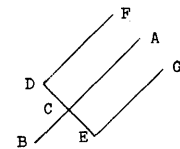
The scaling of the elements of a draw-string may be given by the user in one of four ways: in terms of a data scale, in inches as measured on an existing copy, in inches in the final output print, or in CRT raster positions. For the DRAW statement, the scaling of the elements given in the draw-string is primarily indicated by means of the XSCALE and YSCALE initial arguments. However, for an element of a draw-string that is a variable, the scaling may also be indicated in the

DEFINE statement defining the variable. Where this is done, the scaling so indicated supersedes the scaling indicated in the XSCALE and YSCALE arguments of the DRAW statement. For user-defined statements, the scaling of elements given in a terminal draw-string is defined by the user.

At various times in processing a draw-string, the system stores the coordinates of the ending point of the last drawn line segment in the system variable *CAT*, called the concatenation point. This variable consists of a string of two elements, one representing an X coordinate, and the other a Y coordinate, relative to the global origin. This variable may be referenced in the same manner as a user assigned point graphic variable. An example of such a usage is contained in illustration 8. The value of *CAT* at any particular point may be saved by the user for later reference. This may be done by means of a DEFINE statement, which sets a user assigned variable equal to *CAT*. It may also be done within a draw-string by means of a set control operation, as described below. At the start of processing an element in a draw-string, the concatenation point has a particular value. The drawing of an element is related to the value of the concatenation point. An element, in turn, may cause the system to change the value of the concatenation point.

The system includes the option to process any particular draw-string on an *immediate* or on a *deferred* basis. Where drawing is done on an immediate basis, the

EXAMPLE OF GRAPHIC ARITHMETIC



PROBLEM

GIVEN: Graphic point variables A and B

DRAW: AB

GE || AB and length = 2/3 AB

EC ⊥ AB and length = 35 units

CD ⊥ AB and length = 35 units

DF || AB and length = 2/3 AB

SOLUTION

DRAW FORK, XDATA, YDATA: A, B, \$S,

PØL(A, ANG(A,B) + 90, 35),

PØL(CAT, ANG(A,B), DIS(A,B)*2/3),

PØL(CAT, ANG(A,B) - 90, 70),

PØL(CAT, ANG(B,A), DIS(A,B)*2/3)

ILLUSTRATION 8

draw-string is resolved into a set of raster coordinates which are outputted to cause the indicated line segments to be drawn. Where drawing is done on a deferred basis, the draw-string is likewise resolved into a set of raster coordinates, but they are not actually outputted. However, all of the control operations imbedded in the draw-string, and in particular, the set control operation, are performed. Drawing on a deferred basis is used in determining distances required for composition, particularly for the placement of text.

The various types of elements of a draw-string are discussed below.

(a) *A coordinate pair*

This element is of the form (X, Y) , where X and Y represent an abscissa and ordinate relative to the global origin. X and Y may individually be an explicit numerical value, or a FORTRAN real or integer variable or expression, or a graphic expression. An explicit numerical value takes the form of an optional sign, followed by a sequence of decimal digits with or without a decimal point. A graphic expression is one using one or more of the graphic operators, or referencing a graphic function. Graphic expressions are discussed further at point (c) below. In the present context, where a graphic expression is used as a component of a coordinate pair, it is assumed to yield a single value.

The effect of a coordinate pair is to draw a line connecting the concatenation point, CAT, to the point described by the coordinate pair, except where the preceding draw-string element is a space control operation, \$S. In any case, the final action is to change the value of CAT to the value of the coordinate pair. An example of coordinate pairs is shown in illustration 2, where the components of the coordinate pairs are all explicit numerical values.

(b) *A symbol for a variable*

A variable appearing as a separate element of a draw-string must be a graphic variable. A graphic variable takes two forms: a string variable or a point variable. These are discussed separately below. Where a variable is referenced as an element of a draw-string, the value of the variable is in effect substituted in place of the symbol for the variable. The value of a graphic variable may be null.

Point graphic variable

A point graphic variable consists of a pair of elements, representing the abscissa and ordinate

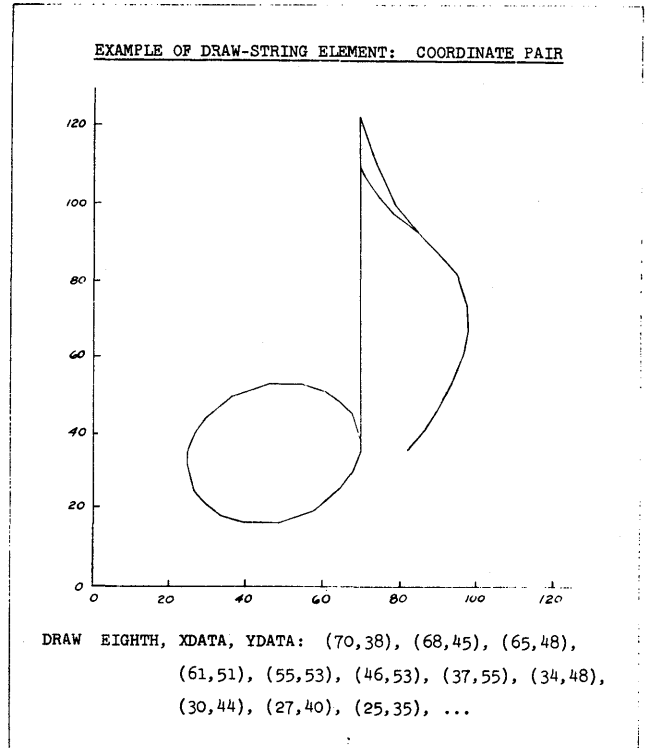


ILLUSTRATION 2

of a point in the picture. A point variable is represented by a FORTRAN symbol. The symbol for a point variable may be used as a separate element in a draw-string. It may also be used as an initial argument of a graphic function call, or a graphic subroutine call, or a graphic statement. In addition, a point variable may be referenced in any FORTRAN statement. In fact, in the FORTRAN program generated in the macro-processing phase, a FORTRAN complex variable is set up for each point variable. Point variables thus afford a bridge between the graphic statements and FORTRAN statements.

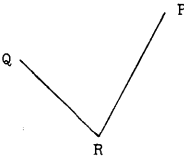
The value of a point variable may be changed by a FORTRAN statement, or by a DEFINE statement, or by a set control operation. At execution time, a point variable defined by a DEFINE statement is evaluated immediately at the appearance of the DEFINE statement.

Examples of point variables are contained in illustrations 7, 8, and 9. In illustration 9, LEFT and END are point variables.

String graphic variable

A string graphic variable consists of a draw-string, which may contain any of the elements described herein for draw-strings, including other

EXAMPLE OF GRAPHIC ARITHMETIC



PROBLEM

GIVEN: Graphic Point Variables P, Q, and R
 DRAW: PR or QR, whichever is shorter

SOLUTION

```

IF (DIS(P,R) - DIS(Q,R)) 5, 5, 10
5 S = P
GO TO 15
10 DEFINE S: Q
15 DRAW SHØRT, XDATA, YDATA: S, R
  
```

ILLUSTRATION 7

EXAMPLE OF SET CONTROL OPERATION

Desired Result:

THE RELATIVE MAXIMUM VALUE OF Ψ IS 3.

Input:

```

DRAW TEXT, XDATA, YDATA: $U, "THE ", $SET(LEFT),
"RELATIVE MAXIMUM", #BØX, " VALUE ØF ", &PSI,
" IS ", $SET(LEFT), "3", #BØX, " ."

DEFINE #BØX, GLØBAL, XRAST, YRAST: $SET(END),
$S, LEFT + (-4,4), LEFT + (-4,-34),
END + (4,-34), END + (4,4), $S, END
  
```

Variables LEFT and END:

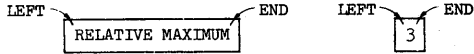


ILLUSTRATION 9

variables. A string variable is identified by a symbol starting with a # and followed by any combination of characters from the standard character set, except the arithmetic symbols and the punctuation symbols used as delimiters. The symbol of a string variable may not be sub-

scripted. A string variable is used only as a separate element in a draw-string. A string variable may not be referenced in any FORTRAN statement.

The value of a string variable is set by a DEFINE statement and may vary throughout the course of the drawing program. The number of elements in the defining draw-strings may vary from one definition to the next for the same variable. At execution time, a string variable is evaluated not at the appearance of the DEFINE statement defining the variable, but at the execution of a DRAW statement referencing the variable.

The origin and the scaling of the coordinates in the draw-string defining a string variable may be specified to be the same as the parent draw-string referencing the variable, or, they may be specified independently in the DEFINE statement defining the variable. In the second case, the coordinates may be given with reference to the global origin. They may also be given with reference to an *hypothetical* origin. In this case, the variable is called an *image*, which may be "floated" to various positions in the picture. At a particular use of the variable in a draw-string, a point, P, within the picture is established. The hypothetical origin of the image is superimposed on the point P, and the image is copied into the picture about this origin. The position of point P is specified by means of the ORIGIN argument of the DEFINE statement which defines the variable. P may be specified to be the concatenation point, CAT, or the value of a coordinate pair given in the form indicated in (a) above, or the value of a graphic variable or expression which resolves into two numeric values. The X and Y coordinates of point P may be considered as relocation factors. The X factor is added to all X coordinates in the draw-string defining the variable, and the Y factor is added to all of the Y coordinates. Provision is also made to rotate and size images.

An example of a variable which is an image is shown in illustration 3. It is to be noted that the first line segment drawn in an image need not emanate from the concatenation point. A slight change in the placement of the hypothetical origin will cause a different effect, as shown in the illustration 4. Note that in both cases, in the DEFINE statement for the image #FLAG, the ORIGIN argument is CAT. Accordingly, for the first reference of FLAG in the DRAW statement,

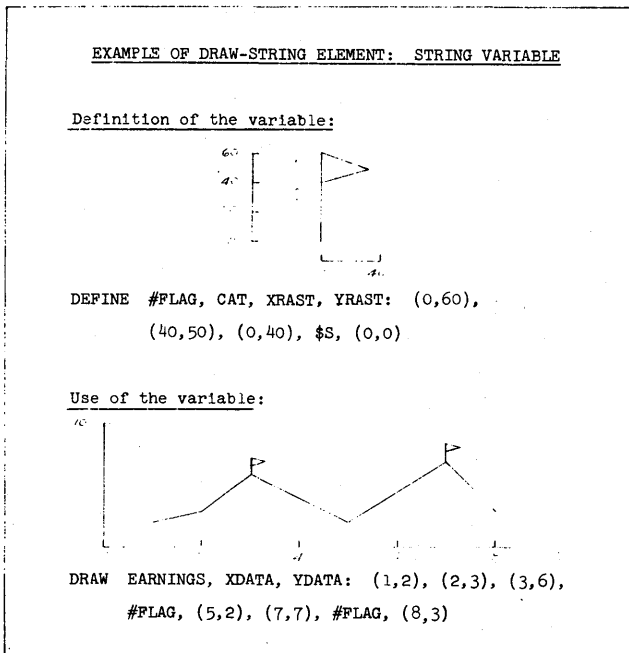


ILLUSTRATION 3

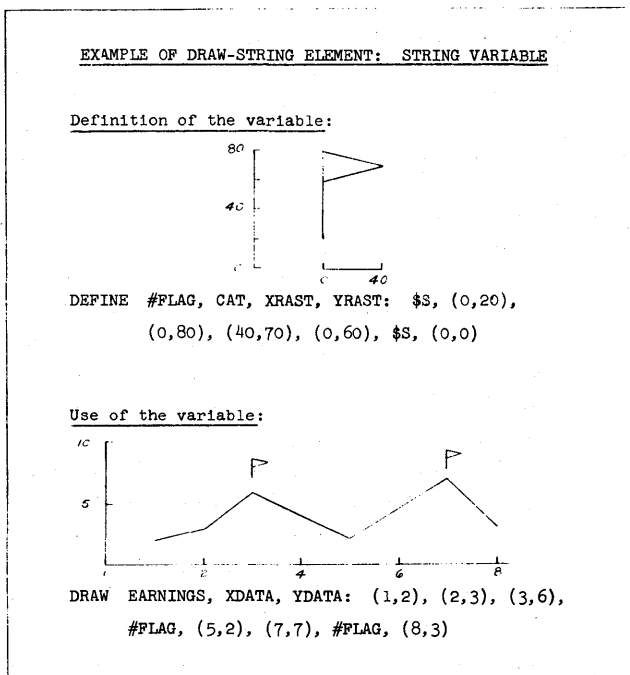


ILLUSTRATION 4

the point P, as indicated above is (3, 6), and for the second reference P is (7, 7).

An image may be used in building another image. A few simple examples, are the building of a prism from a triangle, of an eighth note from

a quarter note, of a magnetic core inductor from an inductor, as shown in illustration 5.

In the illustrations indicated above, the definition of the string variable yields a fixed image, An example of a string variable which varies depending upon the context in which it is referenced is shown in illustration 9. Note that in this case in the DEFINE statement for the image #BOX, the ORIGIN argument is GLOBAL, thus setting the point P, as indicated above, at (0, 0).

(c) *A graphic expression*

A graphic expression is one using one or more of the graphic operators, or referencing a graphic function. The graphic operators invoke the system facility to perform "graphic arithmetic" at execution time. This arithmetic manipulates operands which are ordered pairs and in some cases scalars. An ordered pair represents the coordinates of a point within the picture. A number of graphic functions are supplied by the system. Provision is also made for the user to define other graphic functions. A graphic expression may also contain the usual algebraic operators and FORTRAN function references. At execution time, the expression is evaluated, and the resulting value is in effect substituted in place of the variable. In the present context, where a

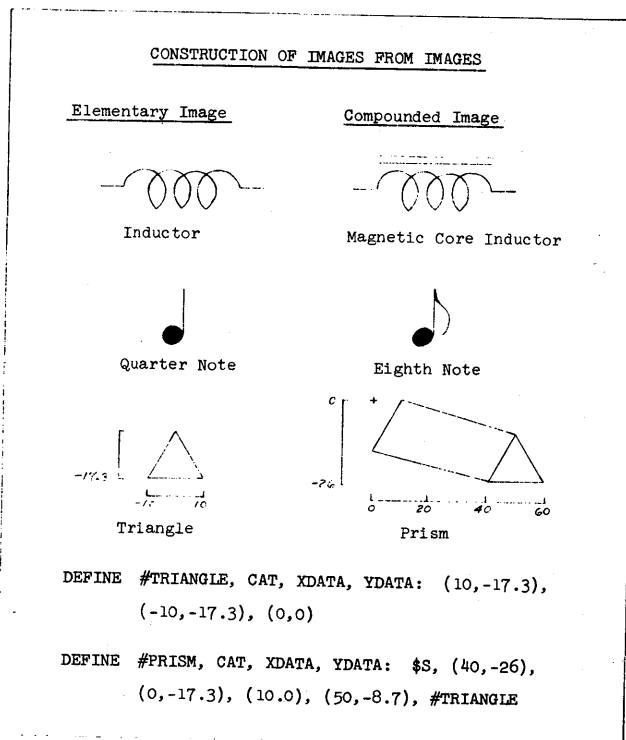


ILLUSTRATION 5

graphic expression is used as a separate element of a draw-string, it is assumed to yield an ordered pair.

The graphic operators and functions supplied by the system are indicated in illustration 6. The graphic operators parallel the arithmetic operators for complex numbers. For addition and subtraction, the operands represent points within the picture, or displacements from such points. For multiplication and division, in general one operand represents a point within the picture, and the other operand comprises rotation and sizing factors. The results of any of these graphic operations is an ordered pair.

The system graphic functions operate on a list of arguments each of which is an ordered pair or scalar as applicable. These functions are summarized as follows:

$ANG(P1, P2)$ gives the angle that the line joining the two points $P1$ and $P2$ makes with the horizontal, originating at $P1$ and extending to the right.

$DIS(P1, P2)$ gives the absolute distance between the two points $P1$ and $P2$.

$POL(P, ANGLE, LENGTH)$ gives the cartesian coordinate pair corresponding to the point

given in polar coordinate form. As shown in illustration 6, the components indicate a vector originating at the point P , and making an angle, $ANGLE$, with the horizontal extending to the right from P . The length of the vector is given by the argument $LENGTH$. The value of this function are the coordinates of the end point of the indicated vector.

$PER(P1, P2, P3)$ gives the coordinate pair which is the intersection of the perpendicular extending from a point $P1$ to a line, defined by two points $P2$ and $P3$.

$TAN(P1, P2, R, K)$ gives the coordinate pair which is the point of tangency of the line passing through the point $P1$ and tangent to the circle with its center at point $P2$ and of radius R . The parameter K indicates which of the two possible points of tangency is to be used.

The user may define other graphic functions either by means of a $DEFINE$ statement, corresponding to a FORTRAN arithmetic statement function, or by an external subprogram. In any case, the user must include in the main body of input describing the picture, an $OPFORM$ statement declaring the form of the arguments of the function and the number of elements in the result draw-string returned by the function.

Examples of graphic arithmetic are given in illustrations 7 and 8.

(d) *A text element*

A text element represents textual matter to be drawn. It takes two forms: a text string, and a special character symbol as discussed below.

The primary form of a text element is a text string, which is a sequence of characters drawn from the standard character set, preceded and succeeded by the double quote character. The standard character set consists of 64 characters, and is listed in Appendix A. For each character in this set, the system contains an internal definition of the line segments to be used to draw the character and additional factors for concatenating successive characters. In processing a text string, the system accesses the appropriate definition for each character in the string, and causes the indicated strokes to be drawn. The font style for the standard character set is News Gothic English.

In addition to the standard character set, the system includes a set of special character definitions for the Greek alphabet and various mathematical and other symbols. Associated with each of these characters is a system name starting with an $\&$ and followed by a combination

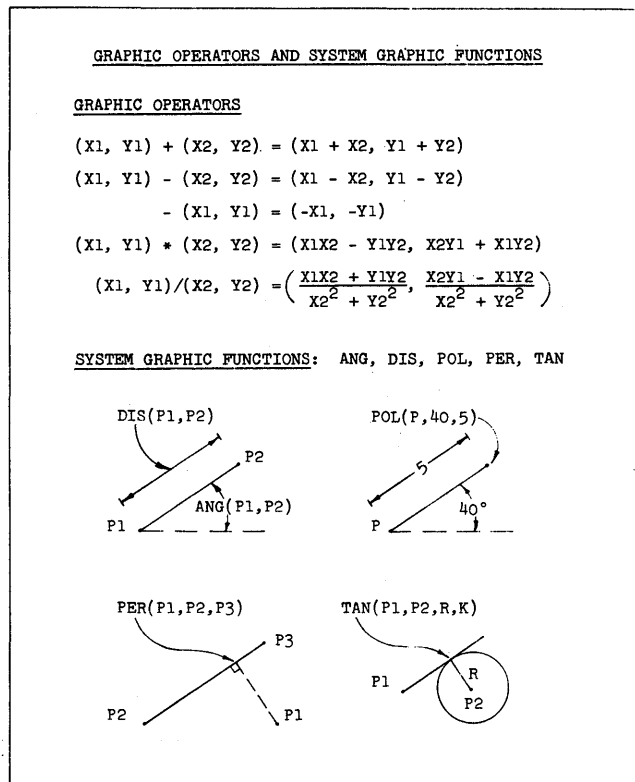


ILLUSTRATION 6

of characters from the standard character set except the arithmetic symbols, and the punctuation symbols used as delimiters. To cause a special symbol to be drawn, the user includes its system name as a separate element of a draw-string.

For the English alphabets, definitions are provided for upper and lower cases, in Roman, bold, and italic faces. For the Greek alphabets, definitions are provided for upper and lower cases. The case and face applicable for a text string or special character element is indicated by a preceding draw-string element that is a text control operation, and is explained below.

Standard and special characters are defined relative to an hypothetical origin. This origin is assumed to be generally to the left and above the character. Where a character is not preceded by another character, the hypothetical origin of the character is simply superimposed on the concatenation point which was defined at the end of the previous element in the draw-string. Where a character is preceded by another character, concatenation proceeds according to an algorithm which permits the spacing between any two characters to be a function of the spatial relationship between the particular characters involved.⁸ Provision is also made to rotate and size characters.

See illustration 9 for an example of the two forms of a text element.

(e) *A range function call*

In its simplest form, a range function call identifies the name of an independent variable, and states a function of the variable. The system evaluates the function for successive values of the independent variable within a specified range at appropriate small increments. The resulting sequence of coordinate pairs is substituted in the draw-string in place of the range function call. The function may be stated in non-parametric or parametric form. Where the non-parametric form is used, the range function call is stated in one draw-string element, consisting of an equals sign, followed by the function of the independent variable, X, followed by optional suffixes specifying the range, as shown in the example below. Where a range suffix is deleted, the function is evaluated over the entire range of the data scale.

$$= \text{LOG}(X) + \text{BTU}(X = 2, \text{UPPER})(Y = \text{A} + 6, \text{DIS}(P, Q))$$

When the function is stated in parametric

form, the range function call requires two successive draw-string elements, one for each parametric equation. The first element states a function of the parametric variable, P, and the second element states a function of the independent variable, X. The range of computation is specified for the parametric variable only. The range suffix is appended to the first element, and must always appear.

(f) *A graphic subroutine call*

This element is a user-defined statement imbedded in a draw-string. The only difference between a user-defined statement used as a separate graphic statement, and one used as an element in a draw-string, is that in the latter case the entire set of arguments is enclosed in parentheses. The action taken is the same in either case.

(g) *A code for a control operation*

Control operations cause the system to take internal actions relating to the handling of a draw-string. There are five control operations: set, dash-dot, weight, space, and text. These are explained individually below.

Set operation

The set operation consists of the characters \$SET followed by one or two arguments. The arguments are separated by a comma, and enclosed by one set of parentheses. The first argument is always the symbol for a point graphic variable. This operation causes the system to set the value of the variable to the current value of the concatenation point, CAT. The second argument is a code indicating the scale to be used in setting the value of the variable. The variable set by a set operation may be referenced at some later point in the drawing process, either in the same draw-string or in a separate context. This feature is provided because the user will not always explicitly know the exact position of the concatenation point unless he performs cumbersome manual calculations. This is particularly true where text or successive images are being processed. Illustration 9 shows an example of how the set operation is used to draw a rectangular box around part of a line of text.

Dash-dot operation

The dash-dot operation consists of the characters \$D either alone or followed by an argument

enclosed in parentheses. Where an argument is present, this operation causes the ensuing draw-string elements to be drawn in a pattern of dashes and/or dots. The argument is either a code indicating one of four standard patterns (long dashes; short dashes; dots; alternating dashes and dots), or it is a variable indicating a user-defined pattern. Where \$D appears alone, cursive drawing is resumed.

Weight operation

The weight operation consists of the characters \$W followed by an argument enclosed in parentheses. The argument is a code indicating lines of varying heaviness. This operation causes the ensuing draw-string elements to be drawn with the indicated line weight. In the absence of any weight operation the lightest weight is assumed.

Space operation

The space operation consists of the characters \$S, and causes a space to occur between the line segment indicated by the previous draw-string elements, and the following draw-string element. In effect, it causes the concatenation point CAT to be reset to the pair of coordinates specified by the following draw-string element.

Test operations

These operations consist of a \$ followed by an alphabetic character as shown below. They control the case and face settings for ensuing text elements in the draw-string. The case setting is applicable for both the English and Greek alphabets. The face setting is applicable for the English alphabets only. In the absence of a case setting, \$U is assumed. In the absence of a face setting, \$R is assumed.

Operation	Effect on ensuing text elements
\$U	Upper case
\$L	Lower case
\$F	First character upper case; succeeding characters lower case
\$R	Roman face
\$I	Italic face
\$B	Bold face

Summary of basic statements

Draw statement

DRAW NAME, XSCALE, YSCALE: STRING

This statement supplies a name to, and describes a part of a drawing, referred to as a structure.

NAME is a unique symbol, consisting of any combination of characters from the standard character set, except the arithmetic operators and the punctuation symbols used as delimiters. In addition the initial character may not be a #, &, or \$, which identify a string variable, a special character symbol, and a control operation respectively.

XSCALE indicates the scale used in specifying the X coordinates contained in the STRING argument. It may assume one of the codes XDATA, XCOPY, XFINE, XRAST, which correspond to the four ways in which the scale may be given: in terms of a data scale, in inches as measured on an existing copy, in inches in the final output print, and in CRT raster positions. XSCALE may also be null. In this case, XDATA is assumed.

YSCALE indicates the scale of the Y coordinates in the STRING argument. It may assume one of the values YDATA, YCOPY, XFINE, XRAST, or it may be null.

STRING is a draw-string, specifying the structure to be drawn. Provision is also made to specify the structure to be drawn in a separate set of cards prepared by subjecting an existing drawing to a digitizer.

Define statement

DEFINE SYMBOL: STRING

DEFINE SYMBOL, ORIGIN, XSCALE,
YSCALE: STRING

The DEFINE statement defines a graphic variable or a graphic function.

The first form of the DEFINE statement is used where origin and scaling data for the defining draw-string either are not applicable or assumes the same values defined for the parent draw-string containing the variable. This form is always used to define a point variable. It may be used to define a string variable where applicable.

The second form of the DEFINE statement is used where the origin and scaling data for the draw-string are stated explicitly in the ORIGIN, XSCALE and YSCALE arguments. This form of the DEFINE statement may be used to define a string variable only.

SYMBOL is the name of the graphic variable or graphic function and consists of any combination of characters from the standard character set except the arithmetic symbols and the punctuation symbols used as delimiters. In addition the initial character may not be & or \$ which identify a special character symbol and

a control operation respectively. An initial character of # is permissible, and identifies a string variable. Where the operation defines a graphic function, the symbol is followed by the dummy argument names, separated by commas, and all enclosed in parentheses.

ORIGIN indicates the origin of the coordinates in the *STRING* argument. *ORIGIN* may assume various values as follows:

- (a) the code *GLOBAL* to indicate that the coordinates are given relative to the global origin.
- (b) the code *CAT* to indicate that the coordinates are given relative to an *hypothetical* origin. At a particular use of the variable in a draw-string, the hypothetical origin of the image is superimposed on the current value of the concatenation point, *CAT*.
- (c) a graphic variable or expression which resolves into two numeric values.
- (d) the form of a coordinate pair (X, Y) as explained in section I, 3 above.

YSCALE and *XSCALE* take the same form as these arguments for the *DRAW* statement indicated above.

STRING is a draw-string defining the graphic variable or stating the algorithm defining a graphic function. Provision is also made to specify an image on a separate set of cards prepared by subjecting an existing drawing to a digitizer.

Erase statement

ERASE TYPE, SKIN, XSCALE, YSCALE,
X1, Y1, X2, Y2

This statement specifies a part of the picture to be erased.

TYPE indicates the type of entity to be erased and is coded with one of the values: *BOX*, *CIRCLE*, or a variable name, or the name of a structure specified by a *DRAW* statement. *X1*, *Y1*, *X2* and *Y2* give the various parameters of the entity to be erased. Where *TYPE* is a variable name, the area to be erased is described by a *DEFINE* statement defining the variable. Where *TYPE* is the name of a structure specified by a *DRAW* statement, the structure is erased starting at the point with coordinates *X1*, *Y1*, and ending with the point with coordinates *X2*, *Y2*.

SKIN indicates if the borders of the indicated area, as well as the internal area, or if only the internal area is to be erased.

XSCALE and *YSCALE* indicate the scale used in specifying the arguments *X1*, *X2*, *Y1* and *Y2*. *XSCALE* and *YSCALE* are coded in the same manner as indi-

cated for these arguments in the *DRAW* statement.

Expand statement

EXPAND SYMBOL, EXPANSION

This statement defines a symbol which is used in place of a contiguous part of the input describing a picture. The value of the symbol is substituted for the symbol at *macro-processing* time.

SYMBOL is the name of the entity begin defined. It is constructed as indicated for the *NAME* argument of the *DRAW* statement.

EXPANSION is the definition of the symbol.

This statement is used where a set of pictures of a similar nature are to be drawn, and where certain parts of the input specifications are the same for each member of the set. A common part is written once and assigned a name, *SYMBOL*. For a particular picture, the user includes the symbol at the appropriate place in the input. It is envisioned that a user may build a library tape containing the *EXPAND* definitions pertinent to his application.

Opform statement

OPFORM NAME, RESULT, TYPE1,
TYPE2, ...

This statement indicates the structure of the arguments of a user-defined statement or a graphic function. The code generated by the macro-processor for a user-defined statement or graphic function consists essentially of a call to a subroutine which bears the name of the statement or function. The arguments of the *CALL* statements are derived from the arguments of the user-defined statement or graphic function. Depending upon the type of the argument of the user-defined statement or graphic function, the macro-processor takes different actions to set up the corresponding argument of the *CALL* statement. The *OPFORM* statement declares to the macro-processor the number of arguments, and the type of each argument. For a graphic function, the *OPFORM* statement also indicates the number of elements in the result draw-string return by the function.

NAME is the symbol of the user-defined statement or graphic function. This is constructed as indicated for the *NAME* argument of the *DRAW* statement.

RESULT is the number of elements in the draw-string returned by a graphic function.

TYPE1, *TYPE2*, ... declare the types of successive arguments in the user-defined statement or graphic function, as indicated below.

TYPE code	Form of Argument
FOR	FORTRAN numeric or logical constant, variable, expression
GR1	Graphic expression yielding 1 argument
GR2	Graphic expression yielding 2 arguments, or graphic point variable, or a coordinate pair as described above
LIT	Literal constant
STA	Executable statement number or FORMAT statement number
NAM	External function or subroutine name or NAMELIST name
STR	Draw-string

Any contingent set of TYPE arguments may be applied repetitively by enclosing them in parentheses and preceding by a repeat count. A terminal set of TYPE arguments may be repeated as many times as is required at a particular use of the user-defined statement by enclosing them in parentheses without a repeat count.

Provision is also made to specify default values for null arguments.

Sizeout statement

SIZEOUT BOTTOM, LEFT, LABEL, PAGE1, PAGE2

This statement causes the system to determine a subsection of the CRT within which to compose the drawing.

BOTTOM and *LEFT* are the dimensions in inches of the "drawing area" of the output picture to be produced. This area does not include page margins.

LABEL is coded LABIN, LABEX, or LABNO respectively for the cases where the drawing area indicated above does or does not include space for scale and/or axis labels, or where there are no labels.

Provision is made to handle two types of drawing situations: a "stand-alone" drawing which bears no relationship to any other drawings, and "series" drawings, which comprise a set of drawings to appear within the pages of a single volume. The arguments *PAGE1* and *PAGE2* are used for series drawings only and specify the maximum drawing area of a page in the volume. For stand-alone drawings, the system determines a rectangular CRT area, A, called the *picture proper*. The bottom and left borders of this area comprise a pair of scaled reference axes, intersecting in a point called the *global origin*. The scaling of the reference axes is determined from the appropriate BORDER statements. In general, coordinate values in a draw-

string are given relative to the global origin. Where they occur, scale and axis labels appear exterior to the area A. For a series drawing, the system first determines a CRT area, T, corresponding to the maximum drawing area of a page in the volume. The system then determines the area A to be the appropriate subsection of area T. By following this procedure, the same enlargement ratio is used to form prints from any microfilm frame in the series. This insures that the line weights are uniform for all of the drawings in the series, and limits the number of standard type sizes required. Area A is determined to usurp the maximum CRT area possible to obtain the finest resolution possible.

Border statement

BORDER LOCATION, SCALE, BASE, FIRST, LAST, SCALAB, AXILAB, NODRAW

This statement specifies the range of the data scale for a border of the *picture proper* and indicates if the border is to be drawn. This statement may also give an indication of the space required for the scale and axis labels along the indicated border.

LOCATION indicates which border is being specified and is coded with one of the values: BOTTOM, LEFT, TOP, RIGHT.

SCALE indicates if the scale is linear or logarithmic, and is coded LIN or LOG as applicable.

BASE is the base of the scale if it is logarithmic. *FIRST* and *LAST* give the initial and terminal values in the data scale of the indicated border. They must be specified for at least one of the borders TOP or BOTTOM, and at least one of the borders LEFT or RIGHT.

SCALAB indicates the number of characters in the longest scale label along the indicated border.

AXILAB indicates the number of lines in the axis label along the indicated border.

NODRAW is coded NODRAW if the indicated border line is not to be drawn. Otherwise it is null.

Sizing statement

SIZIN BOTTOM, LEFT

This statement is used where an existing drawing is being used as a model for the drawing to be made by the system. Various structures in the existing drawing are subjected to a digitizer. This statement, in conjunction with the SIZEOUT statement, indicates to the system how to convert the digitized data into raster positions.

BOTTOM and *LEFT* indicate the dimensions in inches of the *picture proper* of the existing drawing. They are assumed to cover the same range of the data

scale, as specified in the BORDER statement above. Note that the physical proportion of the width to height of the area in the existing drawing need not be the same as that for the CRT area.

APPENDIX A

Standard character set

<i>Symbol</i>	<i>Description</i>
A through Z	Alphabetics
0 through 9	Numerics
+	Plus
-	Minus
/	Virgule (Divide, Slash)
=	Equals
*	Asterisk (Multiply)
\$	Dollar Sign
.	Period
,	Comma
(Open Parenthesis
)	Close Parenthesis
'	Apostrophe (Close Single Quote)
"	Double Quote
:	Colon
;	Semi-colon
&	Ampersand
!	Exclamation
?	Question Mark
-	Short Dash
—	Long Dash
#	Number Sign
%	Percent
[Open Bracket
]	Close Bracket

<i>Symbol</i>	<i>Description</i>
^	Logical And
⌋	Logical Not
<	Less Than
>	Greater Than
	Space

REFERENCES

- 1 M V MATHEWS J E MILLER
Computer editing typesetting and image generation
AFIPS Conference Proceedings Volume 27 Part 1 1965 Fall
Joint Computer Conference p 389
- 2 70/9300 *composition language system*
RCA Graphic Systems Division Dayton New Jersey March 1968
- 3 O D SMITH H R HARRIS
Autodraft
Proceedings of the SHARE Design Automation Workshop
June 1965
- 4 S PARDEE
XYMASK—A system for generating microelectronic circuit masks
Unpublished memorandum
May 1967
- 5 J F KAISER E J SITAR
A versatile subroutine TPLOT for automatically generating complete graphs on the 4020 microfilm printer
Unpublished memorandum July 1967
- 6 *Specification for standard graphic output subroutines as proposed by the SHARE standard graphic output language committee*
Original prepared by G. B. Knight 1965 Revision prepared by P Pickman 1966
- 7 S C JOHNSON
BSTRNG and A BSTRNG macro processor
Unpublished memoranda May and June 1968
- 8 M V MATHEWS
Automatic Kerning program
Unpublished memorandum January 1968

Interactive languages: design criteria and a proposal

by RICHARD K. MOORE and WALTER MAIN

Tymshare, Inc.
Palo Alto, California

INTRODUCTION

Algebraic languages currently available on time sharing systems can be divided into two categories: batch-oriented languages and conversational languages. The batch languages (ALGOL, FORTRAN, PL/1.; though quite powerful in their ability to express complicated algorithms, are in many ways unsuited to an interactive environment.

Batch languages unsuited as conversational tools

The "Complete Program" requirement

Let us use ALGOL as an example. An ALGOL program is a single compound statement, usually a block. Thus it starts with

begin

and terminates with a matching

end

Within the program must occur: declarations for all mentioned variables, the bodies of all called procedures, the locations of all referenced labels, and matching *end* for each *begin*. If any single component of the program is missing or syntactically incorrect, the program as a whole is invalid.

Preparing and testing a complete, well thought-out program is quite appropriate to a batch environment, where maximum output from each run is an economically sound goal. In a conversational environment however, the preferred practice should be to test and debug each section of a program as soon as possible. Thus, the number of untested statements in a program at any one time remains small, and debugging is straightforward. Such incremental testing of programs is difficult in the batch languages and usually can be accomplished only in spite of the syntax.

The "Deck of Cards" assumption

No meta-statements are provided in the batch languages to allow manipulating or altering a program. (The "compile time statements" of PL/1 form a permanent part of any given program.) The assumption is that a program will be altered by changing physically the position of a card in the deck, or by punching a new card and inserting it into the program. For time sharing systems, text editors must be provided to accomplish this card shuffling. Thus a programmer must learn two unrelated languages to write programs effectively.

The "Perfect Program" assumption

No provision is made in FORTRAN or ALGOL for the problems of debugging. The ON CONDITION statement of PL/1 is really more for detecting bad data than for program debugging. Such features as core dumps (shudder!) and variable tracing are considered part of the specific "implementation" and usually are not even adequate for the requirements of a batch environment.

In an interactive environment there is great potential for computer aided dynamic bug detection. A language which does not tap this potential must be considered lacking.

The "Professional Programmer" assumption

A fairly bright non-programmer would probably need 50-100 hours of study before he could solve even trivial programming problems in the batch languages. Because of the "complete program" requirement, considerable over-learning is required before the student can approach the computer without apprehension. In training a professional programmer, 50 hours is certainly a small investment, but for an engineer who needs a simple calculation performed by a computer, 50 hours is more

time than he can spare. Thus even for trivial programming tasks, an intermediary, the professional programmer, stands between the computer and its primary user. This leads to inefficiency, communication difficulties, extra expense, and prohibitive delays. A superior language can exploit its interactive environment to use the computer as a teaching aid. As the beginner types in his first program, diagnostic feedback will terminate any erroneous learning path, immediate confirmation of each correct statement will reassure him; rapid display of program results will encourage him to pursue the complete solution of his problem.

The introduction of specialized conversational languages

The need for conversational languages is clear. Two of these languages, JOSS and BASIC, overcome most of the objections raised against the batch languages. The most striking feature of JOSS and BASIC is their lack of syntax structure. For example,

1. READ X
2. PRINT SQRT (X), SIN (X)

is already a valid, executable program in either language. In fact, any collection of statements forms a "complete program" provided only that each statement is individually complete. A call to an undefined function is simply a run-time fault, much like an attempt to divide by zero. The programmer this is encouraged to exploit the conversationality of his environment through modular program development.

An integral part of each program statement is its line number. The line number serves two purposes. It determines the order in which statements occur, and it serves as a reference label for GO TO statements or other control statements. A new statement can be inserted simply by typing in the new statement with a line number which falls numerically in the desired position. A section of the program can be referenced as a whole by mentioning the first and last line numbers of the section. For example, in BASIC, the command

```
LIST 1 - 5
```

displays all statements whose line numbers are between 1 and 5 inclusive. Line numbers, together with meta-commands such as LIST, LOAD, MOVE, SAVE, and EDIT, provide facilities within the framework of the language itself to allow a program to be modified, corrected, and displayed.

Apart from "modular program creation," there are two features of the conversational languages which ease the debugging burden.

First, there are *immediate statements*. Whenever a statement is typed in without a line number preceding

it, the statement does not become a permanent part of the program, but is executed immediately. Thus if a program is halted due to a run-time error, values of key variables can be determined through an immediate PRINT statement. Also, immediate statements can be used to preset a program to some singular state, so that particular paths can be tested.

The second debugging feature is *partial execution*. Since every subset (by lines) of a program is also a program, the programmer can set his variables to reasonable values and then execute just that portion of a program which seems to be giving incorrect results. Taken together, these two features allow the programmer and the computer to interact in a dynamic way to pinpoint any faulty program logic rapidly.

Despite their many virtues, the currently existing conversational languages fall short of being ideal algebraic languages for an interactive system.

Conversational languages cramped by batch language styles

An unnecessarily rigid subroutine structure has been inherited from ALGOL and FORTRAN. In JOSS for example, the basic program unit is the PART. "PART 4" refers to all statements from 4.000 through 4.999. "DO PART 4" executes those statements from another part of the program. As an algorithm is being developed, the programmer must take care to group together those statements which eventually will be part of the same subroutine. This vestige of the "complete program" requirement forces the programmer to think ahead to the total structure of his program at a time when he would rather concentrate on translating a few simple ideas into program statements.

The situation in BASIC is only slightly more flexible. In BASIC, a group of lines can be called remotely by a GOSUB statement. For example,

```
GOSUB 100
```

transfers control to line 100 and executes statements in normal order until a RETURN statement is encountered. At that time control returns to the dynamically matching GOSUB. The specification of subroutine boundaries is thus more flexible in BASIC than in JOSS. A certain asymmetry arises in BASIC however; whereas the *beginning* of a subroutine can vary dynamically from one call to another, the *end* (RETURN) is relatively static. For example, the statement

```
GOSUB 100
```

calls a routine one line longer than does the statement

```
GOSUB 101
```

but both must terminate on the same RETURN (barring conditional transfers).

While static subprogram boundaries are best not required in a conversational language, certain other features of the batch languages must be borrowed to allow the clear, concise expression of complicated algorithms

Conversational languages found to be overly restrictive

Long variable names

The first conversational languages were implemented under a somewhat unique set of circumstances. The machines were of small capacity, and the assumption was that only small programs would be attempted in the relatively "inefficient" real time mode. A restriction of variable names to two characters (a letter-digit pair) was seen to yield considerable compiling advantage at minimal sacrifice in the readability of small programs. As the capacity of time sharing systems has increased, and as the technique of incremental compilation has become more widely used, this restriction has become less beneficial and more bothersome.

Symbolic labels

The twofold function of line numbers is to be commended for its economy. Often a programmer will need to refer to some statement as the operand of a control statement, but no suggestive label for that statement will come to mind. On the other hand, program readability is enhanced if those statements in a program which have a definite and easily named function in the mind of the programmer can be distinguished and referenced by alphanumeric labels. Such examples as

```
GO TO DONE;
```

```
GO TO ERROREXIT IF X < 0;
```

substantiate this claim.

Local variables

The concept of identifier scope should not be imposed on the beginning programmer (as it is in FORTRAN and ALGOL). JOSS and BASIC allow procedural programming while maintaining globality of all variables. Although total globality is easily learned and adequate for the most elementary programs, the advantages gained by the ability to isolate symbolically sections of a larger program are clear. In keeping with the spirit of incrementality however, this feature should be specified in such a way as to avoid static structural forms such as *begin-end* pairs.

Full-fledged subroutines and functions

Formal subroutines should not be required in simple programs. The kind of control flow which can be handled by a DO PART or a GOSUB should continue to be handled by some direct, concise control statement. However for more complicated tasks, provisions must be made for mnemonically named subprograms with multi-statement bodies, and with call-by-name and call-by-value parameters. Again, we wish to specify this feature in such a way as to avoid the "complete program" requirement. We do not want static structural forms.

The TCL synthesis

The result of these considerations is TCL (Tymshare Conversational Language). The salient features of TCL include long variable names, symbolic statement labels, totally dynamic statement grouping, dynamically bound variables, full-fledged subprograms with parameters, and the ability to handle recursive procedural algorithms.

Long variable names

Arbitrarily long strings of letters and digits beginning with a letter are allowed as identifiers.

Examples

```
HEIGHT
ALPHA
HP124
```

The names of the variables thus can suggest that which they represent.

Statement labels

Any line of the program may be labelled by an identifier (optionally subscripted by a constant) followed by a colon. The statement then can be referred to either by its line number or by its label.

Examples

```
TOP:
NEXT:
L(1):
L(2):
```

Label subscripts allow convenient computed transfers.

Example

```
GO TO L(I)
GO TO ENTRY(I+2*J+4*K)
```

If all GO TO's and other statement references in a given program use labels rather than line numbers, that program is, but for order, independent of its line numbers. The program can be saved on a file without line numbers and then can be merged easily into other programs in any convenient line number range. Hence, independently tested routines can be combined into one program. The problem of identifier scope will be treated below.

Statement groups

TCL allows any group of statements to be called in the most direct possible way. Thus,

```
DO 1:10, 120, 15
```

would execute line 1 through line 10, line 120, line 15, and then return control to the statement immediately following the DO. Marginal statements (initialization, debugging, or I/O statements perhaps) can be included or excluded as the "routine" is called from various places.

Type declarations

Declaring variable types usually is unnecessary in TCL. Variables dynamically assume the type of any value assigned to them and are considered global by default. When declarations are used, they provide scope information and allow TCL to reduce storage requirements and decrease system overhead. The storage savings are most significant where arrays are concerned.

Scope of identifiers

Declarations, when they do appear, are executed in the normal order of program flow. Declared variables remain defined only until termination of the innermost group in which the declaration is included dynamically. The scope is dynamic rather than static; that is, lines which are called as part of a group perform as if their text appeared at the point of call.

Example

1. DO 3:5
2. DO 6:8
3. REAL A(10)
4. A(I) = 0 FOR I=1 TO 10
5. WRITE:A
6. INTEGER A
7. A = 7
8. DO 5

When line 5 is called from line 1, the array A is printed out; but when line 5 is called by line 8 (as part of the range of line 2) the integer scalar A is printed.

A local variable of unspecified type can be declared by the statement

```
DYNAMIC X(10), Y
```

X is then a local array each element of which can hold a value of any type, and Y is a local simple variable.

Dynamic binding of variables accommodates the most sophisticated recursive algorithm and the simplest program with equal ease and economy. Thus,

- a. The novice programmer needs neither scope nor type declarations. He simply uses unique names for each of his variables. When inside a called group, all of his variables are available to him. If he incorporates a library routine into his program, it is required only that the library program explicitly declare all of its local variables. The user's variables will reappear (in case of any conflict) after exit from the library routine.
- b. The sophisticated user will find dynamic binding the functional equivalent of the conventional block structured static binding. The dynamic approach actually will be more flexible in some cases. Most important, the mechanism of dynamic binding is more transparent to the programmer than the subtle situations which can arise in a static language like ALGOL; for example, uplevel addressing and generalized call-by-name.

Subroutines and functions

A *subroutine* is declared by specifying the statement group which comprises the *subroutine body*:

```
DEFINE S(X, Y) AS 10, A:B
```

'S' now names a subroutine of two formal parameters ('X' and 'Y'). When 'S' is called by the statement

```
DO S(T, 14.5)
```

any currently defined variable named 'X' or 'Y' will become temporarily unavailable; 'X' will be identified with the object 'T', and 'Y' will become a variable with initial value 14.5. Statements 10, and 'A' through 'B' would then be executed. Upon termination of 'B', 'X' and 'Y' would recover their prior definitions and values.

Such a subroutine may become part of the definition of a later subroutine:

```
DEFINE P(U, V, W) AS L1:L2, S(U, V+1), 15
```

In general, a subroutine call may appear as part of a statement group:

```
DO TOP:BOTTOM, P(1, 2, 5), S(0, 1)
```

A *function* may be declared by specifying the desired evaluation expression:

```
DEFINE LENGTH (X, Y) = SQRT (X↑2 +
    Y↑2)
```

Or the evaluation can be deferred until completion of a specified function body:

```
DEFINE F(A, B) = (B-A)/(T + U) AFTER
    F1:F2
```

The function body has the same allowed generality of structure as any other statement group:

```
DEFINE PROCESS(T1, T2) = T2-T1 AFTER
    ORIGIN(T1), TOP: BOT, TERM(T2)
```

Thus, TCL subprograms impose a procedural structure upon a program quite like that permitted in a language like ALGOL, but, unlike ALGOL, the structure comes into existence only at the time the subprogram is called. Not only does TCL thereby permit the modular program development so convenient in conversational programming, but a structural flexibility is introduced that far exceeds such ad hoc inventions as the PL/1 "multiple entry" feature. As a simple example, consider the following TCL subroutine declaration:

```
DEFINES (I, J, X, Y) AS L(I): L(J)
```

The following sample TCL session is intended to impart the flavor of TCL as a dynamic programming tool. A detailed specification of TCL syntax can be found in the Appendix. NOTE: Underscored copy in the following example indicates what is typed by the user. All other text is typed by the computer.

A sample session in the TCL system

```
>1. S = 0
>2. S = S+X(I) FOR I = 1 TO N
```

A test array is created by direct statements:

```
>REAL X(10)
>X(I) = I FOR I = 1 TO 10
```

Summation statements can be tested immediately:

```
>DO 1:2 FOR N = 10
>WRITE:S
55
```

The global function SUM is easily defined and tested:

```
>DEFINE SUM(X, N) = S AFTER 1:2
>WRITE:SUM(X, 9)
45
```

SUM is used in an "adding machine" program:

```
>3. READ:N
>4. REAL A(N)
>5. READ: A(I) FOR I = 1 TO N
>6. WRITE: SUM(A, N)
```

The adding machine is checked out:

```
>DO 3:6
5, 100, 120, 105, 130, 110
565
```

SUM is used to create an averaging function MEAN:

```
>DEFINE MEAN (X, N) = SUM(X, N)/N
>WRITE: MEAN(X, 10)
5.5
```

The multi-line standard deviation function STD is created. The function (not array) Y can be passed to SUM.

```
>7. M = MEAN (D, N)
>8. DEFINE Y (I) = (D(I)-M) ↑2
>DEFINE STD(D, N) = SQRT(SUM(Y, N)/(N-
    1)) AFTER 7:8
>WRITE STD(X, 3)
1.581138
```

Finally a complete conversational program makes use of the tested functions.

```
>9. WRITE: 'NUMBER OF VALUES ='
>10. WRITE: 'ENTER VALUES:'
>11. WRITE: 'SUM =', SUM(A, N), 'MEAN
    =', MEAN (A, N),
    'STANDARD DEVIATION = ', STD
    (A, N)
>DEFINE ANALYZE AS 9, 3:4, 10, 5, 11
>DO ANALYZE
NUMBER OF VALUES = 5
ENTER VALUES: 10, 12, 14, 21, 8
SUM = 65 MEAN = 13 STANDARD DEVI-
ATION = 5
```

Now that the program is finished, X is unnecessary.

```
>DELETE X
```

The current program, including direct declarations is printed on the terminal:

```
>LIST
```

DEFINITIONS:

```
SUM(X, N) = S AFTER 1:2
MEAN(X, N) = SUM(X, N)/N
STD(D, N) = SQRT(SUM(Y. N)/(N-1))
AFTER 7:8
ANALYZE AS 9, 3:4, 10, 5, 11
```

STEPS:

1. S = 0
2. S = S + X(I) FOR I = 1 TO N
3. READ: N
4. READ A(N)
5. READ: A(I) FOR I = 1 TO N
6. WRITE: SUM(A, N)
7. M = MEAN(D, N)
8. DEFINE Y(I) = (D(I) - M) \uparrow 2
9. WRITE: 'NUMBER OF VALUES ='
10. WRITE: 'ENTER VALUES: ='
11. WRITE: 'SUM =', SUM(A, N), 'MEAN =',
MEAN(A, N), 'STANDARD DEVIATION =',
STD(A, N)

The program is saved for future use:

```
> LIST ON "ANALYSIS"
NEW FILE
>
```

CONCLUSION

The sample session shows the ease with which a simple problem can be solved in TCL. Not only can the program be built up and tested incrementally, but the final program is concise and readable.

TCL is thus a self-documenting, conversational language, free of artificial structural specifications. Furthermore, the language naturally accommodates the statement and solution of increasingly complex tasks.

APPENDIX - TCL LANGUAGE SUMMARY

1. Language Elements

• *identifier or ident*

Definition:

Alphanumeric string beginning with letter.

Examples:

```
A
B12S
ALPHA
```

• *expression*

Definition:

Usual definition.

Examples:

```
A + B
(S AND T) OR L7
```

• *line number*

Definition:

Decimal constant from .001 to 999.999

Examples:

```
37.5
12
190.002
```

• *label reference*

Definition:

ident or ident (expression)

Examples:

```
A
LOOP(3)
START(I+2)
```

• *line ref*

Definition:

line number or label reference

Examples:

```
12.21
L(A(J))
```

• *range*

Definition:

line ref or line ref:line ref

Examples:

```
1:10
7
A(1):A(I)
1:NLUP
```

• *subroutine ref*

Definition:

subroutine name (expression list)

Examples:

```
SUBL(7, X+Y)
```

• *object*

Definition:

subroutine ref or range

Examples:

```
S(X, Y)
A:10.2
```

• *group*

Definition:

object list

Examples:

```
1:10, 50, S(X, Y)
1,5
```

• *condition*

Definition:

Boolean expression

Examples:

```
A AND X < Y
```

• *iteration-part*

Definition:

expression [BY expression] [TO expression]

Examples:

```
1 BY 2 TO N
```


- *file name*
Definition:
 "string without quote"
Examples:
 "PROG"
 "A"
 ",/."
- *mode*
Definition:
 REAL or INTEGER or BOOLEAN or
 DOUBLE or COMPLEX or STRING or
 DYNAMIC
Examples:
 REAL
 DYNAMIC

2. Primary Statements

- GO TO *line ref*
Definition:
 Transfer control to referenced line.
Examples:
 GO TO A(I)
 GO TO 3.4
- DO *group*
Definition:
 Execute referenced lines and subroutines.
Examples:
 DO 3.4
 DO 6.1, S(X, Y), A:B
- *variable = expression*
Definition:
 assignment
Examples:
 X = (A+B)/3.4
- OPEN *file name* FOR [BINARY] $\left\{ \begin{array}{l} \text{INPUT} \\ \text{OUTPUT} \end{array} \right\}$
 AS FILE *integer expression*
Definition:
 Prepare file for READ or WRITE.
Examples:
 OPEN "A1" FOR INPUT AS FILE 4
 $\left\{ \begin{array}{l} \text{READ} \\ \text{WRITE} \end{array} \right\} \left\{ \begin{array}{l} \text{FILE } \textit{integer expression} \\ \text{STRING } \textit{string expression} \\ \text{IN FORM } \textit{string expression} \\ \text{IN IMAGE } \textit{string expression} \\ \text{DATA} \end{array} \right\} \textit{expression list}$
Definition:
 Perform input/output operation:
Examples:
Free Form Input From Terminal:
 READ:X, Y, Z

Free Form Output to Disk:

WRITE FILE 4:A, B, C

Formatted Input From Terminal:

READ IN FORM S1:A(I)

- CLOSE FILE *integer expression*
Definition:
 Terminate input/output operation on a file.
Examples:
 CLOSE FILE 7
- MAP *array name (subscript bounds)*
Definition:
 Allocate or reallocate storage for an array.
Examples:
 MAP A(0:N)
 MAP X(M, N)

3. Meta-Statements

- LIST [*range list*] [ON *file name*]
Definition:
 List portion of symbolic program.
Examples:
Print Entire Program On Terminal:
 LIST
Write Selected Lines On Disk:
 LIST L:10, 20 ON "SAVE"
- DELETE *range list*
Definition:
 Erase portion of program.
Examples:
 DELETE A:B, 21.2
- LOAD *file name*
Definition:
 Merge contents of file into symbolic program.
Examples:
 LOAD "PROG1"
- RUN
Definition:
 Execute program.
Examples:
 RUN
- SET *range list*
Definition:
 Set breakpoints in program.
Examples:
 SET 1, 2, 5:10, LOOP
- RESET *range list*
Definition:
 Erase selected breakpoints.
Examples:
 RESET 6:10

- EDIT *range list*

Definition:

Allow modification of selected program lines under control of a text editor.

Examples:

EDIT A:B

- RENUMBER [*range*] [AS *range*] [BY *line number*]

Definition:

Assign new line number to a portion of program.

Examples:

RENUMBER A:B AS 20:30 BY 2
RENUMBER 1:100 BY 1

4. Modifiers

- IF *condition*

Definition:

Execute modified statement if condition is true.

Examples:

X = Y IF X > Y

- UNLESS *condition*

Definition:

Execute modified statement if condition is false.

Examples:

Y = SQRT(X) UNLESS X < 0

- WHILE *condition*

Definition:

Execute modified statement repeatedly as long as condition is true.

Examples:

DO SEARCH (A, B) WHILE (B-A) > EPS

- UNTIL *condition*

Definition:

Execute modified statement repeatedly as long as condition is false.

Examples:

DO 1:10 UNTIL C1 OR C2

- FOR *variable iteration-part list*

Definition:

Repeat modified statement assigning each value in *iteration-part list* to indicated variable.

Examples:

DO TOP:BOTTOM FOR YEAR = 1960 TO 1970
("BY 1" ASSUMED)

A(I) = B(I)/C(I) FOR I TO N ("= 1 BY 1" ASSUMED)

WRITE: SQRT(X) FOR X = 0 BY .01 TO 2.5

5. Declarations

- *mode variable list*

Definition:

Type declaration.

Examples:

REAL A, B(10), C() ("C" later will be the object of "MAP")
DYNAMIC I, J, K, X(10)

- DEFINE *ident* [(*parameter list*)] = *expression* [, [AFTER *group*]]

Definition:

Function declaration.

Examples:

DEFINE DISTANCE(X,Y) = SQRT
(X↑2 + Y↑2)
DEFINE F(X, Y, Z) = (A+B)/2 AFTER
F1:F2

- DEFINE *ident* [(*parameter list*)] AS *statement group*

Definition:

Subroutine declaration.

Examples:

DEFINE PSUM(X,Y) AS WRITE:X+Y
DEFINE PROCESS (TIME1, TIME2) AS
INIT
(TIME1),(UPDATE(TIME) FOR TIME
= TIME1 TO TIME 2), SUMMARY
(TIME2)
DEFINE PART 1 AS 1:1.999

- *ident* [(*integer*)]:

Definition:

label declaration

Examples:

A:
LOOP:
T13:
T(5):

BIBLIOGRAPHY

G E BRYAN J W SMITH

Joss language

MEMORANDUM RM-5377-PR August 1967 prepared for USAF Project RAND The RAND Corp Santa Monica Calif

Super basic reference manual

October 1968 Tymshare Inc Palo Alto Calif

META PI—An on-line interactive compiler-compiler

by JOHN T. O'NEIL, Jr.

RCA Laboratories
Princeton, New Jersey

It is difficult to specifically date the origin of the research efforts within the programming discipline that are directed at describing and implementing a language which would produce compilers.

The motivation for these efforts stems from meta languages such as Backus Normal Form (BNF)¹ which attempt to describe in a mathematical notation the syntax (structure) of a programming language. The thinking is that if a given language (FORTRAN, ALGOL, etc.) could be described in rather precise form, then it should be possible to construct a translator that would accept statements, say, in BNF and output the appropriate compiler. This processor is shown schematically in Figure 1.

The actual construction of the compiler-compiler has proved to be an elusive goal; the efficient implementation of the theoretically possible turned out to be far more difficult than originally anticipated.

In early 1966 work began at the RCA Laboratories, Princeton, on what has since evolved into RCA BTSS II (Basic Time Sharing System, Version II). During the design discussions for this system it was decided that the interactive language would be based on FORTRAN IV. It was further decided to implement the language, so far as possible, using a compiler-compiler. The final compiler was named FORTRAN PI and its compiler-compiler parent, META PI. It is the opinion of the author based on implementation experience and user acceptance that the viability of the compiler-compiler has been amply demonstrated by the research effort which produced META PI and FORTRAN PI.

Before discussing META PI it will be necessary to

discuss BTSS II since it is the operational environment within which META PI functions.

RCA BTSS II provides the on-line user the ability to create, modify, execute and correct programs on an interactive basis. The user accesses the system services through three main software components:

A Command Language.

A Text Editor.

The FORTRAN PI compiler and META PI compiler-compiler.

FORTRAN PI and META PI were designed as far as possible to be independent of a given control system and I/O package. Both FORTRAN PI and META PI interface with the system via an interactive executive.

RCA BTSS II is implemented on an RCA SPECTRA 70/45 computer system with 131K of memory. The SPECTRA 70/45 is a third generation computer system with an instruction set which is compatible with System 360. It does not have hardware features (paging, read memory protect, etc.) specifically designed for time sharing. (The RCA SPECTRA 70/46 does have these features, and a version of the PI compiler is operating on it.)

FORTRAN PI was the first language implemented with META PI. A discussion of its design and the structure of the object code produced by it will be helpful in providing the reader insight into the design and function of META PI.

The reader is cautioned to keep in mind the various possible levels of translator activity, that is, the initial creation of FORTRAN PI via META PI, the on-line creation of the user's program via FORTRAN PI, and the on-line creation of the user's compiler (or compiler-compiler).

During the preliminary design phase for BTSS the fundamental decision was made to use a FORTRAN like language as the problem solving language of the system. Three considerations provided the framework for all subsequent design decisions.

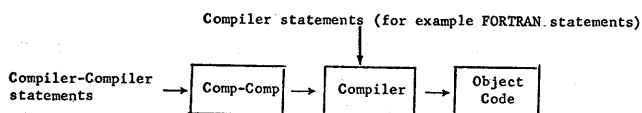


FIGURE 1

First, to gain insight into the viability of compiler-compiler approaches, the implementation of FORTRAN PI would proceed only after the structure of the META PI compiler-compiler was described in detail. As much as possible of the FORTRAN PI compiler would be implemented via META PI.

Second, trade offs would be made in the total META PI approach if, in implementing FORTRAN PI, efficiency of the production compiler would be seriously impaired by this approach.

Third, FORTRAN IV standards would be adhered to wherever possible, but since the language was to be utilized in a time sharing environment, departures from FORTRAN IV standards would be effected whenever the convenience of the terminal user would suffer otherwise.

In retrospect, considering the rather ambitious design constraints, FORTRAN PI was able to meet the bulk of its design objectives. Over 80% of the object code of FORTRAN PI is generated from META PI. The compiler itself is remarkably similar to FORTRAN IV when one considers the conflicts of user utility that arise when one attempts to reconcile a language designed for the batch user with the requirements of interactive time sharing.

For example, some of the present FORTRAN PI alterations to FORTRAN IV are:

1. Free field input format to both compiler and I/O Formatter.
2. Format statements are optional.
3. Recursive functions and subroutines.
4. Arbitrary subscripts.
5. Negative increments in DO loops, etc.
6. Symbolic variable tracing, flow tracing, and other debugging aids.

Further alterations are, of course, easily implemented via META PI.

The FORTRAN PI compiler has the following characteristics:

1. Statements are accepted and compiled a line at a time on an interactive basis.
2. The object code generated is read only and is capable of immediate execution. (The FORTRAN PI compiler is a one pass compiler.)

The above characteristics are desired for several reasons. First, it was desired that programs be compiled rather than interpreted for greater run time efficiency.

Second, the read only feature of the object code permits the executive to omit writing the code back to disc when each run time execution slot terminates.

Third, the immediate compilation allows several

important additional advantages to accrue to the interactive user. Among these are:

- A. The compiler can be used as a desk calculator.
- B. Complete symbolic debugging aids (the principal advantage of interpreters) are still available due to the easy access of the compiler and symbol table at run time.
- C. The user can symbolically alter variables in a running program without re-compiling or re-starting.
- D. The user can cause each program statement to be executed (incremental execution) while it is being compiled a line at a time (incremental compilation).

The compiler itself is composed of two sections; a set of subroutines which are hand coded and the code generated by META PI. The subroutines fall into two classes.

- a. Those which are not sensitive to the language being compiled. These routines are used by both FORTRAN PI and META PI and as such can be used for generating new compilers. An example of this class of subroutines is INUM; this subroutine tests the input stream for a digit string of arbitrary length.
- b. A set of subroutines whose generality is a function either of the hardware on which the compiler is being implemented or the particular source language itself. For example, the routine EFFI detects the occurrence of certain instruction pairs and replaces this pair with a single instruction. This replacement is obviously dependent on a specific hardware instruction set. Another routine FLB is used to detect valid FORTRAN PI FOR-MAT statements. Such a routine is unique to FORTRAN and is not useful in the implementation of other languages. These non-transferable routines comprise less than 5% of the total FORTRAN PI object code.

The second section of FORTRAN PI is composed entirely of code generated by META PI. This coding performs a left to right scan of the source text, testing for syntactic units exactly as specified by the input to META PI. The structure of the input to META PI will be taken up shortly.

FORTRAN PI accepts source statements from terminal users and generates the machine code necessary to carry out the intent of the statement. The compiler uses five regions in creating the users object program. These regions are created in $\frac{1}{2}$ page blocks. A $\frac{1}{2}$ page is 2048 memory locations (bytes); this is the minimum size block that can be memory protected on

the Spectra 70/45. The number of $\frac{1}{2}$ pages allocated to each region is user specifiable at the time his program is created. The five regions are allocated as follows:

Regions 1 and 2

Contains the compiler's working storage, a specially constructed statement table and the source program label table.

The statement table is used for symbolic debugging; it enables the user to trace his program on selected criterion. For example, the program can be halted at any statement number, or the user can cause a symbolic printout when the value of specified variables change.

The label table contains information on every program variable and statement which contains a statement number.

Region 3

This area contains the user's compiled code. The generated code is "read only" and self-relocating. As a result the code $\frac{1}{2}$ pages need never be written when the program is being staged out at the end of an execution time slot (the maximum time slot is $\frac{1}{2}$ second).

Since the code generated is self-relocating it can be used as shared code on virtual memory hardware even though the current implementation is on a processor without virtual memory capabilities.

Region 4

This region contains constants that appear in the user's source statements and all variables that have been declared in COMMON statements.

Region 5

This area contains the value of variables not in COMMON, DO loop indices and the recursive function stack area.

FORTRAN PI functions and subroutines are recursive. The dynamic memory requirement needed to efficiently support a recursive process is obtained from Region 5. Thus the actual storage used in Region 5 expands and contracts dynamically during execution of the user's object program.

Since FORTRAN PI is implemented in the main by META PI its structure, which is designed for efficiency in the time sharing environment, is in fact determined by META PI. The implication is that any other language implemented using META PI would also have this region oriented structure; this without any special effort on the part of the language implementer using META PI.

The full implication of the nature of the compilers generated by META PI will be amplified when the implementation of Dartmouth BASIC using META PI is discussed in a later document.

META PI is a *problem* oriented language, that is, it is designed for use by individuals implementing entire compilers, syntax checkers, or for extending the capability of current compilers to satisfy special language requirements.

It has long been proposed that the structure of languages must be placed within the domain of the user; the logic is that only the user can be truly sensitive to his own specific needs. It is the purpose of problem oriented languages to achieve just this end, that is, they provide the user with a language that enables him to solve problems with special structural characteristics that would be either extremely difficult or, from the economic point of view, impossible to solve with procedure or assembly level language.

One of META PI's problem oriented objectives aims at providing the user the ability to create languages suited to his own needs without requiring that the user be familiar with specific computer hardware or the basic internal structure of compilers. This goal has yet to be achieved in its entirety, but META PI has demonstrated that the concept is feasible and that it's only a matter of time before the user will be provided with the capability for developing his own languages just as he is now able to create his own programs; the only remaining problems to be solved relate to the extent to which symbolics should be used within the compiler-compiler languages themselves.

In order to define a problem oriented language it is first necessary to examine the characteristics of the problem that the language is to solve

The language of a compiler-compiler (META PI) must be designed to solve the problem of compiler generation.

Compilers perform two basic functions:

1. They scan input statements in order to determine their validity within the definition of the language. The valid statements within a language is established by the *syntax* of that language. For example the Dartmouth BASIC statement

```
10 LET X = X + 1
```

is valid
while

```
10 LET X = JOHN + 1
```

is not, since in Case 2 the variable JOHN is not permitted in the language and hence is *syntactically* incorrect.

2. The second requirement of a compiler is the generation of the necessary computer instructions for effecting the execution of syntactically correct statements. This phase of the compiler implies that a meaning (semantics) is to be associated to a given statement. The meaning supplied takes the form of generated object code.

A compiler-compiler then must contain structural elements necessary to provide, in the compiler it produces, the ability both to scan for correct statements (syntactical structures) and also to produce object code. The user of such a language is freed of all the details that are involved in the actual generation of machine code required to implement the compiler itself.

META PI uses as its basic language structure the META series of compiler-compilers described by D. V. Schorre² and his associates at the UCLA computing facility. Its implementation, however, unlike the META series of compiler-compiler of the UCLA group is intended primarily for interactive software system. It has been used to generate two interactive compilers that are used on a production basis.

The basic parsing algorithm of the META type compiler is top-down left to-right, and deterministic. Briefly, "top-down" means the compiler first decides which rule should be satisfied next and then checks the input (or calls new rules) according to the alternatives of the rule. A "bottom-up" parser would, on the other hand, first check the nature of the input and then determine which rules could be used to describe it. A top-down, deterministic algorithm was selected for three principle reasons.

1. Coding can be generated immediately for the META statements as they are read in. This meshes with the goal of having incremental compilation.
2. Errors are easily pinpointed in deterministic parser. Backup is provided only when explicitly specified in the META PI language.
3. Deterministic parsers are faster than non-deterministic parsers.

As has been stated, the first requirement of a compiler-compiler language is to provide the language it creates a syntax checking capability. Fortunately, the syntactical description of programming languages has been provided a powerful symbolism in the Backus Normal Form (BNF).

BNF achieved its fame from its use in ALGOL '60 but is suited for describing a broad class of languages. It provides an excellent vehicle for the statement structure of a compiler-compiler. In order to enable the generated compiler to syntactically test the input statement, a BNF description is converted by META PI to generated code that will perform syntactic tests

on the input statement. Though it is well suited to the syntactic phase of a compiler's work BNF was not designed with the intent of attaching semantic meaning to the statements involved.

It is in the area of semantics that the major effort in design has occurred in the development and definition of META PI.

META PI is computer program written for the RCA Spectra 70 that accepts the description of a language in extended Backus Normal Form. Both the syntactic and semantic functions of the compiler to be generated are contained within a single META PI statement. The output of the interactive version of META PI is (read only, sharable) Spectra 70 machine code which is the compiler for the language being described. This output code is unique to a given on-line user and does not interfere in any way with other on-line users who are sharing META PI interactively. The user of META PI can in fact have any number of different languages in various stages of development; the system does not distinguish between programs written in FORTRAN PI and those written in META PI; FORTRAN PI, META PI and the user's compiler form an integrated language system in RCA BTSS II.

The object code produced by META PI consists primarily of a set of subroutine calls which perform a recursive left to right scan of the source statements of the particular compiler language it describes.

META PI statements are designed to resemble Backus Normal Form. It was important, however, to extend BNF in order to include semantic operations (code generation) within the syntax structure describing the language and to simplify the description of the language. Four extensions were involved:

1. The inclusion of factoring and the addition of an iterative operator. For example the BNF statement

$$A:: = B/AC/AD$$

becomes

$$A: = B\$ (C/D)$$

These changes were necessary for two reasons. First, the use of the \$ sign enables the compiler to identify an iterative operation immediately on the appearance of the dollar sign (\$). This greatly simplifies the compilation process. Second, since META PI is an interactive language the \$ notation reduces input requirements thus increasing terminal efficiency. Furthermore, from the purely descriptive point of view, it simplifies the identification of proper strings defined by the statement, since the \$ can be interpreted to mean "followed by

an arbitrary sequence of." Hence the sample META PI statement above is read as:

"An A is a B followed by an arbitrary sequence of C's or D's."

From just the visual examination of the string

BCCDDCCDDDD

it is difficult to determine using the BNF descriptor whether or not the string is valid. With the extended BNF descriptor of META PI however it is immediately obvious that the above is in fact a valid string, that is a B followed by an arbitrary sequence of C's or D's.

2. The semantics are included within the syntax of a statement. This allows for object code to be generated as the scan of the source statement proceeds; in the vast majority of statements scanned, the complete generation of code and end of scan will occur simultaneously.
3. The ability to backup the code generation to some previous scan point is provided through special commands that are part of the META PI statement structure. This feature allows for efficient identification of those statement strings belonging to a language but not immediately identifiable on a left to right scan basis as a particular statement type. Consider for example the FORTRAN PI statement

DO11 = 1.5

This statement is a valid assignment statement which assigns the value 1.5 to the variable DO11. If the syntax analysis, however, begins analyzing the statement as a DO statement it will not be rejected as such until the analysis of the statement is nearly completed. The backup facility of META PI provides an efficient means for re-evaluating the input string as a different statement type. It must be noted that the backup facility is provided for the scan of the source statement to the compiler being generated. It is *never* necessary to backup during the scan of a META PI statement since META PI is a deterministic language.

4. The compiler writer is provided with the capability of generating compile time error comments via a special error command which is also an integral part of the META PI statement structure. (This feature is not available in the interactive version of META PI described here.)

Before proceeding with a discussion of how META PI statements are written a discussion of META PI vs.

BNF syntax is in order. The following conventions will hold:

<i>META PI</i>	<i>BNF</i>
: =	:: =
/	
ABC	<ABC>
: ABC :	ABC

In addition:

1. A ; will terminate a META PI statement (unnecessary in the on-line version).
2. () [parentheses] will be used to simplify BNF and will indicate factoring.
3. A \$ replaces BNF finite state recursion.

To solidify META PI syntactical symbolism a few Dartmouth BASIC statements are shown below in BNF and META PI.

BASIC READ statement

BNF

<READ statement> ::= READ <read list>

META PI

READST := :READ : READLST

BASIC read list

BNF

<read list> ::= <variable> | <read list> , <variable>

META PI

READLST := VAR\$ (:;:VAR)

BASIC FOR statement

BNF

<FOR statement> ::= FOR <simple variable> = <expression> TO

<expression> <OPTEXP>

<OPTEXP> ::= STEP <Expression> | <EMPTY>

META PI

FORST := : FOR : SIMVAR := : EXP : TO : EXP (:STEP:EXP/.EMPTY)

These examples are included to illustrate the similarities of BNF and META PI syntax. For the purpose of

these illustrations an effort has been made to name syntactic components to convey the same meaning they had in the BNF statement. For example the BNF `< expression >` became EXP.

It should again be emphasized that BNF does not include any facilities for including semantics operations within syntax operations hence none of META PI's semantic operations were shown.

META PI statements contain 3 types of elements:

1. Syntactic elements; these elements are compiled into code in the user's compiler that will test for syntactic elements in the source input to the user's compiler. These elements, then, are used to generate the "sieve" statement identifier or syntax checker of the user's compiler.
2. Semantic elements; the elements are compiled into code in the user's compiler that will effect the generation of object code.
3. META syntactic elements; these elements are compiled into code in the user's compiler that will enable it to efficiently resolve possible conflicts (ambiguities) in the newly defined input source statement via a backup facility. The user constructs META PI input statements by combining these three elements so as to produce his own compiler.

The general form for a META PI statement is:

LABEL: = expression

The left hand side is a unique identifier which serves as a reference to the expression on the right hand side (a META PI identifier is defined as a letter (A-Z) followed by an arbitrary sequence of letters or digits). For example the META PI statement which defines a digit would appear as:

DIGIT: = :0:/:1:/:2:/:3:/:4:/:5:/:6:/:7:/:8:/:9:

The name DIGIT can then be used on the right hand side of an expression to effect the test for a digit.

The character pair `: =` serves as a delimiter and distinguishes META PI statements from FORTRAN PI statements. The reader is reminded that META PI and FORTRAN PI are one integrated language package.

The expression is compiled into code in the user's compiler which is recursive, that is, the expression can contain a reference to itself either directly or indirectly.

When META PI generates the code for the expression within the user's compiler it will be generated such that it can have one of three results after being called.

1. True. This results if the input scanned as a result of being called satisfies the expression. The called routine will return with a truth indicator set, the input pointer will be moved past the data correctly scanned.
2. False. The input does not satisfy the expression, in this case the input pointer will be unaltered. The truth will be set indicating false.
3. Error. The expression prefix is correctly identified but the suffix is not. For example the statement

GO TO 20.3

is an invalid GO TO statement. The prefix GO TO is (possibly) correct but the suffix 20.3 is not. When this occurs an error routine is called, the input pointer is partially updated, the error routine will then insert a ? (question mark) after the last character successfully scanned.

These three conditions describe the behavior of the code that is generated in the user's compiler by a META PI expression. Some of the elements that comprise these expressions will now be discussed in detail.

Syntactic elements

`:XXX....X:` The X's represent any character string. This syntactic element will create code in the user's compiler to test the current input for the string within the colons. In the DIGIT statement, shown previously, code would be generated that would test for a 0 or a 1 or 2 etc.

ABC This results in the generation of code in the user's compiler which will result in a call to the routine named (ABC in this case). This routine will presumably be written by the user with META PI. DIGIT defined above is such a routine; it could be used, for example to identify a number.

INUM: = DIGIT\$DIGIT

The name could also designate one of the currently existing FORTRAN PI routines. This syntactic element is one of two possible methods available for linking to subroutines within META PI. The

second method involves preceding the routine name with a period. When the period notation is used META PI will assume that the routine called is not recursive and that a truth indicator is to be returned. When a routine is called without a period recursion is then possible by the routine called; a truth value will be returned by the called routines in either case.

.ID This is the test for an identifier. Code is generated to link to the ID routine. Note the use of the period. The implication is that the ID routine does not subsequently link to itself.

.EMPTY This is a special syntactic test which forces the true setting of the truth indicator.

.INT This is a test for a FORTRAN integer.

.NUM This is a test for a number which could (approximately) be defined by the following META PI statement:

```
NUM: = $DIGIT(././EMPTY)
$DIGIT(:E:(+ :/: - :./EMPTY)
DIGIT(DIGIT/./EMPTY)/
./EMPTY)
```

The code generated for this statement will identify numbers such as

```
1.23E-01
.001371E-15
1.361, 0123, 1E1
```

the definition could be read as: "A NUM is equivalent to zero or more digits followed by an optional period followed by zero or more digits followed by the optional sequence; E followed by an optional plus or minus followed by a digit followed by an optional digit."

The NUM definition is relatively

simple yet it illustrates factoring, iteration (\$), tests for syntactic elements and the use of .EMPTY; a clear understanding of these elements will benefit the reader when other examples are given later in this document.

LKUP This results in code being generated in the user's compiler that will link to the LKUP routine; this routine scans the label table for the last input detected. Label table entries are statement numbers or variable names. Each entry also contains appropriate control information such as type, memory address and program level. The routine will return one of three possible results.

1. The input was in the label table and assigned memory location is defined.
2. The input was not found in the label table.
3. The label was found but its memory location is yet to be defined. This type of entry is caused by forward references. For example a GOTO statement that specifies a statement number that has not yet been entered.

.TYPE(:NNYY:) This routine looks up the input passed to it in the label table and tests if the type byte is in the class allowed by the argument NNYY. One function of this routine is to check for mixed mode errors.

.XXXX Here the X's represent an arbitrary identifier. The use of this notation will cause META PI to generate linkage to the subroutine named by the symbol. The execution of the subroutine is assumed to effect a test on the input string. The results of this test will set the truth indicator which is returned to the calling routine. This notation is, in fact, the vehicle used by META PI in generating linkage to those syntactic routines previously discussed (.ID, .LKUP, etc.). In

addition to the symbols already defined, the user of META PI can link directly to those routines (written in META PI) that are used in creating the FORTRAN PI compiler; there are over 100 such routines most of which perform functions common to algebraic compilers. The META PI implementation of FORTRAN PI appears in Appendix 1.

Semantics—Code generation in META PI

The syntax operations permit the user who is implementing his own compiler to perform the statement identification function of the compiler being generated. The code generation that will effect the intent of a given source statement is handled by the semantic functions, these functions are imbedded in the META PI statement structure.

The semantic functions are composed of two sub elements:

1. Semantic commands.
2. Semantic operations.

Semantic operations are always contained within semantic commands. The general form is

semantic-command (semantic-Operations).

Semantic commands

Every semantic command has a direct effect on code generated by the compiler. When META PI encounters a semantic command in the input statement it will generate in the user's compiler the object code necessary to generate an element of an object program. There are five basic semantics commands.

.OUT(...) This command causes the current contents of the output area (a temporary area where code is being created by the user's compiler) to be converted to internal form and placed in the user's code area. The output area is a staging area for intermediate output that is in a semi-symbolic form. The code area contains the precise object code that will be executed by the computer. The output itself (the strings that are entered into the output area) is produced by the semantic operations that are specified within the

parentheses (which are shown above as (...)). Three alternate actions can occur depending on the structure of the semantic operations contained within the .OUT(...) command.

1. If the first character is not a letter or a digit, then all subsequent characters are copied directly into the code area until a final colon (:) pair is detected.
2. If the fourth character is a period or a space it is assumed that the output is an instruction using an index register and with a symbolic address following the period or space located in position 4. The symbolic address will be looked up in the label table and from information contained there a real machine address will be generated.
3. If the above two cases fail, the character string is assumed to be machine code and it is converted directly into the code area.

.LABEL(...)

This takes the current contents of the output area and places it into the label table. An error results if the label is already defined. The current value of the code area location counter will be associated with the label.

.IGN(...)

This command will ignore (delete) the contents of the output area. This is useful since several semantic operations produce side effects, such as releasing registers, in addition to generating code.

.NOP(...)

This command is used to produce the effect of the semantic operations without doing anything else. The results of the semantic operations will be left in the output area.

.DO(...)

This is a specialized command whose effect is to cause META PI to execute immediately the instructions contained within the parentheses.

Semantic operations

The semantic operations are used to generate code in the output area. The code generated in the output area by these operations is in a symbolic form and not immediately executable; additionally these operations are generally constrained not to alter the input pointer or the truth indicator. A pointer is maintained to remember the next available location in the output area.

This pointer is updated after each semantic operation. These operations are listed below.

- :CCC...C: Suffix the string between the colons to the output area. Note that no ambiguity exists with syntactic elements contained within colons since this notation has unique meaning depending on whether it has occurred inside or outside of a semantic command.

- * Suffix the current input to the contents of the output area. This is generally used in conjunction with a successful .ID test. To emphasize the different roles being played by META PI, the user's compiler source statement which is input to the user's compiler, and the resulting object code, this simple operation will be explained further. When the * is found in a META PI string, code will be generated in the user's compiler to effect the placement of the last input into the output area. This code is part of the user's compiler. When a source statement is supplied to this compiler the user's compiler will effect symbolic code generation in the output area. This output area will then be converted into executable machine code.

- S Save a copy of the current contents of the output area in a pushdown list and push the list.

- R Restore (suffix to the output area) the top of the pushdown list and pop the list.

- I Ignore (pop) the top element in the pushdown list.

- X Swap the top two elements in the pushdown list.

*1 Generate a globally unique 4 byte character string beginning with the character #. This string will be locally constant and serves as a convenient way to label and reference locations in the generated code.

There are a set of semantics routines which facilitate the use of the general purpose and floating point registers of the Spectra 70 processor in the output code. A type of pushdown list for both of these register types is maintained at run time. There are 6 general purpose and 4 floating registers available to these semantic operations. If more registers are needed, coding will automatically be generated to implement saving and restoring of registers. This save and restore operation is a side effect of the following semantic routines.

- OF Output the current general purpose register.

- O Output the current floating point register.

- + Output the next free general purpose register and make it current.

- +2 Output the next free floating point register and make it current.

- Output two general purpose registers. The first one is the previous register, the second is the current register. When the operation completes the previous register will be made current. The output is always a digit pair. This format is specialized to take advantage of the register to register operations available on the Spectra 70 class of processors.

- 2 Output a pair of floating point registers. The action is the same as the semantic operation for general purpose register pairs.

One final set of elements of a META PI statement have yet to be discussed namely the Meta Syntactic Commands. These commands are included primarily to permit efficient backup facilities in the user's compiler.

META syntactic commands

.LATCH(name) This causes code to be generated in the user's compiler that will result in the routine named in parentheses being called. In addition, if the routine (or any routine sub-

sequently called by the latched routine) exits to the error routine, backup will be affected.

C

This command can occur wherever a semantic operator can occur; it causes code to be generated in the user's compiler that will suppress the occurrence of a .LATCH in the calling routine. This command is generally used when initial ambiguity in a sub-expression has been resolved. Typical examples are when the first comma is detected in a FORTRAN DO statement, or when the logical operator is detected in a logical IF statement. Backup will not occur if a subsequent syntactic error is discovered and the error pointer will more clearly reflect the location of the error in the input statement.

.CLAMP

This command can occur wherever C can occur. It directs the compiler to suppress all preceding .LATCH's that are still in effect. .CLAMP is useful when .LATCH did not occur on the immediately preceding level, or when it is desired to inhibit the PI compiler or META PI from later attempting to scan input intended for the user's compiler. The reader is reminded that META PI, FORTRAN PI and the user's compiler are, in fact, part of an integrated language system.

The task now is to describe how the META PI elements are formed into statements which are used to create a user's compiler. The approach to be used in accomplishing this end will be via example. First several simple examples will be described. Then the entire META PI implementation of FORTRAN PI will be included as an appendix.

EXAMPLE 1.

FORTRAN PI allows the user to include comments in each statement after a concluding semicolon. If the user did not want this feature, but rather desired to permit multiple statements on one line (similar to ALGOL), he could write the following META PI command:

```
USERCC: = LABST.NOP(.CLAMP)$LABST
```

where LABST refers to the FORTRAN PI definition of a (possibly) labelled statement (see Appendix). The meta syntactic command .CLAMP disables the backup mechanism, and allows the error pointer to clearly reflect the location of a possible error in the subsequent arbitrary sequence of labelled statements (\$LABST). Thus, the program segment

```
X = 1 + Y
Z = SIN(X) + W
Y = Y + 10
PRINT 1,X,Y,Z
```

could become

```
X = 1 + Y; Z = SIN(X) + W; Y = Y + 10; PRINT
1,X,Y,Z
```

EXAMPLE 2.

There is no efficient way to shift logically in the FORTRAN IV language. A FORTRAN PI user at RCA Laboratories required such a shift in order to improve the efficiency and readability of his program in which he made extensive use of bit manipulation. He used the following META PI statement:

```
USERCC : = :SHIFT: .NOP(.CLAMP) (:L: .SAV
(:89:)/
:R: .SAV(:88:)) .ID (INTV) ;: IEXP1
.OUT(:58102000:R:103000:).OUT
(:50102000 05E9,;-.E901)
```

This permitted him to enter statements like

```
SHIFTR J, 3      shift the variable J 3 bits to the
                  right.
SHIFTL K, J + 5  shift the variable K J + 5 bits
                  to the left.
```

Note the use of .SAV(...) to save the op-code of the shift instructions. INTV and IEXP1 are references to FORTRAN PI syntax. The "05E9" (BALR 14,9) constitutes a return to the executive and allows variable tracing (and other debugging aids). The fact that this is an assignment statement is communicated at compile time via the .E901 function.

EXAMPLE 3.

To further illustrate how META PI can be used to create new compilers, two statements from the imple-

mentation of Dartmouth BASIC language alluded to later will be discussed. The BASIC statements have been selected on the basis of their ability to convey the structure of META PI statements and not on the simplicity or complexity involved in their actual implementation.

The BASIC READ statement

This statement has the BNF format

< READ statement > ::= READ < read list >

In META PI the statement becomes

READ: = :READ:RID\$(:,RID):;

META PI will scan the statement from left to right generating the following code:

1. A test for the word READ.
2. Linkage to the definition RID. This is a definition contained within the META PI definition of BASIC.
3. Instructions to effect iterative loop that will test for a comma followed by a read identifier.
4. A test for the line termination character “;”. This character is appended to the statement internally.

This META PI definition is totally syntactic. The semantics for the READ statement are handled in the RID definition.

Handling relational operators

BASIC allows six relational operators; these operators are used within the BASIC IF statement; the operators permitted are:

<i>BASIC operator</i>	<i>Interpretation</i>
<>	not equal
<=	less than or equal to
>=	greater than or equal to
=	equal to
<	less than
>	greater than

The META PI definition for a relational is as follows:

REL = :< >: .SAV(:7:)/:< =: .SAV(:6:)/
 :> =: .SAV(:A:)/:=: .SAV(:8:)/
 :<: .SAV(:4:)/:>: .SAV(:2:)

META PI will generate the code equivalent to a sieve on the six possible relational operators. When one of the operators is detected a single character is entered into the pushdown list. This is effected by the .SAV semantics routine. This character is in fact the actual machine code representation of the branching condition. The REL definition is a sub definition of the IF statement. During the scan of the IF statement the character previously entered into the stack by REL will be popped into the output area and the complete branch instruction will be generated.

EXAMPLE 4.

The preceding example have shown how META PI provides a vehicle to allow user controlled generation of code which may be executed later at run time. The following example shows that the user can also control the generation of code to be executed at compile time, that is, he can generate a compiler-compiler. The example shows, first, the definition of a familiar language called BNF. Then in the new BNF language a simple syntax checker is defined. Then some test strings are entered.

/PRINT# BNF

```

10 USEROC:=.NOP(.CLAMP)BNF;:/:<: .ID.LABEL(*)>: :::: :::: :=:BX1;:::0
    UT(:27FA:)
20 BX1:=BX2:(!: .OUT(:58E.:*1).OUT(:278E:)BX2).LABEL(*1)
30 BX2:=BX3.OUT(:58E.:*1).OUT(:277E:)$ (BX3.OUT(:477.ERR:)).LABEL(*1)
40 BX3:=:<:(:EMPTY:.OUT(:2422:))/ .ID.OUT(:41E.:*).OUT(:452.LATC:))>:
    /STRING.OUT(:45E.TEST:).OUT(::::#R:::)
50 STRING := ALPHABET .SAV(*) $(ALPHABET .SAV(R*))
    
```


3. .EL is a pseudo semantic operation. It actually performs a test for a ; and "return"s or generates the "SHOULD END HERE" message.
4. .ERR(:....:), when used, denotes the actual error message to be displayed if the preceding test fails.
5. EFF OFF and EFF ON are special commands to the off-line compiler tell it to turn off and on some special internal optimizing code.
6. The X semantic operation here is identical to the

on-line Z.

7. There are several subroutines referenced but not defined. This is usually because they have been partially hand coded. At any rate, the explanation of all the features of the off-line compiler-compiler is beyond the scope of this document.

Any of these FORTRAN PI routines can be accessed by the user (via his own compiler).

A-1

```

INTV=(.IID,OUT(:41:+: *)/.IPR,OUT(:58:+: *) )SUBEXP;
RLV=(.FID,OUT(:41:+: *)/.SPR,OUT(:58:+: *) )SUBEXP;
RLDV=(.TYPE(:EFA0:),OUT(:41:+: *)/.TYPE(:EFA2:),OUT(:58:+: *) )DSUL ;
CMPXV=(.TYPE(:EFA0:),OUT(:41:+: *)/.TYPE(:EFA2:),OUT(:58:+: *) )CSUBXP;
IEXP1=IEXP2$(:+:ITRM,OUT(:1A:-E2)/:-:ITRM,OUT(:1B:-E2));
SEXP1=SEXP2$(:+:STERM,OUT(:3A:-2E2)/:-:STERM,OUT(:3B:-2E2));
DEXP1=DEXP2$(:+:DTERM,OUT(:2A:-2E2)/:-:DTERM,OUT(:2B:-2E2));
CEXP1=CEXP2$(:+:.SAV(:A:S)/:-:SAV(:B:S)}CTERM,OUT(:2:R-4),OUT(:2:R-4));
BEXP1=BTERM$(:+:BTERM,OUT(:16:-E2));
SUB2ST=.OUT(:47F09ED40080:R:00:)(:(:SUBRSC,OUT(:947F100:R:)):)/,EMPTY
  .OUT(:C000:)).NOP(.EL);
EQST=.LATCH(DUST)/EQ2ST;
NEQST=DECST/LABST/ENDST/SUBST/FCARD;
EQ2ST=.LATCH(LIFST)/.RLATCH(IJST)/ASST;
LABST=GUST/.LATCH(IFST)/:RETURN:;,OUT(:47F09E98:E908)/ENDOST;
DOST=:DU:;.SAV(*).INT.SAV(*: ;).ID.IID.SAV(*).SAV(E9875).OUT(:41:+: 1*)=;IEXP1
  .OUT(:50302000:);.IGN(C-)IEXP1.ERR(:NOT INTEGER:)(:;.IEXP1.ERR(
  :NOT INTEGER:)/.EMPTY,OUT(:41:+:00001:)),OUT(:411:RE986).OUT(:45E09E22 :
  --E901).LABEL(*1).NOP(.EL);
LIFST=:IF:(:.LATCH(LSUBIX).OUT(:19:-E2).IGN(-)/
  DEXP1(RELOP.NOP(C))DEXP1.ERR(:NOT AN EXP:).OUT(:29:-2E2).IGN(-2));:);
  .ERR(:MISSING :).OUT(:58E :*1).OUT(:47:R:09034:))L2ST.ERR(:NOT A STATEMENT:
  ).LABEL(*1);
GUST=:GUTU:(.INT,OUT(:58E :*),OUT(:47F0904C:F902)/:(.OUT(:58E :*1).OUT(:05:+:
  E:).OUT(:4120000141F0904C:))GOINT.ERR(:NOT AN INTEGER:)$(:;.GOINT.ERR
  (:NOT AN INTEGER:)):).ERR(:MISSING :).OUT(:45E09018:).LABEL(*1).OPT(:;.
  IEXP1,OUT(:07F2 :--E902)).NOP(.EL);
ENDST=:END:;.OUT(:47F09038:E900)ENDSB.ERR(:UNTERMINATED DO LOOP:);
ENDOST=:CONTINUE:;.OUT(:05E9:E900)/CALLST/LIFST/CST/IJST/;.
  (:OUT(:.STRING,OUT(*E900)/:LABEL(:.STRING.LABEL(*)):););
ST=.NOP(C)/.LATCH(CCST)/.DDEND.INT.LABEL(*): :(.EQUALS(EQ2ST)/
  ENDOST).ERR(:INVALID DO END:);
DDGEN/: :(.EQUALS(EQST)/NEQST)/:B:BDLST/.INT.LABEL(*): :.ERR(:BAD LABEL:);
  (.EQUALS(EQST)/LABST)/FCARD/.EMPTY: ;.ERR(:BAD LABEL:)).IGN().IGN().IGN();
FCARD=( :F :/:EXTERNAL:);
  .ID.ERR(:BAD LABEL:))CALLSB$(:;.ID.ERR(:BAD LABEL:))CALLSB).NOP(.EL);
L2ST=.EQUALS(EQ2ST)/LABST;
IEXP2=::-:ITRM,OUT(:13:OFOF)/+:ITRM/ITRM;
SEXP2=::-:STERM,OUT(:33:00)/+:STERM/STERM;
DEXP2=:+:DTERM/:-:DTERM,OUT(:33:00)/DTERM;
BTERM=BPRIM$(:*:BPRIM,OUT(:14:-E2));
ITRM=IPRIM$(:*:IPRIM,OUT(:181:OFE2).IGN(-),OUT(:1C0:OF:18:OF:1:))/
  :/;.SAV(OF)IPRIM,OUT(:180:R:8E000020100:OF).IGN(-).OUT(:18:OF:1:));
STERM=SPRIM$(:*:SPRIM,OUT(:3C:-2E2)/:/:SPRIM,OUT(:3D:-2E2));
DTERM=DPRIM$(:*:DPRIM,OUT(:2C:-2E2)/:/:DPRIM,OUT(:2D:-2E2));
CEXP2=:+:CTERM/:-:CTERM,OUT(:33:00:33:XX)/CTERM;
CTERM=CPRIM$(:*:SAV(XX)CPRIM/;/:SAV(XX)CPRIM,OUT(:45E09BDC0:XX:0:))
  .OUT(:45E09BB00:R:0 :-2).IGN(-2));
SUBEXP=SUBSCL,OUT(:1E:OF:1:))/,EMPTY;
DSUBXP=SUBSCL,OUT(:1E11E:OF:1:))/,EMPTY;
CSUBXP=SUBSCL,OUT(:1E11E11E:OF:1:))/,EMPTY;
ASST=.ID(INTVL=:;.ERR(:EXPECTED = HERE:))(.LATCH(IEXP1).OUT(:50302000 :-)/
  (.RLATCH(SEXP1)/.RLATCH(DEXP1)/CEXP1.IGN(-2)),OUT(:45E09E6050002000 :-2))/
  RLVL=:;.ERR(:EXPECTED = HERE:))(RHIEXP/
  .LATCH(SEXP1)/.RLATCH(DEXP1)/CEXP1.IGN(-2))
  .ERR(:NOT AN EXPRESSION:).OUT(:70002000 :-2)/RLDVL=:;.ERR(:EXPECTED = HERE:);
  (RHIEXP/.LATCH(DEXP1)/CEXP1.IGN(-2))
  .ERR(:NOT AN EXPRESSION:).OUT(:60002000 :-2)/CMPXVL=:;.ERR(:EXPECTED = HERE:);
  (RHIEXP,OUT(:2F:+20)/CEXP1).ERR(:NOT AN EXPRESSION:);
  .OUT(:602020086C002000 :-2-2)).OUT(:05E9 :-E901).NOP(.EL);

```

```

RHIEXP=,LATCH(IEXP1),OUT(:180345E09E00 :-+2);
IFST=:IF(:(,LATCH(IEXP1),OUT(:1222 :-)/(:,RLATCH(SEXP1)/DEXP1)
      ,ERR(:BAD EXPRESSION:),OUT(:3200 :-2))IFEND,ERR(:MISSING :));
BDDLST=,DUEND,INT,LABEL(*): :BASST(DUGEN)/(:,INT,LABEL(*): :/: :)(,EQUALS
      (BASST)/BIFST);
DUGEN=(,OUT(:411:RE986),OUT(:412 :RI),OUT(:58E :R),OUT(:45F09E9C:E903),MORDD);
CCSI=( :/:,EMPTY)(,LATCH(UCC)/,ID,LABEL(*): :=:,NOP(C)CCX2*( :/:,OUT(:078A:)
      CCX2): :/,OUT(:07FA:));
CST=( :FLOWON:,SAV(:028:)/:FLOWOFF:,SAV(:02C:)/:STUP:,SAV(:038:)/:PAUSE:,OUT(
      :92108000:),SAV(:024:)/ICEST)
      ,OUT(:45E09:RE900),NOP(.EL)/NICEST;
SUBST=( :SUBROUTINE:/:SUBR:,),ID,ERR(:INVALID NAME:),OUT(:47F09038:E900)SUBRSB(
      SUB2ST),OUT(:05E9:);
IDST=(ALGI0ST/
      ((:READ(:,SAV(:018:)/:WRITE(:,SAV(:000:))IEXP1,ERR(:NOT AN INTEGER:): :),
      ,INT(FLB),ERR(:BAD FORMAT LABEL:),SAV(:581 :*): :),ERR(:MISSING :)/
      (:PRINT:,SAV(:000:)/:READ:,SAV(:018:))
      (,INT(FLB),SAV(:581 :*)/,EMPTY,SAV(:4110B02C:)),OUT(:1F2:+)),OUT(R),OUT(:58F0B
      :R),OUT(:05EF :-),OPT(:,:)(IDSEQ$( :,:IDSEQ)/,EMPTY)
      ,OUT(:45E0F00C:)/
      (:REWIND:,SAV(:008:)/:BACKSPACE:,SAV(:00C:))IEXP1,ERR(:NOT AN INTEGER:)
      ,OUT(:58F0B00045E0F:R: :-)),OUT(:05E9:E900),NOP(.EL);
GDINT=,INT,OUT(:1F32:),OUT(:58E :*),OUT(:072F:);
LSUBIX=IEXP1(RELOP)IEXP1;
RELIP=: :(:LE:,SAV(:3:)/:EQ:,SAV(:7:)/:NE:,SAV(:9:)/:GT:,SAV(:D:)/:GE:,SAV(:5:)/
      :LT:,SAV(:B:)),ERR(:BAD OPERATOR:): :),ERR(:SHOULD BE A ,:)/:<:,SAV(:B:)
      /:=:,SAV(:7:)/:>:,SAV(:D:);
BASST=,ID(INTVL/RLVL):=:ERR(:SHOULD BE =)BEXP1,ERR(:NOT BOOLEAN:),OUT(:5030
      200005E9 :-E901),NOP(.EL);
SUBSCL=( :GINDEX,ERR(:NOT AN ARRAY:))IEXP1,ERR(:NOT INTEGER EXPRESSION:)
      (MIEXPS),OUT(:180:OF:45E0E004 :-): :),ERR(:MISSING :));
BPRIM=,CHCON,OUT(:58:+,C4GEN)/(:,FALSE,/:0:),OUT(:1F:+OF)/
      ,BCONST,OUT(:58:+,BCGEN)/:-: BPRIM,OUT(:57:OF:0B01C:)/
      (:,TRUE,/:1:),OUT(:48:+:0B01C:)/:(BEXP1:),ERR(:MISSING :)/
      ,ID(INTV/RLV),OUT(:58:OF:0:OF:000:E1);
CALLST=:CALL:( :CHAIN(:IEXP1:):ERR(:MISSING :)),OUT(:0AD1 :-)/
      ,IDCALLSB,SAV(*)((:PLIST:),ERR(:EXPECTED :))
      /PLIST),OUT(:58F :R),OUT(:05EF:E904),NOP(.EL);
ICEST=( :TRACE:( :ON:,SAV(:044:)/:OFF:,SAV(:048:))/:DUMP:,SAV(:03C:)
      /:PDUMP:,SAV(:040:))PLIST;
IPRIM=IPRI;
SPRIM=SPRI$( :*: (:,LATCH(IPRI),OUT(:180:OF:2),IGN(-),OUT(:45E09FBEO:00:1:)/
      ,OUT(:45E09FD40:00:1:))SPRI,OUT(:3C:-2:45E09FCCO:00:1:));
DPRIM=DPRI$( :*: (:,LATCH(IPRI),OUT(:180:OF:2),IGN(-),OUT(:45E09FBEO:00:0:)/
      ,OUT(:45E09FD40:00:0:))DPRI
      ,ERR(:ILLEGAL EXPONENT:),OUT(:2C:-2:45E09FCCO:00:0:));
CPRIM=CPRI$( :*: ,OUT(:45E09CB00:XX:0:),SAV(XX)CPRI,OUT(:45E09BB00:R:0 :-2),IGN
      (-2),OUT(:45E09C660:XX:0 :));
IDSEQ=( (:,OUT(:58E :*) ,OUT(:05:+:E18:+:1:))$,LATCH(PARCOM),ID,IID,OUT(:1F :*)
      ,SAV(E987S),OUT(:411:RE986),OUT(:45E09EFC181:OF: :-)
      ,OUT(:07A:OF:07F1:),LABEL(*1),SAV(*)=:,ERR(:EXPECTED = HERE:),NOP(.CLAMP)
      IEXP1,ERR(:BAD EXPRESSION:),OUT(:50:OF: :R),IGN(-): :),ERR(:MISSING ,:))
      IEXP1,ERR(:BAD EXPRESSION:)( :,:IEXP1,ERR(:BAD EXPRESSION:)/,EMPTY
      ,OUT(:41:+:00001:)),OUT(:411:RE986),OUT(:180:OF: :-),OUT(:18F:OF: :-)
      ,OUT(:90F01000051:OF: :-): :),ERR(:MISSING :))
      /,SAV(:0:S)IOPARAM,OUT(:45E09E86:);
PARCOM=IDSEQ,: :;
INTVL=,TYPE(:FF45:),LEVL,ERR(:LEFT SIDE IS FUNCTION:),OUT(:41:+:0D030:)/INTV;
RLVL=,TYPE(:FFC5:),LEVL,ERR(:LEFT SIDE IS FUNCTION:),OUT(:41:+:0D030:)/RLV;
RLDVL=,TYPE(:FF85:),LEVL,ERR(:LEFT SIDE IS FUNCTION:),OUT(:41:+:0D030:)/RLDV;

```

```

CMPXVL=.TYPE(:FFA5:).LEVL,ERR(:LEFT SIDE IS FUNCTION:),OUT(:41:+:0D030:)/CMPXV;
BIFST=:IF(:BEXP1,ERR(:NOT BOOLEAN:),OUT(:1222 :-)IFEND,ERR(:SHOULD BE A ));
IFEND=:);INT,ERR(:NOT AN INTEGER:),OUT(:41F0904C:);OUT(:58E :*),OUT(:074F:)
;,:ERR(:MISSING ,:),INT,ERR(:NOT AN INTEGER:),OUT(:58E :*),OUT(:07CF:E3)
;,:ERR(:MISSING ,:),INT,ERR(:NOT AN INTEGER:),OUT(:58E :*),OUT(:07FF:E3
E902),NOP(.EL);
CCX2=%CC0(CCX3),OUT(:58E.:#1),OUT(:077E:)%$(CC0/CCX3,OUT(:477,ERR:))
.LABEL(*1E90);
NICEST=((:EXECUTE:.DD(:SR 8,8:))/SAVE;
(.DD(:LA 8,36:):SOURCE:/,DD(:LA 8,8:):OBJECT:/,DU(:LA 8,44:),DPT(.EMPTY))/
SQUEEZE:.DD(:LA 8,16:))/CALC(.DPT(:ULATUR:),DD(:LA 8,28:))
.OPI(.EMPTY),OPT(ONOFF),OPT(.EMPTY)
/:BATCH:.DD(:LA 8,24:));,:ERR(:SHOULD END HERE:),DD(:L 1,SAVSTK:);
DD(:EX 0,*+8(8:)),DD(:B *+56:),DD(:USING STACK,1:);
DD(:OI SW,1:),DD(:NI SW,254:),DD(:NI SW,253:),DD(:OI SW,2:);
DD(:NI CLINE,251:),DD(:OI CLINE,4:),DD(:OI CLINE,128:);
DD(:OI SW,131:),DD(:NI SW,124:);
DD(:NI SW,127:),DD(:OI SW,128:),DD(:NI SW,125:),DD(:OI SW,130:);
DD(:DROP 1:),DD(:SPM 2:));
ALGI0ST=((:READ(<:.SAV(:0:))/PRINT(<:.SAV(:4:)),OUT(:580 :#1)
,OUT(:5000C0685000C06C:),OUT(:58F0802:R)PWDRDC$(,;<:PWDRDC)
,OUT(:1FE50E0C06:858E0C06C07F9:E900),LABEL(*1:));,ERR(:MISSING :)),NOP(.EL);
URG ALGI0ST+14 FOR TABLE GENERATION ONLY
CPRI=.LATCH(CELMF)CEXP1:);,OUT(:45E09:R:0:XX:0:))/
XCON(,;:XCON,ERR(:NOT A NUMBER:))/,EMPTY,OUT(:2F:+20:))/
.LATCH(XCONST)/:(:CEXP1:);,ERR(:MISSING :))/
.ID(CMPXV,OUT(:68:+2:0:OF:00068:+2:0:OF:008 :-))/,TYPE(:F5A5:))FCN
.SAV(RS),OUT(:68:+2:0:R:03068:+2:0:R:038:))/DPRID,OUT(:2F:+20:));
DPRID=.LATCH(ELEMF)DEXP1:);,ERR(:MISSING :)),OUT(:45E09F:R00:0:))/LATCH(ABSF)
DEXP1:);,ERR(:MISSING :)),OUT(:30:00)/XCON/:(:DEXP1:);,ERR(:MISSING :))/
.ID(DPRID),ERR(:BAD TYPE:));
DPRID=RLV,OUT(:68:+2:0:OF:000:E1),IGN(-)/,TYPE(:F585:))FCN
,OUT(:68:+2:0:R:030:))/
RLV.SAV(:78:+2:0:OF:000:E1),OUT(:2F:00),OUT(R: :-)/,TYPE(:F5C5:))FCN
,OUT(:2F:+20:78:0:0:R:030:))/ITORD;
SPRID=.LATCH(ELEMF)SEXP1:);,ERR(:MISSING :)),OUT(:45E09F:R00:1:))/
.CHCON,OUT(:78:+2.C4GEN)/.BCONST,OUT(:78:+2,BCGEN)/
.NUM,OUT(:78:+2,NGEN)/:(:SEXP1:);,ERR(:MISSING :))/
.LATCH(ABSF)SEXP1:);,OUT(:30:00)/.ID(SPRID);
IPRID=.INT,OUT(:58:+.IGEN)/.CHCON,OUT(:58:+.C4GEN)/
.BCONST,OUT(:58:+.RCGEN)/:(:IEXP1:);,ERR(:MISSING :));
/.LATCH(ABSF)IEXP1:);,ERR(:MISSING :)),OUT(:10:0FOF)
/.ID(INTV,OUT(:58:OF:0:OF:000:E1))/,TYPE(:F545:))FCN,OUT(:58:+:0:R:030:));
MIEXPS=,;:IEXP1,ERR(:NOT INTEGER EXPRESSION;);
(MIEXPS),OUT(:180:OF:05EE :-))/,EMPTY,OUT(:41E09F4E1F11:R);
SPRID=RLV,OUT(:78:+2:0:OF:000:E1),IGN(-)/,TYPE(:F5C5:))FCN,OUT(:78:+2:0:R:030:))/
ITORD;
ITORD=(INTV,OUT(:5800:OF:000 :E1-))/
,TYPE(:F545:))FCN,OUT(:5800:R:030:)),OUT(:45E #ITJR:+2:);
FCN=.SAV(*):(:PLIST:);,ERR(:MISSING :)),OUT(:58F :R),OUT(:05EF:E904),SAV(:1:))
/.EMPTY,LEVL,ERR(:BAD FUNCTION CALL:),SAV(I:D:));
PLIST=.OUT(:58:+:0D000:),SAV(OF),OUT(:41:+:0:R:030:))
(PARAN2,SAV(:0:))$(,;:PARAM1,ERR(:NOT A PARAMETER:),SAV(I:4:))(:,;:PARAM2
,ERR(:NOT A PARAMETER:),SAV(I:0:))/,EMPTY),OUT(:9680:OF:00:R))/,EMPTY
,OUT(:92C0:OF:008:)),IGN(-),IGN(-);
ABSF=:ABS;.DPI(:F:);(:;
ELEMFF=((:SQRT:.SAV(:DCO:))/SIN:.SAV(:E40:))/COS:.SAV(:ECO:))/(:LOG:/:ALOG:),SAV
(:D40:))/:EXP:.SAV(:CCO:))/:ATAN:.SAV(:C60:))/:TANH:.SAV(:F40:)),DPT(:F:);(:;
XCONST=((:DEXP1:);,NOP(C)DEXP1,ERR(:NOT AN EXPRESSION:));,ERR(:MISSING :));
XCON=.DNUM,OUT(:68:+2,DCGEN)/.CHCON,OUT(:68:+2,CBGEN)/.BCONST

```

```

      .OUT(:68:+2,B0GEN)/:PI:.OUT(:68:+2:09DF0:);
PARAM1=.SAV(:4:S)PARAM;
PARAM2=.SAV(:8:S)PARAM.OUT(:41:OF:0:OF:008:);
CELEMF=(.SQRT:.SAV(:C90:))/:SIN:.SAV(:CD0:))/:COS:.SAV(:CD8:))/(:LOG:/:ALOG:);
      .SAV(:C80:))/(:MAG:/:ABS:).SAV(:C14:))/:ARG:.SAV(:C2C:))/
      :EXP:.SAV(:C66:))/:ATAN:.SAV(:D3A:))/:TANH:.SAV(:D02:)),OPT(:F:):(;;
ONOFF=:ON:/:OFF:.DB(:LA 8,4(0,8:));
CC0=(.OUT(:/:.IGN(:.OUT(:92FF900A:))%CC01:):.OUT(:05E9:)/
      :.LABEL(:$CC01:):.OUT(:45E,LABE:))/:DO(:$(,SR.OUT(*):):)/:OPT(:CCX1:):/
      :.SAV(:$CC01:):.OUT(:45E,SAV:))/:NOP(:$CC01:):);
CC01=CC0SUB.OUT(:45E.:*)/C:.OUT(:92015000:)/,SR.OUT(:05E4:).OUT(:#:#*:::):);
EFF OFF
CC0SUB=:*1:/:R:/:I:/:+2:/:+:/:S:/:-2:/:-4:/:-:/:X:/:OF:/:O:/:*:/:#:/:ID;
EFF ON
CCX3=.ID.OUT(:41E.:*).OUT(:0503:)/,SR.OUT(:45E.TEST:).OUT(:#:#*:::):)/
      (:CCX1:):/:.EMPTY:.OUT(:0420:))/:$.LABEL(*1)CCX3.OUT(:58E.:*1).OUT(:078E:);
      .OUT(:0420:))/:LATCH(:.ID.OUT(:41E.:*).OUT(:450,LATC:)):)/
      :.TYPE(:.SR.OUT(:45E,TYP:).OUT(*):):)/
      :.ID.OUT(:45E.:*);
CCX1=CCX2*(:/:OUT(:58E.:*1).OUT(:078E:))CCX2),LABEL(*1);
FCNST=:FUNCTION:.ID.ERR(:INVALID NAME:).OUT(:47F09038:E900)FCNSB(SUB2ST)
      .OUT(:45E09EBE:.X2);
SUBV=.TYPE(:1505:).OUT(:58:+: *);

```

The organization and formatting of hierarchical displays for the on-line input of data

by GORDON T. UBER, PAUL E. WILLIAMS and BRADNER L. HISEY

Lockheed Missiles and Space Company
Sunnyvale, California

and

ROBERT G. SIEKERT

Mayo Clinic and Mayo Foundation
Rochester, Minnesota

INTRODUCTION

On-line terminals, which we shall call "list selection terminals," are being investigated here and elsewhere as input devices for information systems. These display lists of alphanumeric entries from which a user may select entries by pointing. Whereas users of most terminals (including ours) point with a light-pen, users of the Control Data Digiscribe select by touching electrically conductive regions on the faceplate with a finger, thus completing a radio-frequency circuit. The lists are displayed on either a cathode ray tube or an optical rear-projection screen, and they may be changed rapidly under computer control. The user composes input to the system by selecting words and phrases from the lists. As he proceeds, new lists are displayed as required.

This approach to computer input takes advantage of the wide bandwidth of this class of displays, and of the human eye, to rapidly convey information to the user; he may then respond manually at a low rate. It is particularly useful for users, such as physicians and managers, who generally are not good typists. Although their entry selection rate with a light-pen is lower than their hunt-and-peck typing rate, the information-input rate when using selections is generally higher than with a keyboard because entries each consist of many characters. In one application, the input of medical laboratory test orders, the entry rate was comparable to handwriting. Although

dictation is a faster method of information entry, it interposes a stenographer and results in delay between the user and the computer system.

This approach has several features that improve user accuracy and system acceptability. This user, presented with a list of acceptable and meaningful responses to the system, is only required to recognize an acceptable response rather than to recall it from his memory. In this sense, the system is tutorial. The tutorial aspect can be further exploited by including explanatory material on the displays themselves. Because an entry can be transmitted to the computer with a single selection, it is not necessary to substitute an abbreviation or a numeric code for the entry in order to shorten the input transmission time. Thus, the errors inherent in manual encoding are eliminated. Spelling errors are eliminated, and through the proper organization of the displays, most syntactic errors can be eliminated as well.

For many applications, the entries are arranged into a hierarchy comparable to that used in information storage and retrieval systems such as AESOP.¹ This hierarchy is essential as an index to the many possible responses that the user may make in entering information. Through the entry process, he becomes familiar with the structure of the hierarchy, and thus, he is familiar with the organization of the content within this system when he desires to retrieve. The entries, although appearing in full form for the user, may have invisible coding associated with them for use by the

computer system. Thus, there is an inherent translation from a man-understandable to a machine-understandable form. The syntactic structure of the user's responses also can be preserved for further processing by the system. Thus, the time-consuming scanning and analysis of character strings that are normally done in information retrieval systems are not required. The syntax can be tightly controlled when the computer must act on the input data and may be left uncontrolled when only narrative input is required for the subsequent reading by users.

In the remainder of this paper, we shall discuss the organization of information hierarchies, search strategies, and the manual and automatic formatting of displays. The examples will be taken from medical information systems for entering laboratory test orders,² medical histories,³ and x-ray reports.⁴

Hardware

For our experiments we are using an experimental Lockheed Video-Matrix Terminal^{2,5} shown in Figure 1. The word "matrix," which refers to a display page, was derived from the row-and-column arrangement of the entries. This terminal of the cathode ray tube has a 24 row-by-40 column display, 7 by 9 inches in size. Although the present display is adequate for the routine entry of medical orders, we would prefer a larger display, for example, 24 by 64 characters, for the entry of narrative data and especially for information re-



Figure 1.—Display and use of light-pen, with experimental video matrix terminal.

trieval. Selections are made by use of the light-pen, and the output signal from the light-pen is fed back into the video signal so that the characters that are aimed at are brightened. Except for information retrieval and automatic formatting experiments, this system has maximal response time of 0.5 second. Such fast response is essential to avoid interrupting the user's thought process.

Function codes

The bottom two rows of the screen (Figure 1) contain light-pen selectable function codes through which the user initiates system actions. In the normal Operational Mode, the code ERR enables the user to sequentially cancel the effects of as many as 16 previous light-pen selections. The code BACK permits him to sequentially back up to 16 previous matrices. As the user makes light-pen selections, New Text is accumulated, the last two lines of which appear at the top of the screen. The entire page of New Text is automatically presented to him (New Text Mode) when a full page of message has been accumulated. The user also may enter the mode by selecting NEW TEXT. In either the Operational or New Text Modes, a typing cursor is displayed when a key is depressed. This cursor indicates where the keyed character is to appear and may be positioned either with the light-pen or with the carriage return, backspace, tab, and space keys. Characters may be keyed in or erased at the position designated by the cursor. In the New Text Mode, the function code ASBL (ASsemble) closes in any gaps left by erasure. The New Text may be entered permanently into this system with code ENTER or may be totally erased by CLR (CLear). The permanent patient record may be accessed by the function code OLD TEXT. The user may page through this record in the forward direction either a full page or a half page at a time by NEXT or HALF or in a backward direction by BACK. The Operational Mode is entered by selecting RETURN.

Matrices may be created and modified on-line in the Edit Mode, accessible to systems programmers. Each light-pen selectable entry is terminated by the entry marker character (§), which is visible only in the Edit Mode. By means of special function codes and the cursor-positioning features, entries may be moved vertically and horizontally. New entries also may be inserted. After an entry has been selected in the Edit Mode,

a function code enables the control data for that entry to be accessed. These control data define the following for the Operational Mode:

1. The character string to be added to the New Text. This may be from any matrix.
2. The next matrix to be displayed. This is often the present matrix.
3. Punctuation to precede the new character string before it is added to the New Text.

These control data may be modified in the Edit Mode. The ability to do on-line editing has enabled the rapid development of our present set of about 2,000 matrices.

A logging feature enables us to record and subsequently reconstruct the actions of a user at a terminal. All light-pen selections and the time interval preceding each may be printed out. Long intervals indicate areas of user difficulty to the matrix designer. This feature is also a potential source of successor probabilities for use in the optimal design of hierarchies as described below.

Data hierarchies

To facilitate their retrieval, selectable entries are arranged into a multilevel hierarchy. An example of such an entry hierarchy is a set of medical order matrices reported by Siekert and associates² from which physicians may order any combination of about 500 laboratory tests (Figure 2). Since these test names will not fit on a single

matrix, it is necessary to organize them into a number of matrices. It is further necessary to index these matrices so that the user may find the matrix containing the test that he wishes to order. There are numerous ways in which these test names may be classified. Let us examine several of them. Most commonly used and easily understood is the alphabetic arrangement of entries. The top level matrix can contain the letters A to Z. Selection of a letter such as "A" takes the user to the matrix containing tests beginning with "A" (lower left corner of Figure 2). In this instance, two matrices are required to contain all of the tests named. The entries on these matrices can be arranged in alphabetic order, but if the user wishes an entry appearing near the end (for example, Autohemolysis-RBC), he must page through to the last matrix in order to select it. If such an entry occurs with great frequency, this is a nuisance. To reduce the number of selections required to reach a common item, a second classification method may be used: that of frequency or probability. The common tests beginning with the letter "A" may be placed on the first matrix, the less common on succeeding matrices. Because a test name can be overlooked, we have found it desirable to repeat the common tests with the less common to ensure that they can be found.

Such alphabetic hierarchies work efficiently when only a single test is to be ordered. Often a group of perhaps 10 tests is required. To facili-

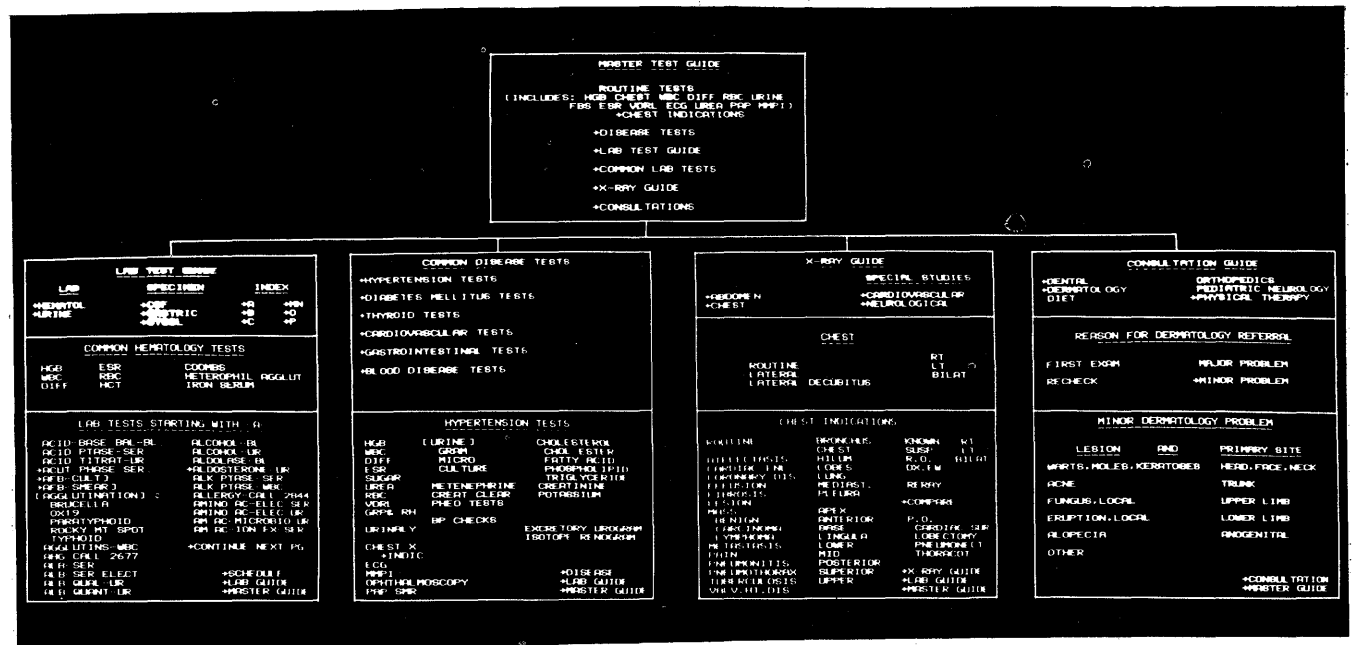


Figure 2.—Hierarchy of laboratory test ordering.

tate the selection of such groups of entries, several alternate classification methods have been developed. The user may use any or all of them as he wishes. In particular, test names have been classified by the specimen on which they are performed (such as cerebrospinal fluid), the laboratory in which they are performed (blood chemistry), and the disease category (hypertension) with which they are most commonly associated. The disease categories are most commonly used because the users are generally able to order a number of tests rather than just one from a given disease test matrix.

In any classification system based on meaning, different people may place the same item in different categories. To ensure that a user will find an entry in the category to which he assigns it, entries are placed in all applicable categories. Users may use different names for a given test or different permutations of the words in the test name. All such synonyms are included. Thus, the test for serum alkaline phosphatase is listed both under "alkaline phosphatase-serum" and under "phosphatase, alkaline-serum." It should be noted that the probability of a given test being ordered varies with different users or different classes of users. Furthermore, the probabilities vary with time, as various tests are introduced and in turn superseded by newer tests. If the number of selections required to select a test is to be minimized, then the system should have the capability to modify the matrix content for each user and as the probabilities of ordering vary. The automatic formatting procedures described in subsequent sections are a step toward this capability.

Search strategy

Let us organize the hierarchy so that the user can select the entries he wishes with a minimal number of selections. One selection is required either to go from one list to any other to which it is connected or to select an entry on a list. Initially, let us consider the selection of a single leaf (an entry which does not link to another list) with the user starting at the top of the structure. We shall consider only the probability of an entry's being selected and shall in general follow the synthesis procedure developed by Huffman.⁶ If all of the items are equally probable, then an optimal hierarchy is one in which the entries are on the lowest level, with the higher levels being indices to the

lower level lists. If some items are much more probable than others, however, then these higher probability entries should occur higher in the hierarchy. The criterion for placement follows Huffman's synthesis of minimal length codes. Assume that we have N items to be organized into a hierarchy of lists, each list having a capacity of B items. We rank the items by their probability and assign the B lowest probability items to a list. (In practice we may have to add dummy items to make all of the lists full during the synthesis; these are deleted when the synthesis is complete.) We replace the B items on the list thus formed by a new item that is assigned a name and has a probability equal to the sum of the probabilities of the items in the list. We have thus eliminated $B-1$ items from the pool of those to be assigned. We re-rank the pool and repeat the assignment. Doing this until all items are continually assigned, we reach the point at which names (that is, lists) are being assigned, and the synthesis of the next-to-the-bottom level in the hierarchy has begun. With the creation of the top level list, the pool is exhausted.

Although this procedure has produced an optimal hierarchy, it has not satisfied the condition that the user must know which list a given item is on. The lists must contain meaningful combinations of items, and the list names must relate meaningfully to their contents. Fortunately, items may be reassigned to different lists at the same level so as to meet semantic criteria without affecting optimality.

Thus far we have considered the selection of a single entry. If a sequence of entries is to be selected for which the probabilities are independent of previous selections, then we may still achieve optimality merely by automatically transferring the user to the top of the hierarchy after each selection of a leaf. In practice, successive entries are generally related so that a single hierarchy is not optimal. If the size of the structure were no limitation and we knew the conditional probabilities, given a choice of succeeding selections, then we could use these conditional probabilities in the Huffman synthesis to yield an optimal set of hierarchies. We are deterred from this approach by the difficulty of storing or dynamically generating such a large set. Extensive training is required to bring the user's selection rate up to a satisfactory level for the large number of matrices which result. In practice, we approximate the optimal set

by providing upward and lateral transfers to previously selected lists and to related lists at the same level.

Optimal list size

If we assume all lists to be of equal length, then the number of leaves in the structure equals the list length raised to the number of levels. To minimize the number of selections required to reach a leaf, we minimize the number of levels by maximizing the list size. The maximal list size is limited by the number of entries that can be placed on a matrix, as will be discussed in the section on display formatting. Experienced users have memorized the position of entries on matrices and can find a given entry in an interval of time that is relatively independent of the number of entries on a matrix. With fewer lists, fewer list names need be assigned, and this simplifies the problem of list identification, as discussed by Fuller.⁵

Tests of inexperienced users revealed an additional psychologic constraint on list length. These users scanned lists entry by entry until they found the entry they desired. They did this even when lists were alphabetized and more efficient search strategies were possible. Assume a hierarchy with N leaves and B items per list, a selection time t and a scan time of $\frac{t}{r}$ per entry. Assume also that, on the average, a user scans half of a list to find an entry which he then selects (although some will scan the entire list to ensure that they have not overlooked anything). The total time to select a leaf is given by the expression

$$\frac{(\frac{1}{2} Bt + t)}{r} \log_b N.$$

Differentiating this expression with N and t constant, we obtain a minimum

$$r = \frac{1}{2} B (\ln B - 1),$$

which is graphed in Figure 3. Typically, the scan time is 0.5 second per entry, and the selection time is 2 seconds per entry, so that r equals 4. The corresponding value of B is 7.7, which is the minimal list length that is optimal under these conditions. Because experienced users know the positions of the entries they wish to select and also can scan the lists more efficiently than with

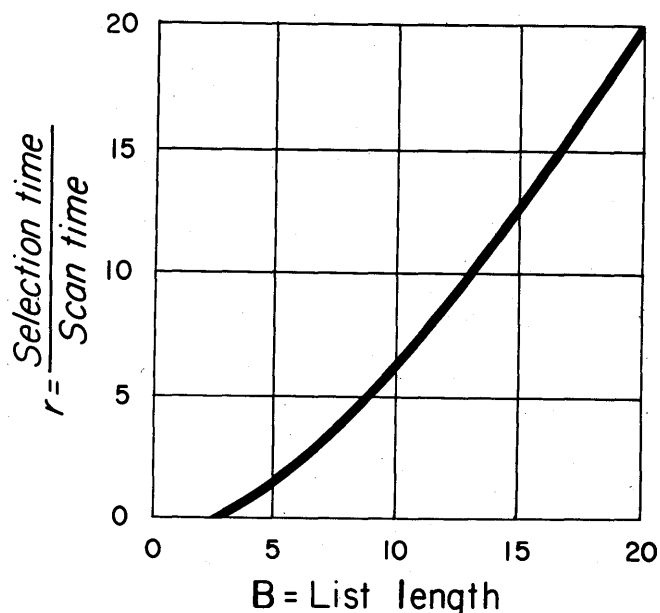


Figure 3.—Optimal relative scan time per entry versus list length.

the method used in the above model, practical values of B range from 10 to 40. Because a matrix usually is composed of several different lists, the optimal number of entries per matrix is greater than B .

Display formatting

The following is a summary of some empiric rules for display formatting. Lists in column form may be scanned more easily than those in row form; at least three columns of blank character should be left between lists. Matrices should be given meaningful titles, and lists themselves should be titled when more than one appears on a matrix, although this latter requirement can be ignored for certain classes of experienced users. To facilitate learning, each area within a class of matrices should have the same function. For example, transfers to other matrices are usually placed in the lower right-hand corner. Although it is possible to include explanatory material within the matrices, we have preferred the alternative of keeping the organization simple enough so that it can be used without explanations after a few hours of training. We have introduced several cueing symbols to guide the users. Nonselectable entries are either underlined or bracketed, and entries that cause transfers to other matrices are preceded by an arrow. Abbreviations, when used,

should be familiar to all users of a set of matrices. Full terms may be inserted into the message area for clarity, even though the entries are abbreviated. Although users tend to concentrate on light-pen selection and to ignore the message, the use of full terms does facilitate communication between groups of users who are unfamiliar with each other's abbreviations. For example, VERT P is an abbreviation for VERTICAL PORTION OF THE ANTERIOR COMMUNICATING ARTERY (Figure 5).

Automatic display formatting

In many of the classes of displays we have developed, a small number of format types is sufficient. Examples include displays for drug ordering, alphabetic lists of laboratory tests, and descriptions of medical symptoms. In such cases, it is feasible to separate the data and the format just as in FORTRAN programs. This makes it possible to standardize the format for a class of displays and to revise the formats of the entire class merely by revising one format definition. Separate procedures may be used for revising the content, which now takes the form of a set of lists. In developing displays for the input of medical histories, for example, the physician selects the descriptors relevant to a symptom from a master list of descriptors. Thus, a headache location, for example, may be BEHIND EYE, FRONTAL, GENERALIZED, or MAXILLARY. A headache may be precipitated by ALCOHOL, BENDING, and so forth. This approach enables the user to develop display content directly in many cases without knowing either computer programming or display programming. Further, it greatly simplifies display updating and maintenance as, for example, when new drugs are made available for use. Although at present our formatting rules are incorporated within the formatting routines, we hope to develop programs in the near future that will accept external format definitions. A system could be designed to maintain statistics on the frequency of selection of each item (either for each user or for all users as a group) and to automatically revise the hierarchy based on these statistics, thus making common items easier to select.

Sample matrices

The following examples indicate the range of

ACETYSALICYLIC ACID		
DOSE / ROUTE	FREQ	MODIFIER
300 MG--PO	STAT	NO MOD
600 MG--PO	STAT &	
900 MG--PO		
300 MG--PR	Q3H	PRN PAIN
600 MG--PR	Q4H	PRN FLVLR
	→OTHER	
→FDS DOSE/ROUTE		→MEDS GUIDE
→OTHER DOSE/ROUTE		→MASTER GUIDE
LANALGESIC, ANTIINFLAMM., ANTIPIRETTIC		
NEW MAST PREV	TYPI TYII	Q3H
ERR TEXT GUID TEXT CLR	HELP COPY	QUIT

Figure 4.—Acetylsalicylic acid matrix for drug ordering.

applicability of the list selection approach. Figure 4 is a matrix for ordering the drug acetylsalicylic acid. This matrix, one of 500 drug matrices, is displayed whenever the physician selects either acetylsalicylic acid (generic name) or one of the two equivalent names, aspirin or ASA. This series is the first of our automatically formatted matrices. The data specific to each drug are stored in compressed form and are expanded into the format shown when the matrix is selected. It is an example of what we call the "Forcing Technique" in which the user cannot leave a matrix until he makes a selection from each of the columns. This technique ensures completeness of the message by forcing the user to remain on the present drug matrix until all three selections are made, for example, "600 Mg—PO, Q4H, PRN PAIN" (600 milligrams by mouth every 4 hours as indicated for pain). Although users can be trained to use this technique, any condition in which the system does not respond to the user leads to frustration and confusion. In this case, if the user attempts to leave this matrix without completing his order, it is desirable to display a message informing him of his omission.

Narrative generation is exemplified by the Arteries matrix in Figure 5, which is part of the system developed by Uber and Baker⁴ for reporting roentgenographic findings of the blood vessels (angiograms) of the head. An experienced neuro-radiologist can use this matrix for generating such statements as "LEFT DISPLACEMENT OF THE ANTERIOR CEREBRAL ARTERY,

```

ARTERY, ELEVATION OF THE CALLOSOMARGIN
AL ARTERY, AND DILATATION OF THE
ARTERIES
E      DISPL  A CEREB  P CEREB  CAR SI
&     ELEV  A COMMUN CALC B   A CHOR
      DEPR  HORIZ P  TEMP B   P COMM
MARK  DILAT  FPOLR B
MOD   STEN  ORBFR B
MIN   OCCL  PCALLOS
      ORIGIN CALMARG  VERTEBR  +PREV
RT    M CEREB  BASILAR
LT    SYL TRI  A I CBEL
MED   TRIFURC  P I CBEL  +VEIN
LAT   ASC FP  PONTINE B +VENT
ANT   P PAR B  I ACOUST  +CIST
POST  ANG B    SUP CBEL
SUPERIOR  A SPINAL  +REGN
INFERIOR  P TEMP  +PATH
SUPERFIC  A TEMP  +XRAY
DEEP     THALMOS +MAST
NEW PREVIOUS <MARY L CRANDAL MLC>
ERR TEXT TEXT CLR HELP COPY ENTER
    
```

Figure 5.—Arteries matrix for neuroradiologic reporting.

MARKED ELEVATION OF THE CALLOSOMARGINAL ARTERY, AND DILATATION OF THE MIDDLE MENINGEAL ARTERY. DEPRESSION OF THE SYLVIAN TRIANGLE.” The system is about half as fast as dictation.

A second example of narrative generation is from the medical history entry system reported by Kiely and associates.³ The matrix (Figure 6) exemplifies the use of the digital keyset. A typical phrase relating to severity of chest pain is “MILD AT ONSET, VARIABLE PAST 2 MONTHS, and SEVERE AT PRESENT.”

In both of these examples of narrative generation, the user is free to select the entries in any sequence he desires. He may, if he wishes, type

```

SEVERITY:
AT ONSET
AT PRESENT
AFTER WALKING
MILD
MODERATE
SEVERE
VARIABLE
UNKNOWN
INCREASING
1 2 3 YRS
4 5 6 MOS
7 8 9 WKS
0 - / DAY
DEC HRS
ERR MIN
SEC
PAST
BLOCKS
AND
+ADVANCE
+CATEGORY GD
+SYMPTOM GD
NEW MAST PREV
ERR TEXT GUID TEXT CLR TYP TYP OPS
HELP COPY BACK
    
```

Figure 6.—Severity matrix for medical history.

in words that are not present on the matrix from the console. If the application requires it, it is possible to prohibit type-in entries and to restrict sequences of entries to those that are syntactically correct. This could be done either by modifications of the “Forcing Technique” or by displaying only those entries that are correct. This approach has proved useful for on-line programming in systems such as AESOP.¹ In a programming language such as COBOL, the key words such as BEGINNING-FILE-LABEL could be single entries. Names of variables could be typed in once and subsequently selected from lists.

DISCUSSION

The art of designing man-machine systems is still in its infancy. List selection terminals, by placing the output burden on the data system, are able to increase the input rate that an untrained user can achieve. By so doing, terminal operation is made feasible for a much broader class of users. So far this approach has proved useful in applications in which the vocabulary is limited to several hundred words. We are just beginning to develop the automatic formatting procedures that will expedite the design of the next generation of matrices. The potentials of information systems that adapt to the user’s response patterns are yet to be realized. To the retriever, this approach offers the ability to control the quality of the data at the time that they are entered without, we hope, placing an undue burden on the enterers. (For other experiments with on-line terminals see the paper by Douglas C. Engelbart elsewhere in this Proceedings.) Although we have been heartened by our limited successes in facilitating man-machine communication, we have at the same time been humbled and challenged by our ignorance of how a dialogue should be structured, how we should mold the machine to fit the man. It is perhaps in this area that the next advances will be made.

ACKNOWLEDGMENTS

We thank our colleagues on the Mayo Medical Information System Project and in the Lockheed and Mayo organizations for their help. These experiments would not have been possible without the dedicated efforts of our programmers, Miss Ruth Short and Mr. David C. Schrantz, both of whom have made substantial contributions to this work.

REFERENCES

- 1 E BENNETT E C HAINES J K SUMMERS
AESOP: A prototype for on-line user control of organizational data storage retrieval and processing
AFIPS Conference Proceedings Vol 27 Part I 435-455 Spartan Books Washington DC 1965
- 2 R G SIEKERT B L HISEY P E WILLIAMS G T UBER
A video terminal/light-pen device for ordering medical tests
JAMA In Press
- 3 J M KIELY J L JUERGENS B L HISEY
P E WILLIAMS
A computer-based medical record: Entry of data from the history and physical examination by the physician
JAMA 205: 571-576 August 19 1968
- 4 G T UBER H L BAKER JR
System for recording neuroradiologic diagnoses in a computer
Radiology In Press
- 5 W D FULLER
Physician-machine interface in a hospital information system
Proceedings of 8th National Symposium on Information Display May 24-26 1967 pp 111-122
- 6 D A HUFFMAN
A method for the construction of minimum-redundancy codes
Proc Inst Radio Engineers 40:1098-1101 September 1952

Projections of multidimensional data for use in man-computer graphics

by THOMAS W. CALVERT

Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION

The utility of man-computer graphics as an engineering design tool is now well recognized.¹ With the use of an interactive visual display it is possible for a designer to input information graphically or digitally, observe the results of his input and then modify his data to achieve a more satisfactory result. By using preprogrammed subroutines the designer can quickly perform quite complex manipulations on data and observe the result.

Unfortunately, this very powerful design tool, which allows a designer to use both sophisticated mathematical techniques and his own intuition has found little application to a class of problems where the data are essentially multidimensional. Typical problems of this class include:

- a) signal design where a signal is represented as a point in a multidimensional space of time or frequency samples,
- b) controls system design where systems are described by multidimensional state vectors, and
- c) pattern recognition where pattern feature vectors are multidimensional.

Although in a few cases, it may be possible to obtain a useful graphical representation of multidimensional data with an orthogonal projection onto two dimensions, in general this is not feasible. This paper presents a method of obtaining a meaningful two dimensional representation which can be applied to a wide class of problems.

Projections of multidimensional data

Multidimensional data can be represented as a collection of vectors. Consider m vectors each with n dimensions say x_1, x_2, \dots, x_m . These might represent points on a hyperobject (e.g., vertices of a hypercube), feature

vectors describing patterns or state vectors of control systems. The n -dimensional vectors can be projected into a two-dimensional space by a linear transformation e.g. $y = T'x$ (prime denotes transpose) where y is a two-dimensional vector, and T is a matrix with n rows and 2 columns

$$\text{i.e. } T = [t_1, t_2],$$

$$\text{then } y_1 = x \cdot t_1$$

$$y_2 = x \cdot t_2.$$

Thus t_1 and t_2 are two basis vectors for the two-dimensional subspace. It will be seen that there are an infinite set of choices for T , some of which may give a useful representation of the data and most of which will not. Perhaps an obvious transformation to try is that where t_1 and t_2 are chosen to be orthogonal and to represent the data with minimum mean square error. These can be calculated by finding the eigenvectors of a grammian matrix.² Define a mean vector

$$\mu = \frac{1}{m} \sum_{i=1}^m x_i$$

and matrix $X^{(n \times m)} = [(x_1 - \mu), (x_2 - \mu), \dots, (x_m - \mu)]$.

Then the grammian matrix G for the data is

$$G^{(n \times n)} = XX'.$$

The eigenvalues λ_j and eigenvectors u_j for $j = 1, \dots, n$ are found by solving $Gu_j = \lambda_j u_j$. The basis vectors u_1 and u_2 corresponding to the two largest eigenvalues λ_1 and λ_2 give the transformation which projects to a plane with minimum mean-square error. An example of this

projection is shown in Figure 1 where a four-dimensional hypercube is projected to two-dimensions. Most observers find this projection of a simple hyperobject through a modest decrease in dimensionality too difficult to interpret.

This problem has been studied by Noll³ who used perspective projections to produce two slightly different displays side by side. When these displays were viewed appropriately a stereoscopic effect was obtained, so that the observer could obtain a three dimensional impression of the four-dimensional hypercube. Noll's interesting results included the effects of rotating the hypercube. They showed that it was possible to obtain some intuition about a hyperobject from perspective projections if the original dimensionality was four, but that for hyperobjects of five or more dimensions the displays became confusing.

"Pseudo" projections

From the above discussion it seems clear that it would be desirable to have some "pseudo" projection from multidimensional space to two dimensions where the only requirement is that result be meaningful and easy to interpret. No such technique can possibly give a complete representation of n-dimensional data in two dimensions. However, in another context, Shepard and Carroll⁴ have developed an approach which gives poten-

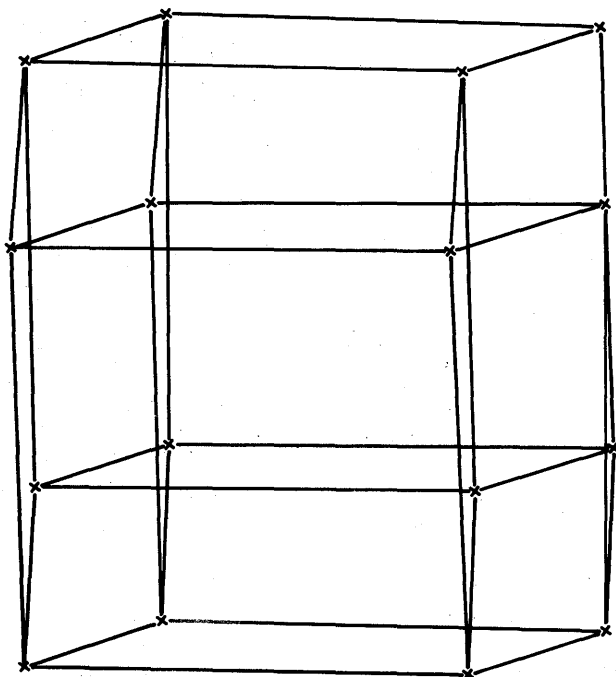


FIGURE 1—Orthogonal projection of a 4-dimensional hypercube. (Vertices are indicated by an "x" and cube edges are shown).

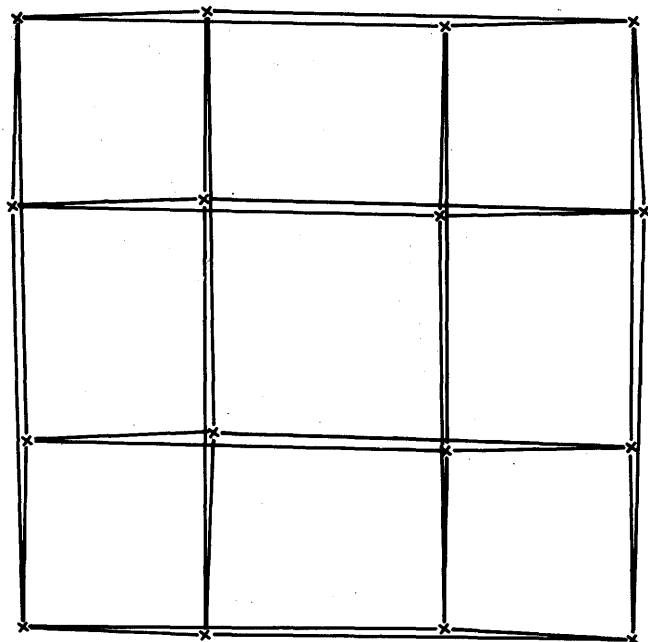


FIGURE 2—Pseudo projection of a 4-dimensional hypercube. (Vertices indicated by an "x" are slightly displaced so that cube edges can be added).

tially useful results. Their method consists of "unfolding" hyperobjects into two-dimensions by maintaining the interrelationships between points which were originally close together. The application of this technique to a four-dimensional hypercube is illustrated in Figure 2. Another example, which has been taken from Shepard and Carroll is shown in Figure 3 where points originally on a three-dimensional torus embedded in four dimensions are shown unfolded into two dimensions. It is claimed that this method is useful since a point in the new two-dimensional space bears approximately the same relationship to its neighbors as it did in the original space. Thus local interrelationships between points can be studied, conveniently. Since unfolding a hyperobject involves "cutting" it open, points on the edge of the two-dimensional representation may originally have been neighbors to points on the opposite edge of the display (c.f. two-dimensional projections of the world).

Shepard and Carroll's method, which was developed to study psychological data, depends on maintaining "continuity" between the original and the new space. This is done by minimizing a criterion function K which is defined as

$$K = \frac{\sum_{ij} \frac{d^2_{ij}}{D^2_{ji}} \cdot W_{ij}}{S}$$

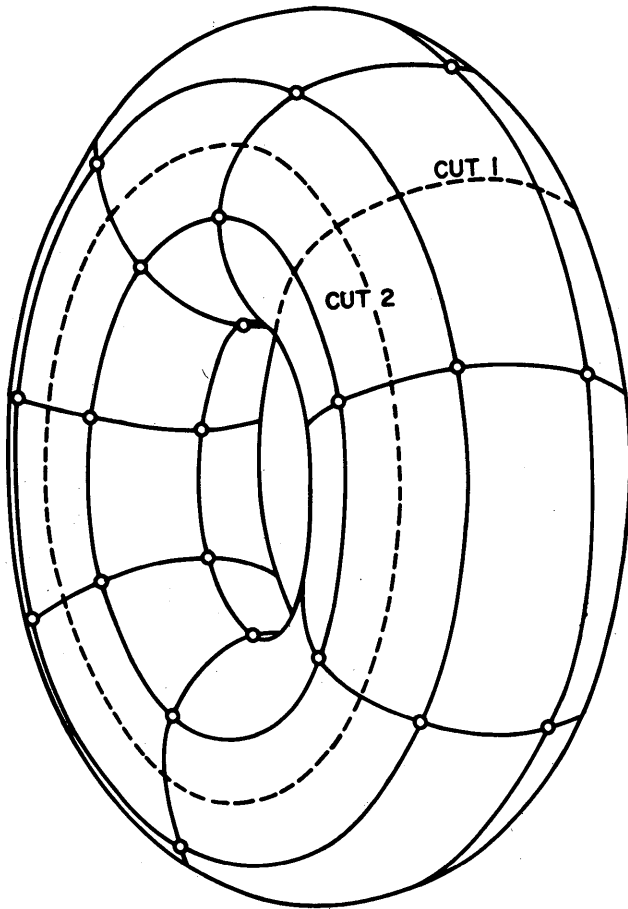


FIGURE 3(a)—An example from Shepard and Carroll.⁴ A three-dimensional torus embedded in four dimensions.

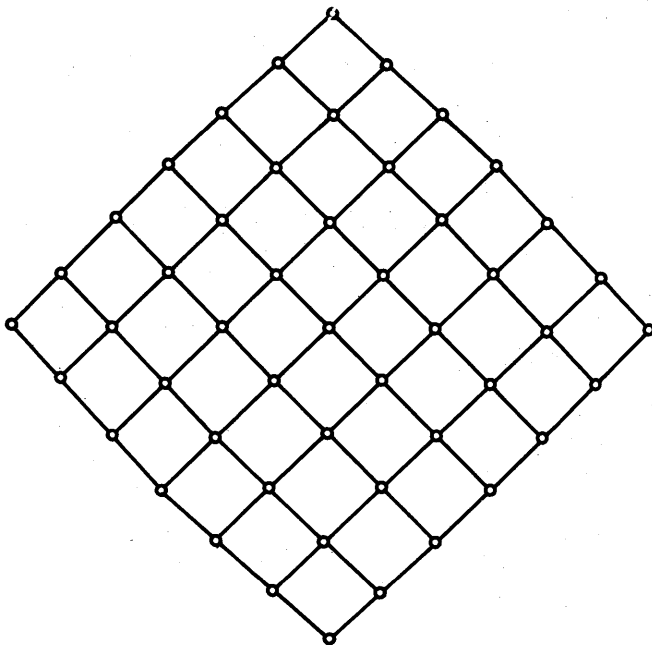


FIGURE 3(b)—Two dimensions.

where $i, j = 1, 2, \dots, m$ are the points defining hyper-object,

d_{ij} = distance from point i to point j in original n -dimensional space,

D_{ij} = distance from point i to point j in new two-dimensional space,

S = normalizing function to ensure K does not depend on scale of D_{ij} ,

W_{ij} = weighting function to ensure neighboring points (i.e., shortest distances) have most effect on K .

It will be seen that K has an absolute minimum when $\frac{d_{ij}}{D_{ij}}$ is constant for each pair of points. In general this will not be possible if the dimensionality of the new space is less than that of the original, and a constrained minimum will result. The procedure starts with an arbitrary set of m points in two-dimensions and uses gradient methods to iteratively move each point to minimize K . The weighting function W_{ij} is chosen to decrease with interpoint distance so that neighboring points tend to maintain their interrelationships in the new two-dimensional space. Satisfactory results have been obtained with $W_{ij} = \frac{1}{D_{ij}}, \frac{1}{D_{ij}^2}$, and $\frac{1}{D_{ij}^3}$

With $W_{ij} = \frac{1}{D_{ij}^2}$ the expression for K becomes

$$K = \left[\sum_{ij} \frac{d_{ij}^2}{D_{ij}^2} \right]^2$$

The algorithm to minimize this function can be efficiently implemented, and has been used as a subroutine with an interactive graphic display system. The dimensionality of the original space has little effect on the computation since the d_{ij} 's are only calculated once. The efficiency is mainly determined by the number of points m since there are $m(m-1)/2$ interpoint distances.

An application to pattern recognition

Patterns are generally described by n -dimensional feature vectors.² It is a serious problem to choose a good measurement system and to extract from the measurements an efficient set of features so that patterns belonging to different categories are easily classified. Although mathematical techniques are available to test whether classes of patterns are separable on a given set of features, the techniques give little intuition as to

which features should be changed if categories cannot be separated. Thus any graphical display system which gives some indication of interrelationships between pattern classes can be useful in designing feature extraction systems.

In principle the approach described above could be applied directly. Suppose there are p_1 examples of class 1, p_2 of class 2, and so on up to p_r of class r , then the total number of feature vectors is $m = \sum_{i=1}^r p_i$. Un-

fortunately this is typically quite large (80 in an example below), and the procedure becomes very inefficient. A more efficient approach is to represent each of the r pattern classes by one or more representative feature vectors. This can simply be the mean feature

vector for each class, $z_i = \sum_{j=1}^{p_i} x_{ij}$ where x_{ij} is the j th example feature vector of class i . If the classes are multimodal, this can be detected and each mode represented by its mean. Then the class representatives $z_i, i = 1, 2, \dots, r$ are projected into two dimensions as described above.

A two-dimensional display of class means is interesting in itself, but it is also useful to display all examples of each class. A transformation from the original n -dimensional space to two dimensions can be obtained by applying linear regression to the class means. Let

$Z^{(n \times r)} = [z_1, z_2, \dots, z_r]$ represent the original class means and let $Y^{(2 \times r)} = [y_1, y_2, \dots, y_r]$ represent the new class means in two-dimensional space. Then the well-known linear regression approach gives $Y = TZ$ where $T = Y(ZZ')^{-1}Z'$ if ZZ' is nonsingular or more generally $T = YZ^+$ if Z^+ is the generalized inverse of Z .⁶

This approach is a compromise between the "pseudo" transformation which "unfolds" the hyperobject described by class means and a linear transformation which linearly projects members of each class about their means. That satisfactory results can be obtained is shown by the examples below.

This technique was applied to data obtained from hand drawn characters. There were 8 examples of each of the 10 digits from 0-9. These were drawn on a 4×5 grid and 20-dimensional data resulted. Thus $r = 10$, $n = 20$, $p_1 = p_2 = \dots = p_{10} = 8$ and $m = 80$. The results of the pseudo projection are illustrated in Figure 4. If this diagram is studied it will be seen that some classes were linearly separable and all others could be linearly separated with only one error each. Thus the original features were quite useful, but could be refined by changing the measurement scheme and observing the new display to see if the classes were more separable. An orthogonal projection to minimize the mean square error (as described above) resulted in a display with several classes completely overlapping each other.

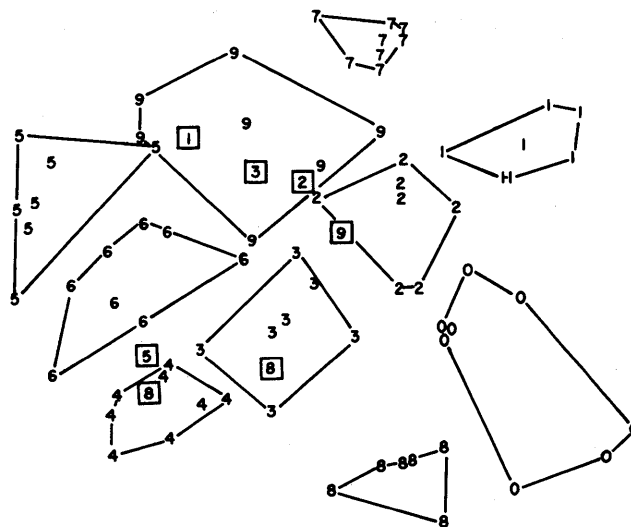


FIGURE 4—Pseudo projection of hand-drawn character data. There are ten categories (digits 0, 1, . . . , 9) and eight examples of each. The original data were 20 dimensional. Class boundaries are added for clarity and points falling outside the boundaries are surrounded by a square.

Another example involves data from what might be described as a speaker recognition experiment. (In fact, this was not the motivation, and the data resulted from research into synthetic spelled speech used in a reading aid for the blind.) The data consisted of 4 examples of each of the 5 subjects making the same sound at the same pitch. The features describing each speaker were the amplitudes of the first 24 harmonics of the fundamental frequency. Thus $r = 5$, $n = 24$, $p_1 = p_2 = \dots = p_5 = 4$ and $m = 20$. The results are illustrated in Figure 5. It can be seen that the speakers are linearly separable on these features and in addition some judgment can be made as to the differences and similarities between the speaker's voices.

DISCUSSION

The examples described here show that it is often possible to obtain useful two-dimensional representations of multidimensional data. For research on pattern recognition the procedure described has been implemented as a subroutine in ALGOL-20. It is used in connection with the Philco graphic display "SCOPES" connected on a time-shared basis to Carnegie-Mellon University's CDC G-21 digital computer. It has already been found that with this "pseudo" projection technique it is possible by a man-machine interaction to

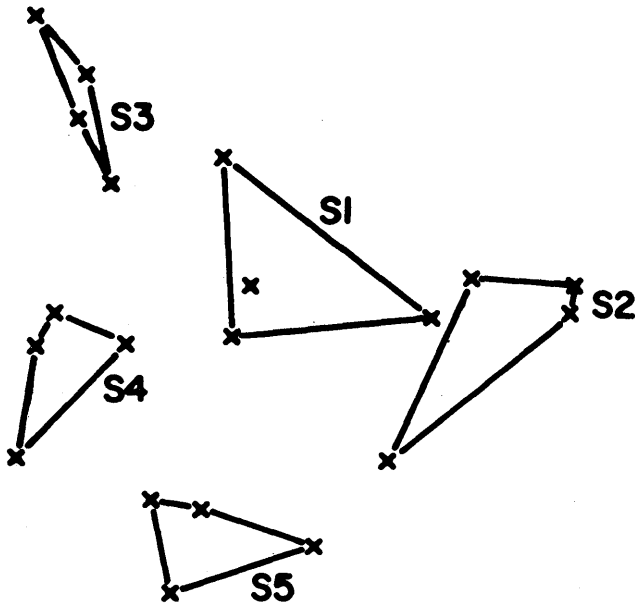


FIGURE 5—Pseudo projection of speaker recognition data. There are five subjects (S1, . . . , S5) and four examples of each. The original data were 24 dimensional. Class boundaries are added for clarity.

efficiently investigate a wide set of features for a number of pattern recognition problems. It appears that the

approach should enable a number of other problems involving multidimensional data to be solved more simply by use of a computer and an interactive visual display.

ACKNOWLEDGMENTS

The author wishes to thank W. D. Strecker for providing the data used in the speaker recognition example.

REFERENCES

- 1 M D PRINCE
Man-computer graphics for computer-aided design
Proc IEEE vol 54 pp 1698-1708 Dec 1966
- 2 F E HOHN
Elementary matrix algebra
Macmillian Co New York 1958
- 3 A M NOLL
A computer technique for displaying n-dimensional hyperobjects
Comm ACM vol 10 pp 469-473 August 1967
- 4 R N SHEPARD J D CARROLL
Parametric representation of nonlinear data structures
Proceedings of International Symposium on Multivariate Analysis ed by P R Krishnaiah Academic Press New York 1966
- 5 N J NILSSON
Learning machines
McGraw-Hill New York 1965
- 6 L A ZADEH C A DESOER
Linear systems theory—The state space approach
McGraw-Hill New York 1963

The on-line firing squad simulator*

by R. M. BALZER and R. W. SHIREY

The RAND Corporation
Santa Monica, California

INTRODUCTION

The purpose of this study is to investigate man/machine interaction in the context of solving a conceptually difficult, formal problem. We want a problem that requires no specialized knowledge, so that a fair comparison can be made between computer-aided and unaided attempts at solution. We also want a problem that is graphic. The firing squad synchronization problem satisfies these criteria extremely well. It has the added advantage that no optimal solution has yet been produced.

The system designed for these purposes is essentially a collection of problem-solving aids that can be divided into three main groups: the first includes bookkeeping aids, useful displays of information, ability to get hard copy, and other basic services; the second, means for testing and simulating solutions; the third, specialized, high level heuristic aids for creating solutions. All three groups attempt to extend the user's power in exploring the universe of the problem, enabling and encouraging him to approach the problem in ways that might otherwise be prohibited by immense amounts of necessary hand calculations or the human tendency toward error.

We hope that this system will result in interesting new solutions to the firing squad problem, and will provide new information on the reactions of humans in such man/machine interactive environments.

We begin by stating the problem and noting some of its inherent difficulties. Next, we discuss the necessary tasks for solving the problem, and then go on to show how and why some of these tasks should be automated. Then, finally, we make general recommendations concerning the design of similar computer systems, based on the experience gained while constructing this one.

*This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15 67 C 0142. Any views or conclusions contained in this Memorandum should not be interpreted as representing the official opinion or policy of ARPA.

Problem statement

This Memorandum concerns a problem publicly first presented in 1964 by E. F. Moore¹:

The problem known as the firing squad synchronization problem was devised about the year 1957 by John Myhill, but so far as I know the statement of the problem has not yet appeared in print. It has been widely circulated by word of mouth, and has attracted sufficient interest that it ought to be available in print. The problem first arose in connection with causing all parts of a self-reproducing machine to be turned on simultaneously. The problem was first solved by John McCarthy and Marvin Minsky, and now that it is known to have a solution, even persons with no background in logical design or computer programming can usually find a solution in a time of two to four hours. The problem has an unusual elegance in that it is directly analogous to problems of logical design, systems design, or programming, but it does not depend on the properties of any particular set of logical elements or the instructions of any particular computer. I would urge those who know a solution to this problem to avoid divulging it to those who are figuring it out for themselves, since this will spoil the fun of this intriguing problem.

Consider a finite (but arbitrarily long) one dimensional array of finite-state machines, all of which are alike except the ones at each end. The machines are called soldiers, and one of the end machines is called a general. The machines are synchronous, and the state of each machine at time $t + 1$ depends on the states of itself and of its two neighbors at time t . The problem is to specify the states and transitions of the soldiers in such a way that the general can cause them to go into one particular terminal state (i.e., they fire their guns) all at exactly the same time. At the beginning (i.e.,

$t = 0$) all the soldiers are assumed to be in a single state, the quiescent state. When the general undergoes the transition into the state labeled "fire when ready," he does not take any initiative afterwards, and the rest is up to the soldiers. The signal can propagate down the line no faster than one soldier per unit of time, and their problem is how to get all coordinated and in rhythm. The tricky part of the problem is that the same kind of soldier with a fixed number K of states is required to be able to do this, regardless of the length n of the firing squad. In particular, the soldier with K states should work correctly, even when n is much larger than K . Roughly speaking, none of the soldiers is permitted to count as high as n .

Two of the soldiers, the General and the soldier farthest from the General, are allowed to be slightly different from the other soldiers in being able to act without having soldiers on both sides of them, but their structure must also be independent of n .

A convenient way of indicating a solution of this problem is to use a piece of graph paper, with the horizontal coordinate representing the spatial position and the vertical coordinate representing time. Within the (i, j) square of the graph paper a symbol may be written, indicating the state of the i th soldier at time j . Visual examination of the pattern of propagation of these symbols can indicate what kinds of signaling must take place between the soldiers.

Any solution to the firing squad synchronization problem can easily be shown to require that the time from the General's order until the guns go off must be at least $2n-2$, where n is the number of soldiers. Most persons solve this problem in a way which requires between $3n$ and $8n$ units of time, although occasionally other solutions are found. Some such other solutions require $5/2n$ and of the order of n -squared units of time, for instance. Until recently, it was not known what the smallest possible time for a solution was. However, this was solved at M.I.T. by Professor E. Goto of the University of Tokyo. The solution obtained by Goto used a very ingenious construction, with each soldier having many thousands of states, and the solution required exactly $2n-2$ units of time. In view of the difficulty of obtaining this solution, a much more interesting problem for beginners is to try to obtain some solution between $3n$ and $8n$ units of time, which as remarked above, is relatively easy to do.*

Goto's solution² apparently has not been published. However, Abraham Waksman³ has found a 16-state minimal-time solution using essentially the same ideas presented in the next section. P. C. Fischer⁴ has also used these ideas in discussing other properties of one-dimensional iterative arrays of finite-state machines. The best solution to date is R. M. Balzer's⁵ 8-state minimal-time solution.

Common approaches and basic considerations

The firing squad synchronization problem can be solved by successively subdividing the line into any number of equal parts, then subdividing each of these parts similarly, and so on, until all the members of the line become division points, at which time they all fire. Most existing solutions use this technique, and it can provide solutions of minimal time, $2n-2$. Balzer's solution⁵ divides the line into halves, quarters, eighths, etc.

Finding a solution entails construction of a finite-state machine by defining for the machine a transition function that yields appropriate behavior when placed in the iterative array. Although automata are usually defined by state tables, here it is easier to interpret a function as a set of rules called productions. These rules take the form

$$\text{LMR} \rightarrow \text{S.}$$

This rule states that if, at time t , a machine is in state M , and the machine on its left is in state L , and the machine on its right is in state R , then the machine's state at time $t + 1$ is S . We call S the "resultant" of the production.

In particular, we are concerned only with minimal-time solutions. To treat the problem resulting from the soldiers at each end of the line, we use an additional state as an end marker, and, at each end of the line, a virtual additional machine which forever remains in the marker state. Since no other machine is ever in the marker state, a single set of productions can be defined for all machines in the array.

Exhaustive search for the function is out of the question, even with the help of a computer, because the number of possible state tables is far too large. For example, if we seek a solution with ten states (plus the end marker), there will be $9^3 + 2 \cdot 9^2 - 2 = 889$ productions. (The problem statement excludes certain productions and fixes the resultant of two others.) Each of the 889 productions can assume ten values, for a total of 10^{889} functions.

Solution by hand

While building a function, say with ten states, the

*Ref. 1, pp. 213-214.

experimenter faces a number of separate tasks—some routine, some challenging, many time-consuming and tedious. He obviously must maintain a large production table. Given some table, perhaps only partially completed, he will need to test it on firing squads of different lengths. This simply involves retrieving values from the table and copying them onto graph paper. Both tasks are routine; nevertheless, performing them will consume much of the experimenter's time.

After several attempts, he may discover that some productions are more important than others, that they are keys to the solution, and he might wish to mark these in order to remind himself that their values should not be altered without special consideration.

The challenging tasks are the creative ones, and the foremost of these is the creation of ingenious approaches to the problem. These schemes usually appear as a two-dimensional plan for propagation of signals along the squad through time. One method for *simultaneously* implementing and testing an approach is to draw on the graph paper a skeleton diagram of the intended function behavior, and then force the productions to conform to this plan. This method of defining productions eliminates many false steps.

Special cases arise when the squad is quite short, say less than fifteen men. After a large portion of the production set is defined, especially key productions, and the function has been tested on longer squads, exhaustive search may become feasible for filling in the special productions required for these cases.

If an error occurs in a simulation, such as a soldier firing too early or too late, or if contradictions arise while attempting to fit productions to a behavior skeleton, some production must be changed. The experimenter then becomes interested in why he originally made this definition. Therefore he finds it useful to keep a history of production usage, particularly a table of first usages in the simulation he is currently considering.

In all these tasks there is a high probability of human error due to the large size of the tables, the large number of separate acts to be performed, and, of course, the repetitious nature of most of the work.

Solving with computer aids

The mechanically repetitious nature of some tasks naturally leads to thoughts of automating them—providing computer aids for the experimenter. The obvious candidates for automation are those tasks which primarily consist of information storage and retrieval, such as table maintenance and simulation. Exhaustive search, where feasible, is handled best by a computer. Having provided these basic services, other more sophisticated tools become possible as well. Finally, the

graphic nature of both the problem and the methods previously described influences the choice of computing hardware; graphic input and output quickly come to mind.

The use of interactive graphic equipment is implied because the reactions of humans to a computing system are highly important. A rapid interaction between man and machine tends to stimulate the intuition and perceptivity of the experimenter; immediate response from the machine maintains a high level of human cerebral activity. Just as not having computer aids at all, using them off-line would slow the response from a second or less to hours. Progress might become so slow that the user would lose interest in the problem.

Summary remarks on the problem

Let us summarize the above discussion—of the problem and the comparison between attempts at solution with and without computer aids—in order to draw some conclusions about the value of this study.

The problem is interesting enough to have attracted wide attention, but difficult enough that no optimal solution has been demonstrated. It requires no special background, and is simple enough that at least an inefficient solution can be found by hand in a few hours. Conversely, it is rich enough to suggest a computer implementation of a number of tools and techniques to aid the investigator. Also, it is naturally oriented toward the use of interactive graphic hardware.

Furthermore, since exhaustive search for a solution is not practical, the computer aids are only tools, and the user still must provide the creative insights and approaches necessary to finding a solution. Thus, the firing squad synchronization problem is a particularly suitable vehicle for evaluating the effectiveness of interactive, graphical problem-solving aids by comparing their effects with the results of unaided efforts.

The system in general

The Firing Squad Synchronization, Simulation and Solution System (FS5) is a highly interactive, graphical computer system. It furnishes three basic groups of tools: the first includes bookkeeping for tables; the second deals with simulation and testing; a third contains the more sophisticated tools, including the ability to draw and implement a skeleton plan, request exhaustive searches, and other functions not obviously needed, but included on the basis of experience with the problem. Associated with these three main categories is a corona of minor devices (e.g., for obtaining hard copy of displays).

Hardware, software and interaction

The FS5 program is written in IBM system/360 PL/I language and runs on an IBM System/360 Model 40. A user communicates with the computer via a RAND Tablet⁶ in conjunction with an IBM 2250 cathode ray tube (CRT) display. The tablet hardware consists of a horizontal 10.24-inch-square writing surface and a pen-like writing instrument, together having a resolution of 100 lines per inch along both Cartesian coordinates.

As the user moves the stylus near the tablet surface, a (hardware generated) dot on the CRT follows the stylus motion; this direct feedback helps the user to position the stylus for pointing or drawing. When he presses the stylus against the tablet writing surface, a switch in the stylus closes, notifying the computer that the user is beginning a stroke. As he moves the stylus across the tablet, the stylus track is displayed (via software) on the CRT; the stylus thus seems to have "ink." When the stylus is lifted, its switch is opened notifying the computer of a stroke completion, and "inking" ceases. A user may "point" at an area on the CRT by closing and opening the stylus switch on the corresponding area of the tablet surface.

The FS5 program uses a set of graphics subroutines written at RAND and called the Integrated Graphics System (IGS). Both character and geometric pattern recognition are included in IGS⁷. A character written by the user is replaced on the display by the corresponding machine-generated character.

The FS5 system presents the user with a picture of a control panel (Figure 1). The controls are used as if they were physical buttons; they are "pushed" by touching them with the stylus. Problem information is displayed in three main areas. On the left, FS5 shows the simulations of firing squads from length one to length 25. On the right, there is a scroll display of production-usage history. At the top center, FS5 offers a variety of messages concerning its own use and status. The use and function of the controls are described in the following sections.

Number of states and external state names

Suppose an experimenter wishes to search for a 10-state solution. He begins by writing "10" in the space provided:

```
# OF STATES    ,10
```

For mnemonic purposes, he will find it convenient to have the states represented by alphabetic characters or other symbols. For example, he might use acronyms: "Q" for the quiescent state; "G" for the general; "F"

for the firing state. Thus, after the number of states is selected, FS5 displays an initial alphabetic choice for the state names:

```
STATES    A B C D E F G H I J
```

At any time, the experimenter may write over this display to replace these choices by his own.

The message center

If we remove the burden of tedious work only to replace it with a large set of system rules and procedures to be learned, the experimenter has gained very little. To avoid this pitfall, FS5 has a MESSAGE CENTER which prompts the user on system usage, informs him of conditions, and suggests actions to take when errors occur. In other words, FS5 supplies copious run-time diagnostics.

For example, when the experimenter begins to write in a value for the number of states, FS5 prompts him with

```
ENTER NUMBER OF STATES
SWEEP TO EXIT.
```

If he begins to rewrite the external state names, he sees

```
ENTER NAMES OF STATES
1 = QUIESCENT    2 = GENERAL
LAST = FIRING
SWEEP PEN TO EXIT
```

Furthermore, FS5 guards against such illegal procedures as trying to enter one of the three reserve state names—"#", ".", "?"—in this case by refusing to accept them.

The policy on a user error is to announce it, correct it and leave the system in a usable condition whenever possible, or else inhibit further action until the user makes a correction, and advise him how to do so.

Entry, storage and retrieval of function values

For a 10-state solution, as many as 891 function values might be needed. As a complication, a large number of productions might be undefined at any given time. FS5 provides several ways to enter, retrieve and alter productions, and takes appropriate action when an undefined production is referenced.

To illustrate, the experimenter may enter a production by writing

```
DEFINE FROZEN    QQG→G
```

If he later wishes to recall this value, he writes

DEFINE FROZEN QQG→?

and FS5 replaces the “?” by the value; here “G”, or by “.” if the production is undefined.

Alternatively, the system might have been designed to display the entire state table upon request. However, at any one moment the experimenter is usually interested in only one production. Moreover, many table entries might never be of interest, because no simulation needs them.

Simple simulation of a firing squad

After defining several productions, the experimenter will want to test the function on firing squads of various lengths; FS5 offers several modes of simulation and testing. For a simple case, suppose that a simulation is desired for length 4. The user enters the “4” with the stylus:

LENGTH 4

FS5 responds by initializing the firing squad:

0	Q	Q	Q	G
1
2
3
4
5
6	F	F	F	F

In addition, the system always provides two productions:

#QQ→Q and QQQ→Q

where “#” represents the end-marker state. These are the two productions required by the problem statement.

Let us further suppose that the user has entered

QG#→G, QQG₁→G, QGQ→Q, and #GQ→G.

He starts the simulation by touching

START SQUAD

Then the message center will display

FIRING IN PROGRESS.

Simulation proceeds from time 1 down, left to right on

successive rows. Because the QGG production is undefined, simulation will cease at time 2.

0	Q	Q	Q	G	Undefined.
1	Q	Q	G	G	
2	Q	G	.	.	
3	
4	
5	
6	F	F	F	F	

The message center will contain

ERROR: FUNC NULL & SQUAD FREE,

and the undefined production will be displayed,

DEFINE FROZEN QGG→

ready for the experimenter to enter a value.

A simulation may be temporarily halted at any time to check its progress. During these manual stops, FS5 continues to advise on system status; and messages are also provided for automatic stops.

Entering constraints

With these simple services at his disposal, the experimenter can turn his attention to finding good solution approaches. FS5 enables him to enter two-dimensional skeleton plans which really are a set of constraints on the function behavior.

The state at time zero and the constraints at the firing time are fixed by the problem statement and provided by the system. To enter other constraints, first the user touches

DEFINE CONSTRAINT

after which FS5 replies with instructions. Next, the user touches two points to define a line segment on the simulation display, and then a name in the STATE display.

0	Q	Q	Q	G	Constraint to be entered.
1	.	.	G	.	
2	.	G	.	.	
3	G	.	.	.	
4	
5	
6	F	F	F	F	

In other words, a constraint is a line segment of states which is “drawn” on the display.

Any number of constraints may be entered, and one may be drawn over another. The "."'s in the display are intended as guides in determining straight lines, and FS5 automatically provides other temporary guides and markers. If an error is made or a change desired, the last constraint entered may be erased:

REMOVE LAST

The ability to enter constraints becomes a powerful tool when used in conjunction with the simulation modes described in the following two sections.

Simulation with constraints

If the experimenter starts a simulation for length 4 with all productions undefined except for $\#QQ \rightarrow Q$ and $QQQ \rightarrow Q$, and with the three-position constraint of the previous example, then FS5 will define the production $QQG \rightarrow G$ from the constraint. However, the simulation will terminate as shown below because neither is $QG\#$ defined nor is the simulation constrained at the position where $QG\#$ is first required.

0	Q	Q	Q	G	Defined by constraint
1	Q	Q	G	.	Undefined
2	.	G	.	.	
3	G	.	.	.	
4	
5	
6	F	F	F	F	

The ability to draw large numbers of complicated constraints thus relieves the experimenter of the task of tailoring many individual productions to produce the same behavior; all the necessary definitions are made by the system.

The system also detects contradictions between constraints and previously defined functions. Such an error would have occurred had the resultant of QQG been set to Q . These contradictions often escape notice when simulations are performed by hand.

As an alternative to drawing constraints, a language to describe them might be devised. However, it is hard to imagine a language as easy or as natural to use as the FS5 method.

Simulation with backtracking

As mentioned above, exhaustive search for a function might become feasible when relatively few productions remain undefined. The use of constraints also can make exhaustive search feasible, because these constraints act as implicit definitions. To take advantage of constraints FS5 was equipped with a widely used method of effi-

cient search called the "backtrack" technique.⁸⁻¹⁰ For readers not familiar with backtracking, or who may know it by another name, a brief review is in order.

Many combinatorial problems can be stated in the form, "Find a vector (s_1, s_2, \dots, s_m) which satisfies p_m ," where s_1, s_2, \dots, s_m are to be chosen from a finite set of N distinct objects, and p_m is some property. The "brute force" approach is to form in turn each of the N^m possible vectors, testing whether or not it satisfies p_m . A backtrack algorithm is designed to yield the same result with far fewer trails.

The backtrack method consists of defining properties p_k for $1 \leq k \leq m$ in such a way that whenever (s_1, s_2, \dots, s_m) satisfies p_m , then (s_1, \dots, s_k) necessarily satisfies p_k . The computer is programmed to consider only those partial solutions (s_1, \dots, s_k) which satisfy p_k ; if p_k is not satisfied, then the N^{m-k} vectors $(s_1, \dots, s_k, s_k + 1, \dots, s_m)$ are not examined by the program. When all choices for s_k are exhausted, the program backtracks to make a new choice for s_{k-1} . If the properties p_k can be chosen in an efficient way, comparatively few cases are considered.

In the firing squad problem, the vector (s_1, s_2, \dots, s_m) consists of production definitions. The backtrack method applied in FS5 serially defines the productions as they are needed in the simulation of a firing squad of fixed length n .

The method begins with all productions undefined except the two required by the problem statement. After initializing the firing squad for length n , the program begins to find the new state of each position in the simulation according to the productions which are already defined. If a production is encountered which is not already defined, and this occurs at an unconstrained position, then the resultant is set to either the firing state or another state, depending on whether or not this occurs at firing time. If the position is constrained, the resultant is set to the constraint value.

The process of serial definition continues until an error occurs. An error is defined to be either a soldier going into the firing state before firing time, a soldier not firing at firing time, of a conflict between a constraint and a production already defined. When an error occurs, FS5 backtracks to find the most recently defined production whose resultant is not the firing state, which is first used where there is no constraint, and for which all the choices of a resultant have not been exhausted. All productions defined after this are now undefined, and this production is set equal to a value which has not yet been tried for it. The program then returns to the position in the firing squad simulation where this production was first defined, and simulation continues from there.

The above process of finding the new state of a soldier

and defining productions as needed is continued until either a solution is found for length n or else no productions remain which are alterable. In the latter case, we have tried all possibilities which could lead to a solution for the given length with the given constraints and a given number of states. Thus there is no solution in this form.

The experimenter can request FS5 to simulate in "AUTO" mode, in which case backtracking will be applied to any undefined productions which are needed. Backtrack mode may be used with or without either constraints or explicit production definitions having been entered. Simulation will only cease if either a successful function is found or all possibilities are exhausted.

Frozen and free productions

The experimenter can "freeze" the value of a production if he wishes to prevent its alteration without his explicit consent; the key productions are of this nature. Frozen productions are not altered by any simulation mode. Hence, a frozen production is another form of constraint and, if used, may further reduce backtracking effort. Other productions are termed "free" because the backtracking mechanism is free to alter them.

Snap view and bright positions

While in backtracking mode, it is useful and necessary to view the progress of the simulation. Sometimes the experimenter can notice an area where much backtracking occurs, and enter explicit frozen productions or additional constraints to eliminate such bottlenecks. Furthermore, if the constraints are neither numerous nor strong, the number of search possibilities could still be astronomical. In this case, if the experimenter periodically views the progress of the simulation, he can decide when it should be aborted.

With the "SNAP VIEW" option "ON," redisplay of the simulation occurs after each row is completed and also whenever the system must backtrack. Otherwise (and in all cases of "STOP" mode), redisplay occurs only when simulation terminates. Since a position in a simulation at which a production is first used is of special interest, all such positions may be brightened by pushing a button:

BRIGHT

Both features are optional because frequent redisplay significantly increases running time.

History scroll, freezing and deletion

Although the experimenter may never be interested

in seeing the entire production table at one time, he may have occasion to view significant portions of it. A scroll display gives him the list of productions used in the current terminated simulation, in order of original usage, and indicates which are frozen.

If production definitions were generated by constraints or backtracking, he might want to freeze some or discard others. Either can be done by pushing the appropriate button

FREEZE VALUE

DELETE VALUE

and touching productions on the scroll.

Image solutions

Experience with the problem, and general consideration of the form that any solution must take, led to giving FS5 another heuristic tool, which requires explanation because its motivation is less obvious than that of other program features.

In any solution, signals must travel the entire length of a squad in both directions because the general, before he can fire, must know that the order to fire has reached the last soldier on the opposite end of the squad. If the signal sent by the general is 1, and the signal returned by the last soldier is 2, then we may think of signal 2 as being the image produced by the reflection of signal 1 from the end of the squad. In other words, the general bounces signal 1 off the end of the squad; the image—echo—returns to him as signal 2.

Experience with various solution methods has demonstrated many other instances in which the image analogy is helpful. For example, suppose that we are applying the techniques of successive subdivision, and have contrived a partial skeleton plan:

0				G		
1			1	G		
2			1	G		
3			1	3	G	
4			1	3	G	
5		1		3	G	
6	2			3	G	
7		2		3	G	
8			2	3	G	
9				G	G	
10			1	G	4	G
11		1		G	4	G
.			
.					.	
.					.	

The general emits signal 1, and it travels to the left at the maximum possible rate of one man per unit of time. This signal arrives at the end of the squad and produces an image, signal 2, which travels at the same rate in the opposite direction.

The general also emits signal 3, and it travels at one-third the rate of signal 1. Thus, signals 2 and 3 meet at the midpoint of the squad and produce the first division point. This central soldier is then promoted to general, and the process can be repeated for each of the two halves.

To repeat the process, the central general sends signal 1 to the left as before, but now a signal 4 is also sent to the right. Signal 4 is intended to behave in the same manner as 1, except that 4 travels in the opposite direction. Signal 4 is, therefore, an image of signal 1, created by reflection about the center of the squad.

Images imply that certain symmetries will probably exist between sets of productions and between pairs of states. Therefore, an additional heuristic for the problem is to look for solutions having the property that for every production $LMR \rightarrow S$, there exists a production.

Image (R) Image (M) Image (L) \rightarrow Image (S)

where the Image function maps the set of states onto itself such that $\text{Image}(\text{Image}(S)) = S$ for all states S .

In FS5, if the image-solution mode is selected and the user defines a proper image mapping, then whenever a production is defined, the image production is also defined. The image method may be used separately or in combination with constraints and backtracking. Obviously, the image method also improves the feasibility of exhaustive search because the number of free productions is again reduced.

Miscellaneous

Other controls allow for reinitializations, for simulation testing over any range of lengths up to 500 men, and for hard copy of displays and tables.

Implementation experience and prerequisites

Any system like FS5 endeavors to provide the researcher with tools and response time that encourage and allow him to apply methods of solution which might otherwise be impractical. On the other hand, if the labor of writing the software is greater than the hand calculation it eliminates, a researcher finds small encouragement. In general, if the cost of building an interactive system exceeds the importance of the problem area, the system will not be built. Our feeling is that the cost of FS5 is reasonable, and that costs relative to more important problems will be significantly lower.

The required hardware includes a digital computer, a CRT display with appropriate graphic input device, and associated interface equipment. The choice of input device is crucial to human reaction. A light pen is at best a clumsy pointing instrument, and a typewriter keyboard with display cursor is an unnatural tool. Had these been the only devices available, many FS5 features would have been neither conceived nor implemented. An appliance used in the manner of a pencil, such as the RAND Tablet, is central to the efficacy of interactive problem-solving systems.

FS5 required three software types, exclusive of programming language and operating system: a graphic software system (IGS); routines to service displays and controls; and routines providing non-graphical aids. IGS allows the user to think globally about displays for his problem, rather than about intricate hardware and bit patterns. Routines to generate and manage displays consist primarily of calls to IGS. Non-graphical routines, such as table maintenance and backtracking, were no different than they would have been if all output was printed.

Thus the major efforts in writing FS5 were to design displays and to interface with the existing graphic software. With such high-level languages as PL/I or FORTRAN, and a good package such as IGS, this is not a very difficult task.

CONCLUSION

An on-line, graphical, man/machine interactive computer system can provide greatly increased research power over a system lacking these attributes. This is true even when a problem is not inherently graphical. Anyone who is planning a computer system to investigate a difficult problem area should consider extending the design to make it graphical and interactive. Since most medium and large computer facilities already have the necessary hardware and basic software, and since construction of routines to generate and to manage displays is quite simple, the added cost should be very small compared to the extra utility gained.

REFERENCE

- 1 E F MOORE
Sequential machines selected papers
Addison-Wesley 1964
- 2 E GOTO
A minimum time solution of the firing squad problem
Dittoed Course Notes for Applied Mathematics 298 Harvard University May 1962 pp 52-59
- 3 A WAKSMAN
An optimal solution to the firing squad synchronization problem
Information and Control Vol 9 No 1 February 1966 pp 66-78
- 4 P C FISCHER

Generation of primes by a one-dimensional real-time iterative array
 Journal of the Association for Computing Machinery
 Vol 12 No 3 July 1965 pp 388-394

5 R M BALZER
Studies concerning minimal time solutions to the firing squad synchronization problem
 ARPA SD-146 (PhD Thesis) Center for the Study of Information Processing Carnegie Institute of Technology, Pittsburgh Pennsylvania 1966

6 M R DAVIS T O ELLIS
The RAND table A man-machine graphical communication device
 AFIPS Conference Proceedings 1964 FJCC Vol 26 Part I Spartan Books Inc Baltimore Maryland 1964 pp 325-331 also

The RAND Corporation RM-4122-ARPA August 1964

7 G F GRONER
Real-time recognition of hand-printed text
 The RAND Corporation RM-5016-ARPA October 1966

8 R J WALKER
An enumerative technique for a class of combinatorial problems
 AMS Proc Symp Appl Math 10 1960 pp 91-94

9 S W GOLOMB and L D BAUMERT
Backtrack programming
 Journal of the Association for Computing Machinery
 Vol 12 No 4 October 1960 pp 516-524

10 M HALL JR D E KNUTH
Combinatorial Analysis and computers
 The American Mathematical Monthly Vol 72 No 2 Part II February 1965 pp 21-28

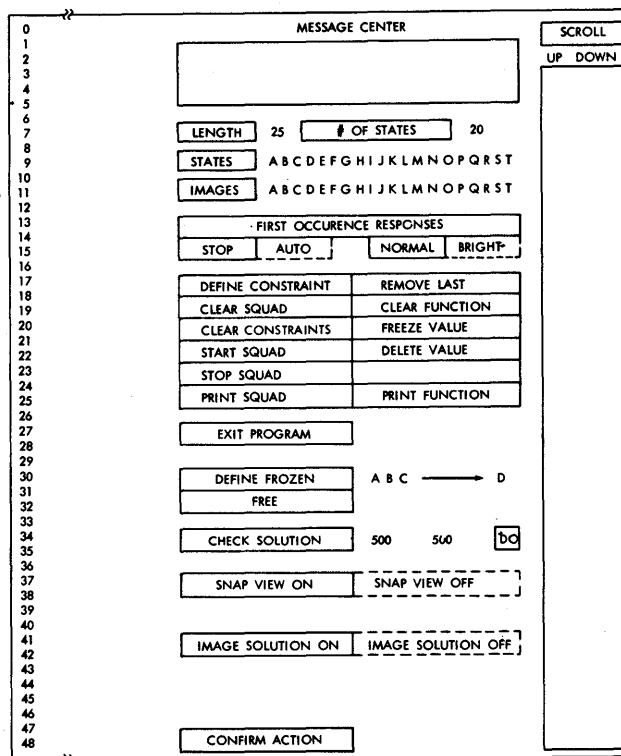


FIGURE 1

Interactive telecommunications access by computer to design characteristics of the nation's nuclear power stations*

by D. W. CARDWELL

Oak Ridge National Laboratory
Oak Ridge, Tennessee

INTRODUCTION

Computer-aided information storage and retrieval systems have been pointed to, frequently, as a means for efficient handling of large masses of data so that users of such systems can rapidly find selected specific items with a minimum degree of effort.¹ In engineering and scientific fields (except for certain rather specialized areas), major progress toward such an objective has been limited to bibliographic sorting of technical publications by keyword search of authors, titles, or abstract context.² The development of a comprehensive system to provide varied users with access to factual technical data, on a broad basis, has awaited a need sufficiently important to warrant the substantial effort required for such an undertaking.

The remarkable sudden surge, since 1965, in electric utilities "going nuclear" has placed a back-breaking burden upon the U. S. Atomic Energy Commission's Division of Reactor Licensing to fulfill their responsibilities in reviewing engineering design proposals for each nuclear power plant to evaluate the adequacy of safety provisions. Figure 1 (a) and (b) show on maps of the United States locations of the many large nuclear power plants now committed for construction as contrasted to the few units that were committed three years ago. In January 1967, the Reactor Division of the Oak Ridge National Laboratory, aided by Union Carbide Nuclear Company Computing Technology Center, was com-

*Research sponsored by the U.S. Atomic Energy Commission under contract with Union Carbide Corporation.

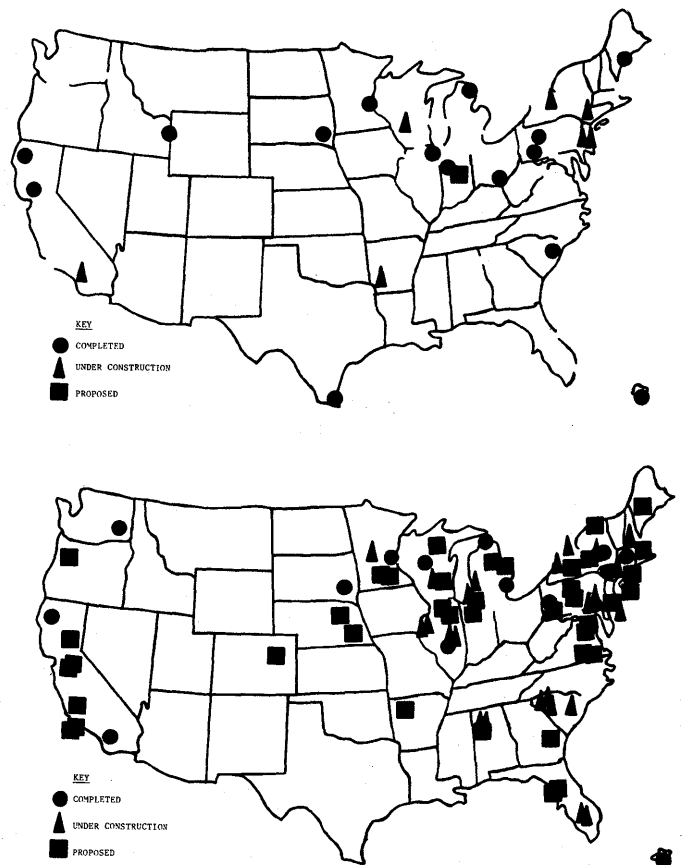


FIGURE 1

mitted to develop a computerized system capable of responding to selective search requests with readout of factual data to a family of telecommunications terminals used by reactor engi-

neering specialists who are engaged in nuclear power plant design evaluations.

This project was given the acronym label of CHORD-S to represent "Computer Handling of Reactor Data—Safety." The CHORD-S telecommunications system became initially operational during 1968, processing information in certain technical categories of greatest current interest to the Atomic Energy Commission. Expanding the data bank volume, improving man-machine dialogue capabilities, and adding to the number and versatility of terminals are a continuing operation, each of these activities directed toward increasing the real value of the system to its potential family of users.

The CHORD-S information system has been designed to include the following unique combination of features:

1. Factual technical data organized in ten level hierarchical structure for input to the Data Bank.
2. Computer input formatted on magnetic typewriter/converter, to provide
 - a. Off-line localized verification by originators.
 - b. Reductions in turnaround time, cost, and frequency of errors as compared to conventional punched input.
3. Multipurpose central computer storage capacity of several hundred million characters, retrievable by rapid random access.
4. Direct access via telephone line from family of remote terminals, in time-sharing mode, at distances up to several hundred miles from computer center.
5. CRT display included in terminal capabilities, in addition to conventional telecommunications typewriter.
6. On-line dialogue (conversational mode), employing almost exclusively common English language words, queries, and responses.
7. "Lead-in" program allowing user option of being guided rapidly by the computer to selected areas of interest without advance knowledge of file structure or reference to code books.
8. "Compare" program wherein computer automatically contrasts values of large blocks of design parameters for several power plants to provide "Readout by Exception," only where differences are significant, withholding undesired alphanumeric data that has no significant value to user's query.
9. Open-ended file maintenance provisions for

frequent efficient up-dating, additions or deletions.

10. Optional tutorial program wherein computer gives user step-by-step online instructions for operation in any of the available modes.

11. Access from terminals to key-worded bibliographic nuclear safety reference files and computational programs that already reside in the central computer. These and other residual resources of the computer serve to enhance the primary capabilities of CHORD-S.

The unprecedented combination of user oriented features provided by the CHORD-S system, should be applicable to many technical areas (in addition to nuclear power plants), where large masses of factual data must be efficiently searched to yield specific responses to queries posed by professional personnel who are engaged in sequential progressive activities wherein real-time output is essential.

System design philosophy

Initial development of design philosophy for CHORD-S was materially aided by use of hardware plus operating experience available from an existing computerized (key word bibliographic) information system that has successfully functioned within the overall ORNL Nuclear Safety Program since 1963. That operating system is the Nuclear Safety Information Center (NSIC).⁴ By such direct observations and intensive study of current reports on other automated information storage and retrieval systems,^{5,10} plus knowledge of the specific needs of potential users, we directed the development of the CHORD-S system within the following primary guidelines:

1. Queries originated by users must be as free as practicable from regimented "computereeze" language and format, with heavy dependence upon natural language expressions, limited in number.
2. Potential users cannot afford delays occasioned by manual look up in extensive code books.
3. For the inexperienced user, computer programmed "lead-in" techniques must be readily called forth in response to simple commands, to automatically reveal the contents of file structure and provide rapid guidance to areas of individual interest.
4. For experienced users, optional short-cuts,

directed to selected bodies of data, must be available to avoid unsolicited, time consuming path-finding.

5. Responses to queries at telecommunications terminals must be compatible in speed to normal human sensory perceptions to minimize breaking "trains of thought."

6. Data drawn from source material must be expressed and organized for file entry in fashions appropriate to conventions prevailing in each technical category, and be carefully screened to include only items of greatest significance to potential users.

7. File structure in the computer memory must be flexible and open ended to easily accommodate frequent additions, revisions, and updating, to keep pace with changes in the nuclear power industry.

8. Output options to telecommunications consoles should include a capability for rough rapid scan to reveal highlights of data file, to then be followed by progressively deeper selective probes for ultimate retrieval of specific data desired by individual users.

9. Terminal readout should allow rapid visual display of information of only transitory value, accompanied by user control of commands for producing selective hard copy of printout.

10. Priority should initially be given to obtaining alphanumeric output. The extent of capability for graphical, diagrammatic, or pictorial type information to be provided must be determined by evaluation of cost of such features against their relative worth to users.

11. System hardware and software must be designed to handle a progressively increasing number of terminals to the network and have flexibility for innovations of an unpredictable nature determined from early experience to be of practical benefit to users.

12. Selection from the many options available for long distance communications transmission must be based on a utility/economy optimization of several factors such as speed, capacity, reliability, and adaptability to newly developed I/O devices.

13. Compatibility features for future interface connection with other information systems being developed for the U. S. Atomic Energy Commission must be provided whenever logical and practicable.

It was obvious to us that fulfillment of so demanding a set of criteria required a merging of

the talents of several specialized disciplines: (1) computer technologists (software and hardware), (2) CHORD-S project nuclear information engineers, (3) communications engineers, and (4) nuclear reactor power plant safety specialists from potential user organizations. Although it is always difficult to combine talents of personnel of contrasting technical backgrounds and interests, we believe that we have gradually fulfilled most of that mission. It is our strong belief that the road to success for a complex computerized information system must be paved by an established willingness on the part of computer technologists to understand and accommodate genuine needs of potential system users; and those same potential users must exert significant effort to obtain at least a surface working knowledge of the capabilities and limitations of computer related hardware and software. Also, where telecommunications is to be employed, practical up-to-date technical knowledge of current advances in the field of communications engineering is an essential ingredient. Rather than seeking rare so-called "generalists" to bridge gaps between the established disciplines, we have depended upon strong overall leadership guidance, with some forcing to the extent necessary for accomplishing essential merging of specialties.¹¹

The man-machine interactive concept

Information storage and retrieval systems, until recently, have been limited mainly to batch processing of queries by the computer, requiring that the user wait an appreciable length of time between successive sets of data readout. The advent of third-generation digital computers, with greatly increased capacity and versatility of hardware and software, have made it feasible for users to communicate directly on-line with a central computer in essentially continuous conversational dialogue.¹² CHORD-S provides such capability for reactor specialists who are engaged in assessing the design of existing or proposed nuclear power stations.

For automated on-line information systems to serve responsible busy professional people, such systems must possess dominant operational features that match the natural habits of these people.¹³ We look upon dialogue between user and the system in a closed loop cybernetic sense. Although the human being in the loop may, for varying justifiable reasons, be relatively slow in formulating and entering queries at his remote

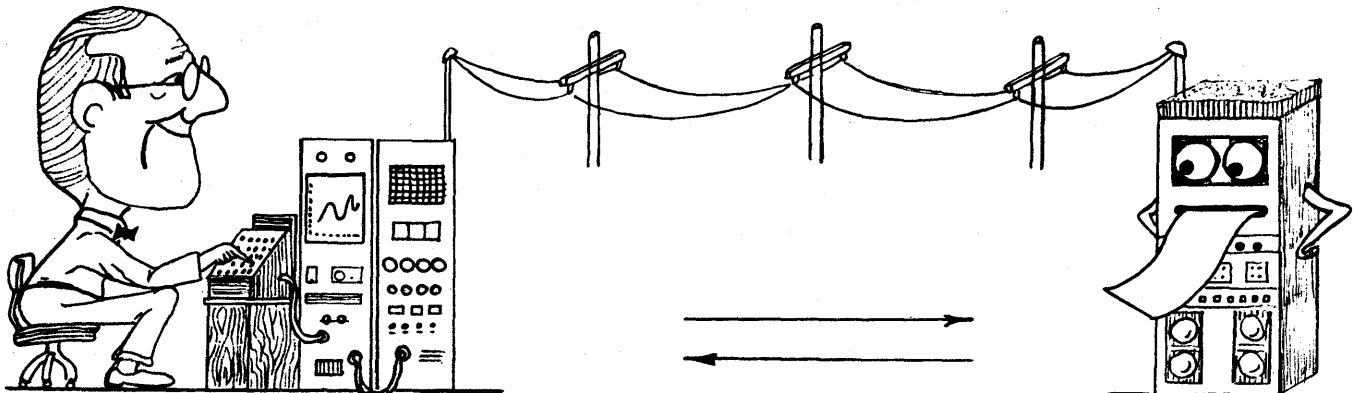
terminal, he has a right to expect intelligent, efficient responses provided in a form and at a speed most conducive to his understanding.¹⁴ Figure 2 presents an elementary diagram to quantitatively emphasize some of the fundamental traits of man in contrast with the capabilities of digital computers as they have evolved from the first to the current third-generation machines. The most striking difference apparent in this diagram is that of speed; as each new generation of machines has made its appearance, the magnitude of this disparity has become greater. Commercial time-sharing computer systems whereby 20 or more consoles converse, essentially simultaneously, with a single central computer have capitalized on this difference. Central processing units (CPU) of modern computers operate at near electron speeds (microseconds and nanoseconds), whereas actions by human beings are limited to tenths of seconds.

In systems such as CHORD-S, users located many miles from the computer center depend upon local console equipment with built-in speed limitations as shown, connected to computer ports by some form of public communications link. Conventional voice-grade telephone lines are logically employed for such service because of their ready availability, dependability, and relative economy.¹⁵ As noted in the diagram, data messages are transmitted via ordinary telephone lines at rates of 120 to 400 char/sec. Such conventional rates of transmission fix an additional time limiting parameter on the overall system.

In the CHORD-S interactive system where dialogue flows back and forth rapidly between a user and the computer, each link of the loop needs to be optimized in design to provide time responses appropriate to the key human perceptive senses. As may be seen in Figure 2, man's voice is the most efficient medium for formulating queries at a

FIGURE 2

<u>MAN</u>	<u>TRANSMISSION SYSTEM</u>	<u>MACHINE</u> (Third Generation Digital Computer)
<p>A. His Communication Capabilities</p> <ol style="list-style-type: none"> 1. Sensing Senses <ol style="list-style-type: none"> a. Speech: 5-20 char/sec b. Handwriting: 3-10 char/sec c. Typing: 1-5 char/sec 2. Receiving Senses <ol style="list-style-type: none"> a. Sight <ul style="list-style-type: none"> -Pictorial: Very fast -Reading: 10-15 char/sec b. Hearing: Fast <p>B. His Memory Capabilities</p> <ol style="list-style-type: none"> a. Total Storage: Up to $(10)^{15}$ char b. Prompt Recall: 1-10% of total c. Speed of Recall: <ul style="list-style-type: none"> -Up to 10% fast; remainder slow (and inaccurate) <p>C. His Psychological Reactions</p> <ol style="list-style-type: none"> a. Rational - Likely predictable b. Emotional - Unlikely predictable 	<p>A. Terminal I/O Equipment</p> <ol style="list-style-type: none"> 1. Telecom Typewriter <ul style="list-style-type: none"> Printing Speed: 10-15 char/sec Char/line: 72-156 2. Cathode Ray Tube (CRT) <ul style="list-style-type: none"> Display Speeds: 240-100,000 char/sec Total no. of char: 1,000-4,000 3. Remote Batch Printers <ul style="list-style-type: none"> Printing Speed: 600-660 char/sec <p>B. Telephone Network</p> <ol style="list-style-type: none"> 1. Voice Grade - Single Channel <ul style="list-style-type: none"> Speed: 120-400 char/sec 2. Broad Bank - Multi-Channel <ul style="list-style-type: none"> Speed: Up to 23,000 char/sec 	<p>A. Central Processing Unit</p> <ol style="list-style-type: none"> 1. Execution rate 10^6-10^8 2. Memory cycle time: <ul style="list-style-type: none"> Microsec to nanosec 3. Core memory capacity: <ul style="list-style-type: none"> $10(10)^3$-$10(10)^6$ <p>B. Peripheral Storage</p> <ol style="list-style-type: none"> 1. Access Speed: Up to $300(10)^3$ char/sec 2. Capacity: <ul style="list-style-type: none"> Up to $1.6(10)^9$ char



terminal. Although rapid advances are currently being made in voice communication with computers, we have concluded that, except for limited vocabularies, such an approach is not yet ready for broad exploitation. This leaves only man's sense of touch for initiating communications, which usually is applied by operation of a typewriter keyboard, an inefficient avenue for self-expression by most technical people. Handwriting is usually more rapid than typing, and for some circumstances that medium ("Rand tablet," etc.) is used for computer input, although it may not now be practical for application to a broad data base system, such as CHORD-S. The most encouraging development for speeding up manual direction of computer operation involves the use of a light pen, or cursor, to select from computer generated cathode ray tube (CRT) displays items for further exploration.¹⁶ Also, CRT terminals, when of the vector type, can display on screens complex graphical output, a form of communications very effective in appealing to man's faculty for quick pattern recognition.¹⁷ In any event, it has been clear that terminals for the CHORD-S interactive system should require minimum amounts of input from the user to eliminate excessive tedious efforts by amateur typists.

It has been observed¹⁸ that for psychological, as well as for practical reasons, technical personnel seeking on-line responses from the data bank become impatient and dissatisfied whenever the time between the end of their query and the beginning of response extends beyond a few seconds. For the system to be acceptable, readout at the terminal should approach the rate for natural efficient comprehension by the user. Comprehension, at present, depends almost solely upon the human's exercise of vision accompanied by his mental reaction, which, depending upon the complexity of the information received, is fairly rapid. Consequently the speed, format, size, etc., of visual display provided at the terminal is closely related to user acceptability, and hence the success of the system.

In addition to man-machine considerations given to terminal hardware characteristics, the ease of entering CHORD-S queries and obtaining useful rapid responses from a computer is highly dependent upon the degree of naturalness of terminal language employed for information exchange.¹⁹ The development of a terminal language for CHORD-S has included involved original ideas

as well as ideas found to have been successful in other IS&R systems. Opportunity to use the CTC IBM 360/50 general purpose computer for initial operation of the CHORD-S system has presented advantages in early availability of equipment with basic telecommunications and operational programming. However, in order to pursue the theme of optimum man-machine response throughout the entire loop, considerable ingenuity has been necessary in the development of special CHORD-S programming to assure compatibility with fixed features of the computer center.

In addition to accommodating form and speed of dialogue considerations, we have exerted considerable effort to make certain that users are provided with built-in programming options that always allow them full control over any decision making processes that they wish to exercise. This minimizes any chances of the automated system "jumping to unwarranted conclusions."

Functions of the information network

A general layout of the CHORD-S system as an information storage and retrieval network is shown in Figure 3. The following steps are followed by CHORD-S project personnel to establish flow of information between sources and users of the system:

1. Technical data are primarily drawn from documentary material that has been prepared by nuclear reactor power plant designers and operators for license application submitted to the U. S. Atomic Energy Commission. Auxiliary informa-

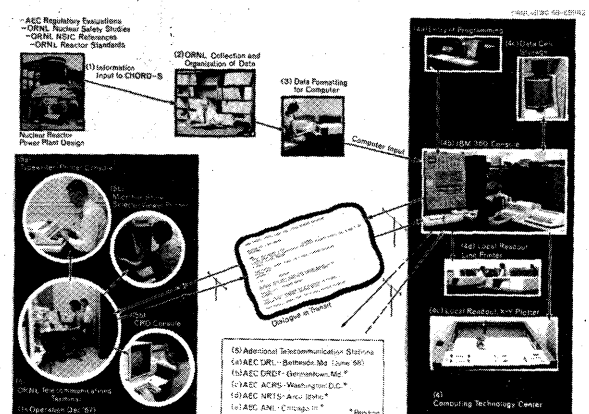


FIGURE 3

tion of pertinence comes from other sources such as those shown on the diagram.

2. Reactor specialists originate standardized power plant system characteristics listings of greatest significance to nuclear safety with specific data for each plant organized systematically for efficient computer handling. The multi-hierarchical structure of the data file demands that careful attention be given to recognizing logical technical relationships between major headings and successive sublevels as items of information progress from the general to the more specific. Examples of data retrieval, to be given later, will show how this type of file structure when intelligently organized for input, offers users an opportunity to obtain readouts that collectively encompass spans of subject matter individually self-sufficient in the evaluation of specified areas of engineering design.

3. Technical and clerical personnel, with the aid of automated magnetic tape typewriter equipment (IBM-MT/ST with Digidata Converter),²⁰ structure reactor data in standardized formats, producing at the point of origin, magnetic tapes suitable for direct entry into the computer, eliminating keypunching. Where prime responsibility for originating complete and accurate file additions resides with individual technical specialists, there is considerable advantage in applying this newly developed data entry technique because it provides centralized capability for rapid and efficient off-line verification.

4. Data on reels of seven-track magnetic tape are entered and stored in the memory of the IBM 360/50 computer at the Oak Ridge Computing Technology Center (CTR)²¹ by means of special programming developed by CTC specialists. The basic Data Bank is built as an over-night batch operation, on-line input from remote terminals being unnecessary. Output of data at the computer center can be obtained whenever desired as a batch processing operation from several conventional types of readout devices as shown in Figure 3. Such batch output may be a partial file dump for checking the accuracy of updating operations, or it may be a selected set of engineering data considered too lengthy for efficient remote terminal readout. Ports providing interactive access to the CHORD-S Data Bank from remotely situated terminals are indicated in the diagram by symbolic telephone lines emanating from the computer center.

5. A representative operating telecommunica-

tions terminal is shown at the lower left of Figure 3 time sharing with other terminals. Meaningful conversational exchanges via telephone line between terminals and the central computer employ query and response techniques developed jointly by CTC programming personnel and ORNL reactor engineers. As illustrated in Figure 3, a telecommunications typewriter transmits queries, and prints at a rate of 15 char/sec output data selected for preservation in hard copy. An alphanumeric CRT console provides rapid (up to 200 char/sec) visual display of transitory output information. (Rapid responses displayed by the CRT hasten progressive dialogue up to the point of obtaining ultimate search objectives.) A vector-type CRT may be employed to provide graphic displays for the system users to gain the advantage of information in pictorial form.²² Diagrammatic illustrations and voluminous material, not readily adaptable to computer storage, are referenced in the data bank to guide automated retrieval from local terminal auxiliary files which are mechanized by microfilm storage-readout devices.

Entry of data into computer

Figure 4 shows how CHORD-S data has been entered in the central computer and how the special programming has been handled.²³ Note how new or revised technical data are merged with basic CHORD-S update and input programs to produce revised master files for data cell storage and terminal access. The computer input program makes a diagnostic scan of the data to check for errors. Obvious errors, such as improper formats, incorrect field type and length, and the absence of flags and delimiters are detected by the present

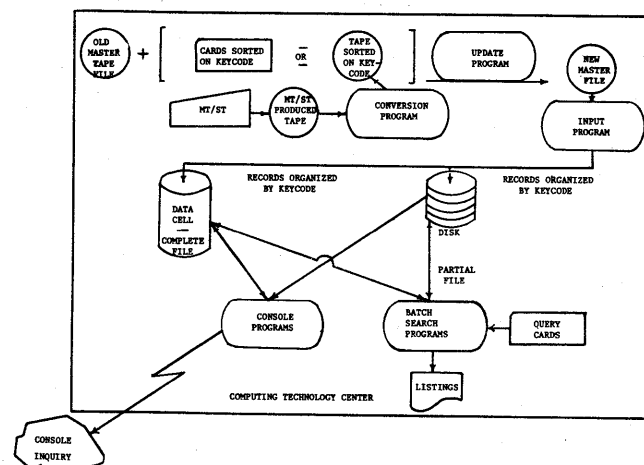


FIGURE 4

program. Another class of errors, known as logical errors, is much more difficult to detect. These errors can only be found by providing the computer with a greater knowledge of the ranges that variables may assume and the possible conflicting attributes of data items. That being impractical, these types of errors are found and corrected by the nuclear engineers who originated the computer input information by systematic review of batch dump output. (It is important to note that checking of input data for accuracy throughout is the responsibility of the originator. Any erroneous information in this Data Bank could have an effect of delaying the construction of one or more \$100,000,000 nuclear power stations.)

After being screened for errors, the input data are sorted by the computer so that they are in the same order as that on the master file. The file maintenance program matches the updated information against the master file, and records are added, deleted, or replaced as required. The updated file is now written onto a disk or data cell, and an index file, consisting of keycodes with pointers to records in the master file, is created on a disk to permit direct access.

Writing the master file and index file on direct access devices permits the remote terminal and batch query programs to retrieve the data in a nonsequential manner. Later additional index files can be created and used to index the master file on the basis of other parameters.

File organization for such large volumes of data is a critical problem especially when using direct access devices to support remote consoles. The problem is one of minimizing the external storage requirements and at the same time providing efficient computer use and fast terminal response. This problem has been solved, in part, by the local implementation of several computer software routines which provide for variable length record blocking and track overflow capabilities.

Modes of access

Several modes of access to the CHORD-S data file have been developed and placed into operation to accommodate various needs of users. One output program responds to a direct command of "\$ Display" from the user to read out factual data selected by subject matter key codes. In another mode of retrieval ("\$ Compare"), the computer scans selected lists of data from various reactors for major differences and reports these as "read-out by exception" thus selecting only that data

relevant to the user's interest. Here, a variable element in the query is set to fit the degree of difference desired by the user. In the third mode of operation ("\$Lead-in"), the computer assists an inexperienced user in finding information he needs, by automatically guiding him into the data file structure without requiring familiarity with the organization of information. Options are available for each mode of access whereby slight changes in user queries will either restrict or broaden the degree of detail of readout. For all of these retrieval modes, the user sits at a remote console and participates in an interactive dialogue with the computer. As his search progresses, he calls on whichever retrieval mode that satisfies his immediate needs. A built-in tutorial program advises the terminal user of the nature of any error in query structure. If desired, the "tutorial program" can be called to read out step-by-step instructions to the terminal for any program available.

Examples of dialogue via remote terminal

In the following search sequence, we assume the user has not had previous experience with the CHORD-S console; he, therefore, first uses the "lead-in" program which instructs him in the use of the console and guides him into the data bank. (On subsequent uses, he may remember from his earlier experience the way information is organized except for details and could enter the search sequence at a more advanced stage.) Since there is no need to preserve hard copy of initial lead-in dialogue, the user elects to save time by employing the CRT terminal as illustrated in Figure 5. By simply typing "A.\$lead-in." there is flashed upon the screen major "Summary Section" subject matter headings of the file with corresponding key codes. Assuming that the engineer, in this evaluation, wishes to investigate design features of the core of certain nuclear reactors, he continues his guided search by typing in or moving the CRT cursor to AC. The next computer response will be a list of subheadings in that area of the data bank. By successive continuation of this rapid pathfinding query—response technique the more detailed subject matter file structure is revealed. Each file entry has a unique key code label which can be used in requesting readout of detailed information on individual characteristics of selected nuclear power plants. An example of CRT readout is given in Figure 6 where the "\$compare" program was employed to retrieve from the computer



FIGURE 5

summary data on reactor core fuel characteristics (key codes ACBB to ACBC) of two plants designated by name abbreviations. From evaluation of the data shown, the user then called on the "Sdisplay" query to simply find the value of one related parameter of interest to him, "Maximum Fuel Thermal Output" (key code ACACH).

Up to this point, the user elected to employ CRT display, so that turnaround on the dialogue proceeded about as rapidly as his natural senses normally function. If he should decide that he needs to retain hard copy of CRT display, he signals the typewriter to automatically copy from the local terminal controller memory. Where time required for mechanical machine copying would handicap the user's next deliberations, this can be avoided by use of a speedy photo-optical device (if such recently developed equipment is available at the terminal).

As users of the system become familiar with their most frequently used code designations, they can address their queries directly to individual codes or to code ranges, for immediate readout of descriptive data, skipping "\$leadin."

The "\$compare" query is a development original with this project, that places tremendous power in the hands of terminal users to contrast at various levels of detail, design characteristics of one or more nuclear power stations. This important feature functions as follows: Nuclear safety information engineers, when the originally establish their standard characteristics listings, assign for storage in the computer memory a "Delta Factor." This represents their profes-

sional opinion of parametric deviations of significance, such as $\pm 10\%$. When a user wishes to make a rapid scan of large volumes of data to find contrasting design features of different power plants, he uses "\$compare" with a controllable modifier "m." Different degrees of course or fine comparison are obtained by the terminal user designating large or small values of "m." Efficient "readout by exception" is then accomplished by means of computer programming which causes the system to ignore data values (where numerical) that contrast by a lesser difference than that requested. By successively varying the value of "m" from large to fractional values for different sections of the file, the user can rapidly "close in" on items wherein the degree of difference corresponds to his specific search interest. The "\$compare" feature of the CHORD-S information system could be given broad application to other information systems.

In the following example, employing "variable depth compare" with typewriter readout, the engineer is exploring design data on three nuclear plants in the area of "Heat Transfer at Initial Full Power (IFP)"

```

acac read: c1 11 s1,m=3
Compare.
KEYCODE CHARACTERISTIC AND UNITS C-YMK 100-PT-2 SURRY-1
      SFT 01 SFT 01 SFT 01
ACAC HEAT TRANS AT IFP
ACACA AVG POWER DEN,KW/L NS 0 92.8
ACACS :((: UWS (L=3) OR CRIT HEAT FLUX RATIO 2.62 1.81 1.86
ACACC AVG PEL TO CLAD GAP THERM CONDUCT,BTU/HR-SOFT-DEGF NS 1,000 1,000
ACACH AVG FUEL THERM OUTPUT,KW/FT NS 0 6.2
    
```

JOB COMPARE. ENDED. TIME: CPU=00042, ELAPSED = 00:10.07.

Explanation of Symbols in Printout
 NS - Data not supplied in source references.
 * - No significant deviation from data shown in first column.

On his first approach, he assigned no value to the multiplier "m" so the computer automatically selected for readout differences based on a delta factor of 1. If we assume that he had been working with a much larger span of subject matter than is practical to illustrate here, he could decide that the volume of readout is too massive for isolating major differences. In that case, he may choose to enter another query specifying that m=3. This effort on his part then would produce the following output at his terminal, which much more clearly highlights the major differences in design that he seeks:

```

acac read: c1 11 s1
Compare.
KEYCODE CHARACTERISTIC AND UNITS C-YMK 100-PT-2 SURRY-1
      SFT 01 SFT 01 SFT 01
ACAC HEAT TRANS AT IFP
ACACA AVG POWER DEN,KW/L NS 0 92.8
ACACS MFR DRG (L=3) OR CRIT HEAT FLUX RATIO 2.62 1.81 1.86
ACACC ACTIVE HEAT TRANS SURF AREA, SQFT NS 1,000 1,000
ACACE AVG HEAT FLUX , BTU/HR-SOFT 35,900 52,200 4
ACACF 'MAX HEAT FLUX,BTU/HR-SOFT 136,400 375,000 191,000
ACACG AVG FUEL THERM OUTPUT,KW/FT 421,500 570,800 538,700
ACACH MAX FUEL THERM OUTPUT,KW/FT NS 0 6.2
ACACI MAX CLAD SURF TEMP,DEGF 13.7 18.5 17.5
      625. 659. 657.
    
```

JOB COMPARE. ENDED. TIME: CPU=00041, ELAPSED = 00:08.73.

The tabular output format of the previous examples is popular with engineers, but has an obvious built-in limitation on the number of columns of data that can be displayed. In order to overcome that restraint, responses from the CHORD-S system that would exceed horizontal space limitations automatically shift to a vertical listing, such as shown in the following example:

```
acac acach. c1 i1 s1 o1
$compare.
```

```
ACAC      HEAT TRANS AT IFP
ACACA     AVG POWER DEN, KW/L
C1        NS
S1        92.8
O1        79.6

ACACB     #IN DNB (W-3) OR CRIT HEAT FLUX RATIO
C1        2.82
I1        1.81
S1        1.86
O1        1.6

ACACC     AVG PEL TO CLAD GAP THERM CONDUCT, BTU/HR-SQFT-DEG F
C1        NS
I1        1,000.
S1        1,000.

ACACD     ACTIVE HEAT TRANS SURF AREA, SQFT
C1        35,900.
I1        52,200.
O1        48,578.

ACACE     AVG HEAT FLUX , BTU/HR-SQFT
C1        136,400.
I1        175,600.
S1        191,000.
O1        167,620.

ACACF     MAX HEAT FLUX, BTU/HR-SQFT
C1        421,500.
I1        570,800.
S1        538,700.
O1        543,000.

ACACG     AVG FUEL THERM OUTPUT, KW/FT
C1        NS
S1        6.2
O1        5.4
```

```
JOB COMPARE. ENDED. TIME: CPU =00485, ELAPSED = 00:09.85.
```

```
C1 Conn. Yankee
I1 Indian Point
S1 Surry 1
O1 Oconee 1
```

The foregoing examples of data retrieval have mainly focused on cases where a user of CHORD-S wishes to make an efficient comparison of selected design features of two or more nuclear power plants. That is because of the uniqueness of this capability, and its great utility to design evaluators. There are other important capabilities of the system that are not included in the examples. A simple query can be placed that will yield com-



FIGURE 6

plete bodies of data covering the design of a single designated plant. Where desired, sets of parametric design values can be obtained representative of different assumed accident conditions. Sources of data can be called for yielding stored bibliographic reference information. Employing a computational program available from the general purpose computer (TERMTRAN),²⁴ a terminal user can perform a wide range of calculations using design data retrieved from CHORD-S.

Future outlook

Development of CHORD-S has produced an operating system that provides conversational mode access from telecommunications terminals to a central computer data bank. Data stored, thus far, is representative of some of the more important factual design characteristics of certain U. S. nuclear power reactor plants. As additional information is added, Atomic Energy Commission personnel will be engaged in evaluating the worth of this IS&R system from actual operating experience. The nature of further development of CHORD-S will be guided by results from such experience.

The particular computer and the initial terminal hardware employed to demonstrate feasibility and long range future potentials of CHORD-S has been based largely upon the matter of ready availability. Commitments for a permanent system require the completion of extensive evaluations of the wide ranges of hardware and software offered by industry to achieve the most desirable operating features within limits of reasonable economy.

As CHORD-S is developed to full potential, the

data bank in addition to increasing substantially in volume, will encompass deeper levels of design detail than can readily be expressed in concise alphanumeric terms. Information structures will need to accommodate more lengthy narrative descriptions, diagrams, graphs, etc. The most efficient methods for presenting such material to terminal users in meaningful forms are under intensive study, evaluating the applicability of sophisticated hardware and software, much of which is just becoming available from commercial vendors.

If the data bank increases in size for more complete inclusion of design characteristics and progressively builds to receive input from larger numbers of nuclear power stations (around 30 currently being committed each year), many hundreds of millions of characters will need to be stored. File structure and search techniques will likely be altered as necessary to assure rapid cross-reference access to individual areas of specialized interest.

Long range plans include a gradual increase in the number of terminals to accommodate various groups of AEC Headquarters personnel in the Washington area. Also, it is intended that the network will be expanded to provide terminals at other locations throughout the United States. Requirements for extensive telecommunications features are being preliminarily evaluated to make certain that the system can be efficiently expanded without basic overall reworking of hardware and software provisions.

In view of the pioneering features of much of the CHORD-S undertaking, care is continually exercised to assure compatibility with other information systems of the government, particularly those of the Atomic Energy Commission. Opportunities for certain interconnections are already obvious and can be expected to increase rapidly in the future.

ACKNOWLEDGMENTS

The initial development of the CHORD-S project reported herein has been accomplished as a part of the Nuclear Safety Program of the Oak Ridge National Laboratory under the direction of W. B. Cottrell, in fulfillment of objectives established by S. E. Levine, Assistant Director of the U. S. Atomic Energy Commission Division of Reactor Licensing.

The author has served as co-director of CHORD-S, being mainly responsible for areas re-

lated to computer applications and project administration. Noteworthy contributions are much in evidence throughout the paper from W. E. Browning, Jr., in his capacity as co-director for nuclear safety data selection and processing. Specialized computer programming for CHORD-S has been originated by S. L. Yount and M. Magnuski under the direction of E. M. Kidd, UCNC Computing Technology Center. A. Goldenson of UCNC Process Analysis provided assistance with query language development. The extensive task of developing technical data for acceptable computer entry has been accomplished by the following ORNL CHORD-S Group staff engineers: F. A. Heddleson, J. O. Kolb, R. E. Lampton, I. K. Namba, and P. Rubel. Mrs. J. D. Joyner, Mrs. B. K. Seivers, and Mrs. J. M. Copeland have materially aided in editing and producing the text and illustrations that make up the manuscript.

REFERENCES

- 1 D PARKHILL
The challenge of the computer utility
Addison Wesley Publication 1966
- 2 D CARDWELL
The impact of recent computer developments on the future of engineering
Address to the Raleigh N C Engineers Club Nov 14 1966
Unpublished
- 3 J HOGERTON
The arrival of nuclear power
Scientific American Vol 218 No 2 Feb 1968
- 4 J BUCHANAN F HUTTON
Analysis and automated handling of technical information at the nuclear safety information center
American Documentation Institute Vol 18 No 4 Oct 1967
- 5 L BERUL
Survey of equipment developments in the information storage and retrieval field
Auerback Corp FID/FIP Conf June 1967
- 6 W SLACK B PECKHAM L VAN CURA W CARR
A computer-based physical examination system
Journal American Medical Association Vol 200 April 1967
- 7 P CASTLEMAN
Information storage and retrieval cycle
Hospital Computer Project Mem Six-D Bolt Baranek and Newman Inc Cambridge Mass
- 8 W HUBBS J McBRIDE A LEVY
The Baylor medical school teleprocessing system
SJCC Vol 32 May 1968
- 9 C LYNCH
A communications revolution
Science and Technology April 1968
- 10 T BOTHWELL
Computer communications systems
Computer Design Dec 1967
- 11 B SPINARD H McDONALD C TUNIS
W SUTHERLAND R WARD J ARCHIBALD
Panel discussion on man-machine interface
SJCC Vol 32 May 1968

-
- 12 D EVANS B PINE D SCOTT J WEINER
Panel discussion on time-sharing status report
SJCC May 1968
- 13 I ASIMOV
The human brain, its capacity and functions
Houghton Mifflin Co Boston Mass 1964
- 14 P ABELSON
The human use of computing machines
Science Vol 153 No 3733 July 15 1966
- 15 H STEADMAN G SUGAR
Some ways of providing communications facilities for time-sharing computing
SJCC Vol 32 May 1968
- 16 P RUBEL D CARDWELL
Telecommunications system access to computers: Basic characteristics of terminal equipment
ORNL-CF-68-6-35
- 17 S BOWMAN R LICHALTER
Graphical data management in a time-shared environment
SJCC Vol 32 May 1968
- 18 S SMITH
Man-computer information transfer
Electronic Information Display Svstems James A Howard
pp 284-301
- 19 A GOLDENSON D CARDWELL
Query methods in information retrieval
ORNL TM-2376 1968
- 20 *Input verification methods vary among inscribers*
Computer World p 5 June 19 1968
- 21 *Computing technology center*
Brochure Union Carbide Nuclear Division Oak Ridge Tennessee
- 22 *Computer industry annual 1967-68*
UNITECH DIV AES Corp subs of Simon & Schuster Inc
New York New York
- 23 M MAGNUSKI
CHORD-S A system of programs for computer handling of Reactor data for safety
Information Systems Department Computing Technology Center Union Carbide Nuclear Division Oak Ridge Tenn Report No K-1756
- 24 G KNIGHT R NEU
Termtran, a programming system for engineering calculations via remote terminals
UCND CTC Oak Ridge Tenn Report No K-1735 Feb 1968

Computer-driven display facilities for an experimental computer-based library*

by DONALD R. HARING

Massachusetts Institute of Technology
Cambridge, Massachusetts

BACKGROUND

Project Intrex (*information transfer experiments*) is a program of research and experiments intended to provide a foundation for the design of future information-transfer systems. The library of the future is conceived as a computer-based communications network, but at this time we do not know enough details about such a network to design it. Lacking are the necessary experimental facts, especially in the area of user's interaction with such a system. To discover these facts, we want to conduct experiments not only in the laboratory, but above all, in the real-life environment of a useful operating library.^{1,2}

The initial efforts of Project Intrex have been concerned with the problems of access—bibliographic access through an augmented library catalog, and access to full text. This paper describes the design of the initial computer-driven display facilities being developed for the Project Intrex experimental computer-based library. To provide further background information, we will first give some details of the augmented library catalog and the text-access system that are being developed.

For a number of reasons, computer-based libraries that service a wide spectrum of users, such as is found in a university, will be faced with operating on two basically different types of data—that which is digitally stored and that which is photographically stored in some micro-film form. The latter will be images of the original full text of the documents contained in the

library whereas the digital data will constitute the augmented catalog of the library from which the library user gleans information about the library data and documents by conducting on-line computer searches.

One of the concerns of Project Intrex is to conduct a series of experiments to determine how the traditional library catalog can be effectively augmented and combined with on-line computer operation to provide users with a more powerful, comprehensive, and useful guide to library resources. Present plans call for augmenting the traditional library catalog in scope, depth, and search means. For example, the augmented catalog will contain entries for reports and individual journal articles as well as the traditional entries for books. Furthermore, in addition to the title and author of each item, such things as the bibliography, an abstract, key words, and key phrases of each item will be included as part of its catalog entry.^{2,3} A user will be able to conduct searches on nearly any combination of the data contained in a catalog entry. Present plans also call for providing alphanumeric communications between user and computer by means of a high-speed, flexible display console.

Another concern of Project Intrex is to conduct a series of experiments in an effort to devise a workable system that will ultimately provide guaranteed, rapid access to the full text of journal articles, books, reports, theses, and other library materials at stations that are remote from the central store. This goal has several implications. Guaranteed accessibility implies that text never leaves its store and is therefore available to users at all times. Availability with minimum delay at remote locations implies transmission of text in electrical signal form, except in special, limited situations where physical transmission (perhaps by pneumatic tubes) might be appropriate. Remote accessibility implies more conven-

*The research reported here was made possible through the support extended the Massachusetts Institute of Technology, Project Intrex, the Electronic Systems Laboratory, Under Contract NSF-C472 from the National Science Foundation and the Advanced Research Projects Agency of the Department of Defense, and under Grant CLR-373 from the Council on Library Resources, Inc.

ient library use, and in addition is a preliminary step toward realization of a network of computer-based libraries coupled together by means of data communication links.

In order to conduct meaningful experiments, a small-scale experimental total computer-based library system containing some 10,000 systematically-chosen documents in materials science and engineering is being designed and constructed at M.I.T. The computer-driven display facilities discussed here are a part of this system. Figure 1 illustrates the general organization of this experimental library. The library operates roughly as follows: The digital data and the photographic images are created from the original documents. The former are placed into the storage of the time-shared computer (an IBM 7094 system) and the latter are placed into a microfiche storage and retrieval unit (a modified Houston Fearless CARD machine). The stored data are then accessed through the augmented-catalog user consoles and the text-access user terminals, respectively. This paper is concerned with the display facilities required by the experimental library, for more details about other aspects of the experimental library see References 2 and 3.

Several considerations have led to the decision to construct the experimental display facilities at M.I.T. as opposed to purchasing a commercial system that almost satisfies the needs of the experiments. First, there is no commercially available system in which the "almost" is close enough to the requirements of the experiments. Second, these experiments are designed to identify the functional characteristics of a system ideally suited for a computer-based augmented catalog and full-text access. During the course of these experiments, the system is expected to experience modifications which can be more easily performed on locally designed and constructed equipment. Third, an objective of Project Intrex is to develop competence in the field of information-transfer engineering at M.I.T.

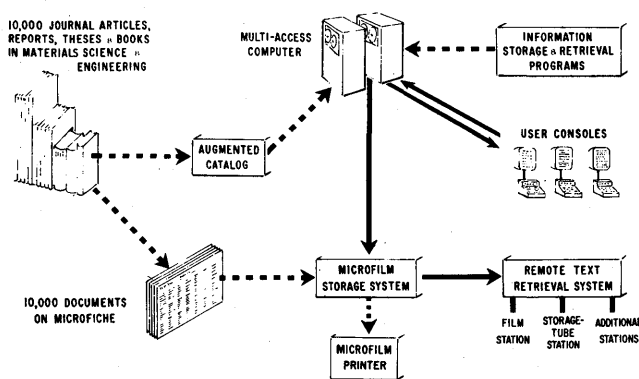


FIGURE 1—Project Intrex experimental library storage and retrieval system

Several points concerning the Intrex program should be emphasized. First, through operational experimentation we wish to obtain information that will be helpful in defining attributes of a large-scale system; that is, our goal is to make full text and its attendant augmented catalog available to a selected well-chosen community of users in order to test user reactions to equipment as well as learn about capabilities and limitations of the technologies used in the equipment. For a discussion of the selection processes used in the choice of the community of users and choice of the material contained in the experimental library to serve them, see References 2 and 3.

Second, in our experimental program, close attention is being given to its scalability. Although the first experimental store of textual material is only a representative fraction of a full-scale library, our purpose is to employ only those techniques and technologies that show promise of being extendable to full library context. Third, the research program envisioned is evolutionary; therefore, the experimental system now being developed is the first of a series that will be needed before the characteristics of a full-scale augmented-catalog and text-access system can be postulated.

The plan of this paper is to first discuss the requirements of the display facilities and then to describe the present Intrex facilities that are being developed. Because of the differences between the storage form and content of the catalog (digital, alphanumeric) and the full text (photographic images, graphic), and because of the differences between user interaction with the catalog and full text, we find it convenient first to describe the two facilities separately, and then to describe the operation of the integrated facilities by means of an example of its use during a library-user's session.

The augmented-catalog display-console requirements

Effective testing of user interaction with the augmented catalog requires a remote computer console optimally suited to the task. Currently available consoles, however, exhibit serious shortcomings as regards catalog experimentation. Impact-printing teletypewriters operating at ten to fifteen characters per second, for example, are clearly too slow for rapid scanning of large quantities of bibliographic data. The cathode-ray-tube (CRT) alphanumeric display terminals now offered by several manufacturers do allow for more rapid, quiet display of computer-stored data. However, they, too, lack features essential to effective user interaction with the augmented catalog. For instance, there is generally a lack of flexibility in operating modes, in formats (e.g., no superscripts and subscripts) and a server limitation on the size of the character set. On the other hand, the CRT graphic display terminals that are

currently available can be programmed to circumvent these deficiencies but are very expensive as regards original cost, communications requirements, and utilization of computer time.

As a result, a study program was initiated to identify the functional characteristics of a console ideally suited for augmented-catalog *experimentation*. Space does not permit a full exposition of the study program. The program involved an investigation of the reactions of various types of users to the terminal facilities serving present time-shared computer utilities, for example, Project MAC,⁷ and some experimental specialized computer-based information networks, for example RECON⁸ and TIP.⁹ The program also involved an investigation of promising relevant new technologies and discussions with many people as regards their thoughts on the input/output requirements of a computer-based library system. The most important items considered in this study program were: (1) the user community that will be served by the augmented-catalog, (2) the type of data that will be contained in the augmented catalog, and (3) the operation of the computer-based library system. The salient point of each of these general considerations are now discussed. The interested reader can find greater detail in References 1-4. Results of the experiments performed on the console system described here will be reported by Project Intrex as they become available.

The user community

The background of the community of users ranges from first year students to researchers on the outer fringes of science, technology, etc. They may be interested in finding such things as the specific heat of carbon steel or the latest theory about semiconductor surfaces. They may use the console occasionally, as in the case of most students, or use the console daily, as in the case of many librarians. It is safe to say that the majority of users will not be computer "buffs" and that they will be demanding of the system. The users will want to be able to operate the system with a minimum amount of re-learning at each session, that is to say, they will want to maintain a useful level of competence as regards the system and have tutoring on specific features of the system operation as needed in a session. Furthermore, the console should not be tiring to use. This effects not only the manner of displaying the data but the layout and organization of the console manual inputs as well.

The augmented-catalog data

The data contained in the augmented catalog are basically alphanumeric and are contained in a special disc file attached to a time-shared computer system. Of

particular consequence to both the display-console design and the entire computer-based library is the large number of alphanumeric symbols that is required. The set of 128 USASCII standard symbols is insufficient, and thus must be augmented. Furthermore, provisions for superscripts, subscripts and underlines must be included to properly display to users in forms with which they are familiar, such items as chemical formulas and mathematical equations.

Relationships among key words and phrases in the catalog as shown in a thesaurus are perhaps most easily displayed as a tree. Thus, as an aid to the user, the display console should have the graphical capability to display a simple tree structure amongst word phrases.

The computer-based library operation

Central to the Intrex computer-based library operation is the concept of an on-line, remote access, interactive time-shared computer system. The advantages and disadvantages of such a computer system are now fairly well known. Of course, the design of the computer-based library should make fullest use of this knowledge.

As opposed to the usual type of short statements interchanged between the users and the computer in the present time-shared computer operations a library user frequently will require a large quantity of data to be sent to him so that he might scan through it. On the other hand, the library user will spend most of his central processor time on generating search specifications rather than generating programs or numerical data.

A library continually receives new acquisitions and library procedures change from time to time. Hence, the augmented-catalog display console must be able to evolve and gracefully accept various types of changes with a minimum of disruption to library services and at a minimum cost.

Since a computer-based library will operate for a long period of time before being replaced, cost of ownership, which not only includes original cost but recurring costs of such items as maintenance, communications and updating, is of great importance.

The augmented-catalog display-console facility

Several basic hypotheses resulted from the study program and have been used in the formulation of the initial design concepts. First, it is advantageous to handle many routine operations at the console in order to minimize communication between the console and the time-shared computer. This approach reduces the demands on the central computer and should result in more rapid access to the central machine when required. It further reduces the cost of transmitting information

from the time-shared computer to the console, an important consideration in any large operational system. Second, careful attention should be given to the size and content of the console alphabet, the ability to produce superscripts and subscripts, and to the human engineering aspects of the console in order to ensure favorable user reaction to the console and to the overall system. Third, it must be possible for the uninitiated user to become familiar with the operation of the console and the catalog system rapidly and easily. Finally, the design of the console should be such that it can be economically reproduced. This feature is a necessary prerequisite to the wide-scale use of computer-based library systems. Consideration of these hypotheses has led to the formulation of the design concepts described in the following paragraphs.

A console has been built that uses a cathode-ray tube (CRT) display with approximately 1,800 alphanumeric-character capacity. The data communication between central computer and console is 120 characters per second with provisions for higher data rates. Several character sets are possible in addition to the English alphabet. User communications are entered by means of a typewriter keyboard, and special function buttons which designate frequently encountered commands. The user's message is displayed on the CRT prior to the transmission to the time-shared computer, and editing of displayed commands is possible. As the user's conversation with the catalog system progresses, certain data

supplied by the computer may be stored locally for future reference, edited as required, and eventually printed in hard-copy form.

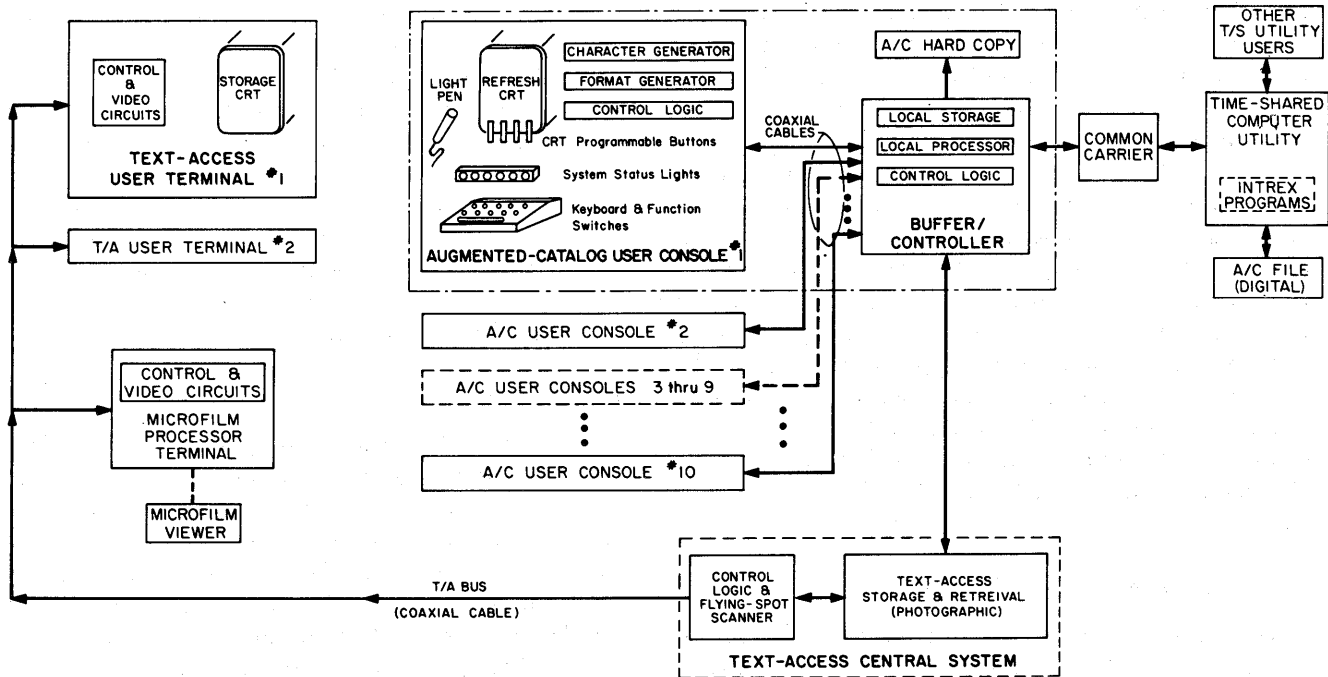
In order to reduce the cost of individual consoles, it is advantageous to cluster consoles around a local station which includes data storage and processing that is common to all clustered consoles. Initial investigations indicate that it should be possible to design economical console systems which cluster about ten consoles at distances of a thousand feet from a local station. Thus, the consoles could be placed in several different rooms of a single building. Interconnections between the consoles and the station are made with coaxial cables, while the connection between the station and the time-shared computer utility are made by common carrier. In the future, a high-speed photographic printer will be located in the vicinity of the console to produce hard copy on command from any of the consoles.

Basic Augmented-Catalog Console System Description

Figure 2 shows the present Intrex display facilities that are being developed. The major components of the augmented-catalog console system are identified by broken lines. Note that individual user console has the following components:

1. A CRT to display information.
2. A set of lights to display information about the

FIGURE 2—An experimental display system for project Intrex



status of the display system and time-shared computer.

3. A set of CRT programmable buttons or function switches that are mechanically operated by the user to select the modes of console operation and the frequently used operations. The labels and functions are determined by the computer program and the labels are displayed on the CRT screen.
4. Mechanical function switches to supplement the CRT programmable buttons.
5. A typewriter keyboard to enter specific information into the system.
6. A light pen to point to information on the CRT screen.
7. A character generator to produce a large alphabet of alphanumeric symbols on the CRT.
8. A format generator to produce the approximately 1800 character spaces on the CRT screen, and provide superscripts and subscripts.
9. Control logic under the direction of the console's CRT programmable buttons, light pen, and function switches and the buffer/controller.

Specific details of these components are discussed in Reference 5. Briefly, the console is organized as follows: All information displayed on the CRT must be stored on a magnetic-drum track located in the buffer/controller (B/C) and pass through the character generator located at the console. A special character generator based on a lensless, flying-spot scanner is being developed for Project Intrex and is thoroughly described in References 2 and 5. It can produce up to 192 high-quality characters. Since the character set is specified by a photographic slide, the set can be easily changed. Each console has its own character generator to provide a flexible system in which each console can operate with a different alphabet. The format generator, which is always connected to the CRT, periodically produces a raster of 31 lines with spaces for 56 characters per line. Human factors studies indicate that the character size resulting from this format is good for the viewing distances at which the console will be operated.^{10,11} The format generator is synchronized to the magnetic drum so that one complete revolution of the drum corresponds to one complete raster. Consequently, drum addresses correspond to specific positions on the CRT screen, thereby simplifying logic. With a refresh rate (controlled by the drum speed) of approximately 57.5 refreshes per second, and 10 bits per character, approximately 1,175,000 bits per second are transferred from the drum to the character generator, hence a coaxial connection between the user console and the B/C is, therefore, required. The time per character is approximately 8.5 microseconds, which is ample time for the

character generator. Since one of the displayed lines of characters must be devoted to the labels for the CRM programmable buttons, approximately 1700 character spaces are available for textual information.

The control logic of the console interprets commands generated by either the console or the processor in the B/C and through these commands the control logic establishes the console's "state" and hence its mode of operation. Through the console commands, the control logic establishes interrupts to the processor which in turn processes these interrupts when its own time is available to treat the console's interrupt. The major items specified by the console state are as follows:

1. The drum track that is being displayed. One track corresponds to one complete display on the CRT and is referred to as a frame.
2. Whether data are being transferred to or from a particular drum track or to or from the time-shared computer.
3. The labels of the CRT programmable buttons, and hence the function of these buttons.
4. Disposition of data generated by the keyboard.

The control logic is itself under the direction of the CRT programmable buttons, the mechanical function switches and light pen, and the processor in the B/C.

The second part of the console system, the B/C, contains four major components. The memory drum, with its associated electronics and logic, stores the display information for the consoles, labels for the CRT programmable buttons, routines for the controller, and instructions on the use of the consoles. The processor, which is actually a small digital computer (a *Varian Data Machines 620 I*), controls the operating modes of the consoles and the data flow among consoles, drum, and central time-shared computer. The input-output buffer and its electronics matches the different data rates in the drum, consoles, and communication link. The buffer also defines code groups corresponding to characters. Direct data communication is provided between the central time-shared computer and B/C.

The low-cost *Vermont Research VR1004S* drum memory being used can provide up to 128 data tracks. Each track represents one complete CRT display, which is called a "frame." One console has five frames dedicated to its exclusive use. Also, a drum track is devoted to the possible labels for the CRT programmable buttons. Another set of drum tracks is devoted to instructions for the use of the console, and still another set to the processor's use. Nondedicated data tracks will be dynamically allocated, giving an individual user a potentially large number of frames viewable instantaneously with no action required by the time-shared computer.

When data phones are used in the communication link, experience with existing display systems indicates that several data phones are required to handle many user consoles from a single B/C. Data phones now available for switched telephone networks provide up to 2000 bits per second of data, but within the near future this rate will be approximately doubled.¹² Since one CRT frame contains approximately 20,000 bits, at least ten seconds is required to transmit a complete single frame. Fortunately, the majority of the messages between the user and the time-shared computer are in the order of one line of text requiring only one-third of a second. However, since a single user might wish to request up to five frames at a time, a maximum of 50 seconds might be required to serve a single user. In order to avoid prolonged delays in service to other users who may be awaiting service at that instant, one 2000-bps data phone must be dedicated to a small number of consoles, or the servicing of the consoles must be interlaced, or combinations of these two must be employed. At the present time, our thinking runs toward use of one data phone for every three consoles, and interlacing the service to provide response time of less than 30 seconds under worst-case conditions. Since, in actual practice, users most frequently will be sending and receiving much less than a complete frame and furthermore, since the probability that several users will be communicating data simultaneously is very small, as regards the communications delays, the service would typically be less than a second.

The processor in the B/C is a small digital computer which operates as the process controller and has interrupt features. A commercially available small computer is being used instead of a specially built computer. This provides a very powerful and flexible console system on which to conduct the Intrex experiments. Although the data rates are high in the CRT channel, by proper organization of the control logic in the user consoles the control data rate, which is the rate important to the processor, can be significantly reduced.

Further details of the augmented-catalog display facility can be found in References 2, 4, and 5.

The text-access display requirements

Much of discussion concerning the augmented-catalog display-console requirements in a previous section are applicable here. The difference between the two sets of requirements are due to the differences of the storage medium and types of user interaction with the stored data. The catalog data are digitally stored in a file that is attached to the time-shared computer and is accessed through and processed by that computer. The library user can conduct computer searches on this data and modify his copy of the data. Thus, if he wants hard-

copy of this data he can have the data organized in any desired form. On the other hand, the full-text data are photographic images that are accessed through a mechanical storage and retrieval unit that is attached to the display facility. These data do not pass through either the time-shared computer or the buffer/controller computer. Their form cannot be modified (except for magnification) and the user cannot conduct computer searches on the data. The only access to full-text data is by call number. Once the data are obtained, the user can read it, can ask for other pages of the same document or ask for a magnification of portions of the page of the document. In short, the dynamics and the dialog between man and computer are considerably different between the two display facilities.

With the original text stored in image form, the problem of remote access of text becomes that of reproducing a high-quality image at a remote point. Our experiments have shown that at least 2000 scan lines, with comparable system resolution, appear to be needed for remote reproduction of microfilmed technical documents (18-to-1 reduction ratio) having average quality and containing the subscripts, superscripts and mathematical symbols that frequently appear in these texts.²

Closed-circuit TV requires high bandwidth (80 MHz) to achieve the desired resolution of 2000 lines per page. Also, because the image is continually refreshed, a separate scanner and transmission line would be required for each simultaneous user. Hence, a facsimile-like system is used in which each text page is scanned and transmitted only once, and the information is captured and stored at the receiver for transient viewing (soft copy) or printing (hard copy). This organization permits a tradeoff between signal bandwidth and transmission time, and also permits time multiplexing of the microfiche on which the image is stored, the microfiche storage and retrieval device, the scanner and the transmission line to serve a number of users. For example, with a transmission time of $\frac{1}{2}$ a second per page, signal bandwidth for a 2000-line scan is reduced to about 4.5 MHz (standard TV channel bandwidth) and a single text-access system can perhaps service 20 to 30 receivers, assuming that users would request new pages at 10 to 15 second intervals.

Let us now describe the present Intrex text-access system being developed in order to further specify the display requirements. The salient features of the text-access system are shown in Fig. 2, and are seen to be an automatic microfiche storage-and-retrieval device (a modified *Houston Fearless* CARD machine) capable of accommodating 750 microfiche (each with a standard 60-page COSATI format) and of being operated under computer control, a single-scan, 2000-line flying-spot scanner for converting microfiche images to video sig-

nals, a 4.5 MHz bandwidth transmission system, two types of receiver stations and a spare station, and the necessary control logic to access documents through the augmented-catalog console. One receiver station provides 35-mm microfilm as its output, a second station produces a visual display of text on a storage tube, and a third station is available for installation of other forms of output equipment as such equipment becomes available.

The document collection for the experimental systems is the full text of the documents in the augmented catalog. Therefore, both catalog and full-text information is available at stations that are remote to the central time-shared computer.

The text-access system is controlled by the augmented catalog buffer/controller unit through its connection to the central-station control unit. Control of the text-access system resides mainly in the buffer/controller (B/C). Thus, for example, if a user requests the transmission of an entire document, the B/C remembers the number of pages to be transmitted and keeps track of the access and transmission operations so as to issue an appropriately timed command for each page. The access number of a document occupies a field in its augmented-catalog entry that is stored in the central time-shared computer. The document number is retrieved automatically whenever a document title is retrieved and thus is available to the B/C for issuance of the text-access command. Our decision to connect the text-access system to the augmented catalog B/C rather than to the central computer is based on our belief that the full-text accessing process could be slowed down by the wait-time of our present time-shared facility.

The text-access display facility

The usefulness of future operational text-access systems will depend to a great extent on the capabilities of the users' terminals. Low cost for the terminals will obviously be of paramount importance. In addition, we are presently facing, in the realm of terminals, severe technological limitations, especially in the area of transient displays (i.e., soft-copy displays in which the user cannot take a copy of the displayed data with him). Substantial research and development activities in the industrial sector should, if successful, contribute markedly to text-access-terminal advancements. In light of these external pending developments we plan, for the present, to employ a minimum number of terminals. We shall have one terminal for each major type of output device, that is, a terminal for transient display of text, another for film copy, and a third for experimenting with new devices as they come along. We also expect to have a separate terminal for making paper copy.

We are favoring the transient-type terminal, since it approaches most closely the capability of providing immediate text access and is potentially the least expensive to operate. Unfortunately, no fully satisfactory device is available for our purpose; however, the *Tektronix* eleven-inch storage tube does afford limited capabilities and it will be incorporated into one terminal. The second class of terminal, the film terminal will also enable us to test the acceptability of film as a primary hard-copy output; that is, we shall be able to decide if film, is an acceptable substitute for the more expensive paper copy.

Storage-tube terminal

The storage-tube terminal, diagrammed in Fig. 3 makes use of the *Tektronix* Type 611, eleven-inch storage display unit. An evaluation of an experimental engineering model of the Tektronix direct-view storage-tube display indicates that the resolution and brightness of this display are adequate for the reader who wishes to make a preliminary examination of text in order to verify its relevance to his requirements.² Resolution may be marginal, however, for perception of poor-quality print or small symbols and characters. (It is estimated that an improvement of approximately 25% in resolution is required). To overcome the resolution limitation, an enlarged version of any one of nine overlapping portions of a page of text can be requested.

Operation of the storage-tube terminal is straightforward. Upon receipt of an ERASE command from the demodulator, any image appearing on the screen is erased. One-half second later the BEGINSWEEP command will be received, followed by the video signal for one page of text. After being written on the screen the text remains on the tube face until the next ERASE command. Because the writing speed of the Tektronix Type 611 display is relatively slow, the scanning of a frame of the original microfilm and the transmission of the corresponding signal must be extended from the desired one-half second writing time to four seconds.

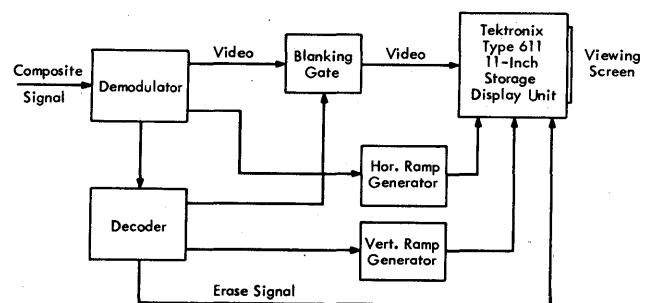


FIGURE 3—Storage-tube terminal

Microfilm-facsimile terminal

The microfilm-facsimile terminal, shown in Figure 4, consist of a high-resolution cathode-ray tube with its associated sweep and focus circuitry, an automatic camera-processor, and control logic required to operate the terminal. On command from the central station the microfilm-facsimile terminal will reconstruct on the face of a high-resolution cathode-ray tube the image of a full page of text from a video signal of 4.5-MHz bandwidth transmitted from the central station. The automatic-camera and film-processor unit will record on 35-mm film the image of the text displayed on the cathode-ray tube and deliver to the user a fully processed strip of film in a convenient form for viewing in a microfilm reader.

The operation and configuration of the high-resolution cathode-ray tube with its associated sweep and focus circuitry is described in the Intrex Semiannual Activity Report dated 15 March 1967² and Reference 6. The operational requirements for the camera and film processor are as follows:

1. The automatic-camera and film-processor unit shall record on 35-mm film the image of a full page of text which is obtained in a single scan and displayed on the screen of a high-resolution cathode-ray tube. It shall also deliver to the user a fully processed strip of film in a convenient form for use in a microfilm viewer.
2. Each strip of film will contain a minimum of one, and a maximum of ten, adjacent images.
3. The maximum combined length of unexposed leader and trailer on each film strip shall be five inches.
4. The film transport of the camera and processor shall handle unperforated 35-mm film.
5. The microfilm-facsimile terminal shall not require an attendant for normal operations and the camera and processor shall not require routing main-

tenance, other than the loading of film and chemicals, more than once per week.

6. The camera-and-processor unit shall be designed for operation by electrical-control signals.
7. In view of the experimental nature of the terminal, the camera-and-processor unit shall be designed with emphasis on flexibility; that is, it shall be possible to change the type of film, the size of the image on the film, the type of chemicals utilized, or the lens, without major equipment alterations.

Since no camera-and-processor unit that satisfactorily meets all the above requirements was found to be commercially available, a *Kodak MCD-II* microfilm camera and *GAF Transflo Type 1206* leaderless film processor were purchased and have been merged into a camera-and-processor unit.²

Display software

Display software operates in the time-shared computer and the small computer contained in the buffer/controller. The time-shared-computer software provides the more sophisticated segments of the man-machine dialog, handles the thesaurus, and the search routines. The small-computer software provides the user with the simple segments of the man-machine dialog, simple editing procedures, buffer/controller storage management, and communications control between the user console and the time-shared computer.

The combination of these two software packages makes a very versatile system such as is required in the types of experiments to be conducted by Project Intrex. Console software² is being developed simultaneously with the console hardware with close communications between the two development programs in order to effect design modifications in both programs so as to produce realizations of tasks that simultaneously make efficient use of hardware and software. This paper will not discuss the details of the software packages.

An example

Let us illustrate some of the features that are being included in the Intrex augmented-catalog and text-access system by means of a simple dialog that could occur during an operating session using the display facilities described here.

The User establishes his right to use the information-transfer system by typing on the console keyboard the appropriate identification. The *System* (S) indicates that it is prepared to service the *User* (U) by displaying READY at the console. The dialog may then continue as follows:

U: Search for information on display consoles.

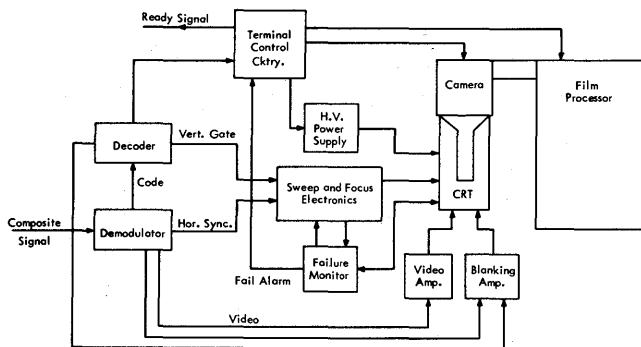


FIGURE 4—The 35-mm film copy terminal

S: Search being made for information on display consoles. Search will be completed within X seconds.

(Two features are noted in the system's reply. First it repeats the query made in order to ensure that it has received the proper request. Second, the number X is determined by the present demands on the System. If it is lightly loaded, for example, X would be very small, whereas a heavily loaded System would have a larger value of X.)

... By X seconds, the dialog continues ...

S. Five hundred documents found. Do you wish titles?

(The number of documents that is found in a search is usually an easy criterion for a user on which to base his next step.)

U: No. Search only for documents on digital-computer display consoles.

S: Fifty documents found. Do you wish titles?

U: No. Search only for documents on alphanumeric consoles published since 1966.

(Two qualifiers [more could be used] have been used in this request, plus, the user simply says "consoles" instead of "display consoles" since all documents now being searched are concerned with display consoles.)

S: Ten documents found. Do you wish titles?

U: Yes.

(Ten documents is a manageable number.)

S: (Displays author, title, publication data, and the library identification numbers of the ten documents on the console CRT.)

U. Erase documents number 2 and number 7.

(By inspection the User knows he does not want to see these two documents. He can make this command in several ways. First, he could point to the document descriptions on the screen with the light pen and push the ERASE function switch. Second, he could point with the cursor and push the ERASE function switch. Third, he could type the command directly.)

S: (Displays the remaining eight documents, renumbering them 1 through 8.)

U: Display technical levels of the documents and group the documents together that are of the same technical level.

S: (Display three groupings of documents. The first group of two documents are under the "primer" heading, the second group of five documents is under the "company report" heading, and the third group of 1 document is under the "recent re-

search" heading. Documents are again renumbered in the order in which they are displayed.)

U: Display the abstracts of these documents. (The User points to documents number 2 and number 8 with light pen.)

S: (Display the two abstracts).

U: Print out abstract and previous data on document number 2.

S: (Makes a hard copy of author, title, publication data, library identification number, technical level and abstract of document number 2.)

U: Is the full text of this document available?

S: It is available in the text-access retrieval unit, on microfilm and in bound volume. Volume on loan. Due back in five days. Call number is 617 369 2257.

(At this point the User could place himself on the waiting list for the volume, he could order a microfilm copy of the document, or he could place the console into a "text-access mode." Suppose he does the latter.)

U: Go into text-access mode, and show me page 1 of document number 1.

S: In text-access mode, document number 617 369 2257, page 1 is being displayed on the text-access console. (This page is displayed on the Storage-Tube Terminal which is adjacent to the augmented catalog console.)

U: Show me the next page of the document.

S: (Displays page 2 on Storage-Tube Terminal.)

(At this point the User can access all pages of the document by "remotely turning pages" back and forth. If, on this inspection, he decides that he wants a copy of a certain page(s) he could request a microfilm copy [or in future systems, paper copy] to be generated at the Film Output Terminal that is located in close proximity to himself. If he found the name of a new reference, or obtained a new lead upon reading the document, he could return to the augmented-catalog mode, and conduct a new search.)

This example can only give one the most rudimentary idea as to the operation of the display facilities being implemented at Project Intrex. Space considerations do not allow further elaborations in the area of operation of facilities.

Console mechanical design

Heretofore not much has been said about the mechanical design features of the consoles. A great deal of design effort is being applied to the human engineering aspects of the consoles since it is imperative that the user's initial contact with the consoles, the only part of

the Intrex experimental library which the average user sees, be a pleasant one. The objectives here are to retain sufficient flexibility in the initial consoles to permit effective user evaluation of various features and options while at the same time to maintain a finished look to the consoles.

The augmented-catalog console takes the form of a two-pedestal desk. One pedestal houses the console electronics and the second pedestal is free for storage of user's materials. Figure 5 is an artist's sketch of the console. The display CRT is in a movable mount that is attached to the center rear of the desk. This places the display directly in front of the user at a comfortable distance from his eyes and hands. The mount allows the display CRT to be moved up or down, left or right, and to be tilted up or down to accommodate user preference. The movable mount is normally firmly locked to its supporting member and is released with a single pushbutton for adjustment. Since the mount is counterweighted, its position can be changed with a minimum amount of effort while the user is sitting down.

The CRT programmable buttons are on the display-CRT mount located at the bottom of the CRT, as shown schematically in Fig. 2. The keyboard is connected to the console with a cable of sufficient length to permit positioning it anywhere on the surface of the desk.

The text-access user terminals are receiving similar attention. Furthermore, integration of the text-access and augmented-catalog user terminals is under study.

SUMMARY

Two basically different computer-driven display facilities for an experimental computer-based library have been described. The first facility is for catalog access. It

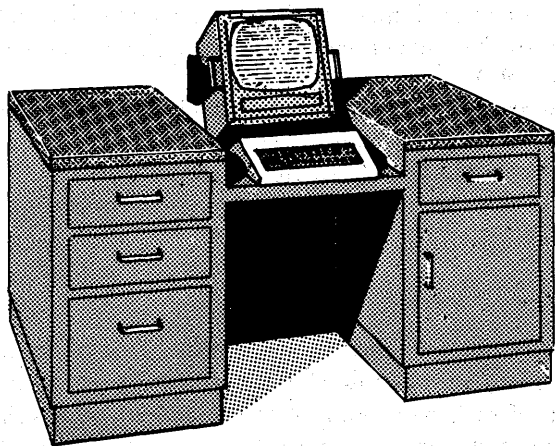


FIGURE 5—Augmented-catalog console

is designed to operate on digitally-stored catalog data in an interactive mode. It consists of several user consoles that are connected by means of coaxial cables to a station containing a drum memory, and a small digital computer. The station in turn is connected by common carrier lines to a time-shared computer to which the data banks for the augmented library catalog are attached. A user console has an alphanumeric, refreshed-CRT display (which is maintained by the drum memory in the station), a typewriter keyboard, mechanical function switches with fixed and dynamically-programmable labels, and a light pen. The small digital computer in the station allows the user to do simple operations on the displayed data and to obtain a certain amount of tutoring on system use without recourse to the central time-shared computer, thus relaxing demands of a single catalog user on the resources of the time-shared computer and the communications facilities serving it.

The second display facility is for text access. It is designed to operate in the full-text photographic images of the documents in the library. It consists of several user terminals and microfilm-processor terminal which are connected to a microfiche storage and retrieval unit by coaxial cable. The storage and retrieval unit in turn is connected to the station containing the drum memory and small digital computer by common carrier lines. A user terminal has a storage CRT that is driven in a facsimile mode to produce a soft copy of the images, and some mechanical function switches. Control of the terminal is through the augmented-catalog console. The microfilm-processor terminal produces microfilm copies of the text image when requested to do so by a user at a text-access terminal.

The display facilities are presently being constructed at the Electronic Systems Laboratory of M.I.T. and is scheduled to become operational the fall of 1968. It is expected that the Intrex experiments will provide new insights into the functional characteristics of display facilities ideally suited for a computer-based augmented library catalog and full-text access. As new functional characteristics are identified they will be incorporated into the facilities described here. Thus, we view the facilities described here as the first of a series of experimental display facilities to be implemented.

ACKNOWLEDGMENT

The author wishes to acknowledge the people in Project Intrex and the Electronic Systems Laboratory for their many contributions to the work reported here.

REFERENCES

- 1 C F J OVERHAGE R J HARMAN (eds)
Intrex report of a planning conference on information transfer experiments

- The MIT Press Cambridge Massachusetts 1965
- 2 *Project intrex semiannual activity reports*
MIT Project Intrex Cambridge Massachusetts 15 March 1967
15 September 1967 15 March 1968 15 September 1968
- 3 A R BENENFELD
Generation and encoding of the project intrex augmented-catalog data base
Proceedings of the 6th Annual Clinic on Library Applications of Data Processing University of Illinois Urbana Illinois 7 May 1968
- 4 D R HARING J K ROBERGE
The augmented-catalog console for project intrex part I
MIT Electronic Systems Laboratory Report ESL-TM-323
October 1967
- 5 D R HARING
A display console for an experimental computer-based augmented library catalog
Proceedings of the 1968 ACM National Conference August 27-29 1968 Las Vegas Nevada
- 6 U GRONEMANN S TEICHER D KNUDSON
Remote text access for project intrex
MIT Electronic Systems Laboratory Report ESL-TM-312
July 1967
- 7 *Project MAC progress reports*
MIT Project MAC July 1963 July 1964 July 1965 July 1966
- 8 D MEISTER D SULLIVAN
Evaluation of user reactions to a prototype on-line information retrieval system
NASA Contractor Report No NASA CR-918 October 1967
Prepared under Contract No NASw-1369 by Bunker-Ramo Corp Canoga Park California
- 9 M M KESSLER
The MIT technical information project
Physics Today Vol 18 pp 28-36 1965
- 10 C T MORGAN J S COOK III A CHAPANIS M W LUND (eds)
Human engineering guide to equipment design
McGraw Hill Book Company Inc New York New York 1963
- 11 H H POOLE
Fundamentals of display systems
Spartan Books Washington DC 1966
- 12 R L SIMMS JR
Trends in computer/communication systems
Computers and Automation May 1968 pp 22-25

Response time in man-computer conversational transactions

by ROBERT B. MILLER

International Business Machines Corporation
Poughkeepsie, New York

INTRODUCTION AND MAJOR CONCEPTS

The literature concerning man-computer transactions abounds in controversy about the limits of "system response time" to a user's command or inquiry at a terminal. Two major semantic issues prohibit resolving this controversy. One issue centers around the question of "Response time to what?" The implication is that different human purposes and actions will have different acceptable or useful response times.

This paper attempts a rather exhaustive listing and definition of different classes of human action and purpose at terminals of various kinds. It will be shown that "two-second response" is not a universal requirement.

The second semantic question is "What is a need or requirement?" In the present discussion, the reader is asked to accept the following definition: "A need or requirement is some demonstrably better alternative in a set of competing known alternatives that enable a human purpose or action to be implemented." This definition intentionally ignores the problem of value versus cost. It is not offered as a universally useful definition of "need." It does enable us to get into a systematic exposition of problems, alternatives and implications. A value-based definition, in contrast to the rational one given here, inevitably leads to a vicious regress that dead-ends only with the agreement that all that humans *really* need are food, water, and a place to sleep.

Another point of view, compatible with the present one, is that need is equivalent to what is demanded and what can be made available; need, therefore, is a cultural and technical outcome. It is the outcome of many vectors, at least one of which is what the marketplace has to offer and the number of Joneses who have one, too.

Operating needs and psychological needs

An example of an operating need is that unless a given airplane's velocity exceeds its stall speed, the airplane will fall to earth. Velocity above stall speed is an undebatable operating need. In a superficially different context, it is a "fact" (let's assume we know the numbers) that when airline customers make reservations over a telephone, any delays in completing transactions above five minutes will reduce their making future reservations with this airline by 20%. A related form of need in this context is that the longer it takes to process one reservation, the larger the number of reservation clerks and reservation terminals that will be required. These are just two examples of the context of operating needs. This report will not look into the problems of operating needs except to mention when they may be more significant than a psychological need. The following topics address psychological needs.

Response to expectancies

Psychological "needs" (in the information processing context) have two major forms, with overlap. One is in the nature of response to an expectation. If you address another human being, you expect some communicative response within x seconds—perhaps two to four seconds. Even though his response may not have the message context you want, you expect him to respond within that time in some fashion, if by no more than a clearing of the throat or a grunt. In conversation of any kind between humans, silences of more than four seconds become embarrassing because they imply a breaking of the thread of communication. This is similar to a phone line going dead. Conditioning experiments (which, of course, should be intro-

duced only with great caution in the context of cognitive activities) suggest an almost magical boundary of two-second limits in the effectiveness of feedback of "knowledge of results," with a peak of effectiveness for very simple responses at about half a second. There is much evidence to suggest that two seconds in human behavior is a relatively long time. Of course even the lower animals can be conditioned (acquire expectancies) to delays, although as the delay is extended the reliability of the performance rapidly deteriorates. The parameters differ for different species.

These points are made only to suggest that the behavior of organisms is time-dependent, and that time spans in the order of one to ten seconds have significance for some forms of behavior involving information transactions with an environment.

Activity clumping and psychological closure

There is a second class of psychological need in communications. This need recognizes that humans spontaneously organize their activities into clumps that are terminated by the completion of a subjective purpose or subpurpose. When I search in a phone book for a telephone number with which to dial a person I want to talk with, I have a sense of temporary completion when I find the telephone number. I have another when I have completed dialing the number. I will more readily tolerate an interruption or delay after such a completion than during the activities preceding this completion. Psychologists call this subjective sense of completion a "closure" and that is the term used henceforth in this report. The rule is that more extended delays may be made in a conversation or transaction after a closure than in the process of obtaining a closure.

Human short-term memory

Here is a rationale for this phenomenon. Performing any task calls for holding a body of information in mind—I call this short-term memory. When I am looking up the telephone number, I am holding in mind the image of the name I am searching for as well as the goal—which is locating this name in the list. When I shift from temporarily memorizing the telephone number to dialing it, short-term memory is holding this set of digits and the goal action of completing the dialing. An interruption or delay in achieving a goal usually results in a degree of frustration. The

longer a content must be held in short-term memory, the greater the chances of forgetting or error. Thus, on both counts (short-term memory and goal aspiration), waiting times within a clump of activities have deleterious effects. A psychological closure results in at least a partial purging of short-term memory or the internal activities that support it.

In very complex problem solving, short-term memory is heavily filled. It is becoming clear in the psychological literature that the degree of complexity of problems that can be solved by a human is dependent on how much information (and in what form) he can hold in short-term memory. Human memory is never passive. Spontaneous noise from within the thinking system, as well as distractions from outside, can interfere with short-term memory contents, and of course these effects rapidly increase when the individual has an awareness of waiting. This awareness comes as soon as several seconds—two seconds still seem to be a good number here.

That is why the tasks which humans can and will perform with machine communications will seriously change their character if response delays are greater than two seconds, with some possible extension of another second or so. Thus, a system with response delays of a standard ten seconds will not permit the kind of thinking continuity essential to sustained problem solving, and especially where the kind of problem or stage of its solution contains a high degree of ambiguity. Such a system will have uses, but they will be different from those of a two-second system.

Psychological step-down discontinuities with increasing response delays

The point here is that response delays are not merely matters of "convenience" to the user, unless the word "convenience" is made to mean more than it usually does. There is not a straight-line decrease in efficiency as the response delay increases; rather, sudden drops in mental efficiency occur when delays exceed a given point. These sudden drops at given delay points can be thought of as psychological step-down discontinuities. Thus, a ten-second response system (aside from operating inefficiencies) may be no better for the human—in some tasks at least—than a one-minute response or a five-minute response. If the human diverts his attention from the thought matrix (e. g., waiting to be filled or completed by

system response to some other train of thought), the significance of response delay changes dramatically.

The statement that "In the past it took two days to get an answer to a question that now is given in fifteen minutes" means, perhaps, an increase in *operating* efficiency for the system, but does not in itself materially change the cognitive (psychological) behavior of the person getting the information.

Psychological closure comes in different degrees. In the telephone example, I get a partial closure when I find the name in a telephone book, another when I complete dialing the number, and another when I am talking to the right person. Talking to the person I have in mind completes the closure of the series of transactions that led to hearing his voice and name. Just as there is a hierarchy of closures in a given task or goal-directed behavior sequence, so there are probably varying amounts of acceptable delays. The greater the closure, the longer the acceptable delay in preparing for and receiving the next response following that closure.

A general rule for guidance would be: For good communication with humans, response delays of more than two seconds should follow only a condition of task closure as perceived by the human, or as structured for the human.

Response time, or system response time, has not yet been defined in this report, so that the two-second rule applies to "meaningful replies" to human requests or commands, and these are defined, along with others in the pages to follow. In addition to definitions and examples of inquiry and response modes, estimates are made of acceptable response times.

Some qualifications about the analysis

The analysis of qualitative behavior and conversion of the analysis into quantitative limits is prone to misinterpretation. This is especially true when the subject is human behavior. Therefore, the following provisos are made explicit.

1. The classes of response categories are not exhaustive.

The seventeen types of response category and response time cited in the next section of this report are certainly not exhaustive of all the possibilities. Without too much

strain, however, they seem to cover a large proportion of interactive behavior between humans and information-processing systems.

2. A response signal can communicate several messages at the same time.

A signal can communicate several messages to the user concurrently. Thus, if the system replies to a user query or command with the statement that is the equivalent of, "I've started doing your work," the user knows (a) his request has been listened to, (b) his request has been accepted, (c) that an interpretation of his request has been made, and (d) that the system is now busy trying to provide him with an answer.

In the next section, the elements listed above are differentiated into four different kinds of response, but in system operation they may (or may not) be combined into a single communication. If so, the response time that should be met is that demanded by the component in the group which demands the fastest response time.

3. The language in the text does not indicate the form of inquiry or response.

In most cases, a topic will be introduced by a title such as "Response to 'Here I am, what work should I do next?'" This expression is intended to simplify communication to the reader of the report. It does not imply that these words would be entered as such into the system. In many cases, the expression of the inquiry, or of the system's response, may be implicit in some other behavior. Thus, lifting the telephone receiver and putting it to my ear has the implicit question, "Are you listening to me and can you give me service?" The dial tone says that it can.

The reader, therefore, is urged to look at the context under a topic title for proper orientation.

4. Tasks can be done in other than the conversational mode.

Whereas in traditional batch activities by computer or by humans, responses to queries may have taken days, a response time of

two seconds may be stipulated in the following pages. Therefore, the critic will ask: "Isn't a response time of 30 seconds or even an hour, better than 24 hours? If so, why isn't it good enough?" The answer must be, "Yes, 30 seconds is better than 24 hours for some purposes, but it is not good enough to maintain the continuity of human thought processes." Where discontinuities in human thought processes are irrelevant or unimportant, both to effective problem solving and to effective use of the professional's time, then the conversational mode is beside the point. But it will be easily demonstrated that many inquiries will not be made, and many potentially promising alternatives will not be examined by the human if he does not have conversational speeds—as defined in this report—available to him. But tasks will still be completed as indeed they have been in the past, without conversational interaction, and at least *some* of them will be completed more poorly by any criterion. This assertion is certainly a testable hypothesis.

5. Permissible ranges of variation are not cited.

Any specification intended for implementation should include not only a nominal value but acceptable tolerance values within which the nominal value may randomly fluctuate. These tolerance limits are not generally specified for most of the response time values cited in the following pages. In principle, the range of acceptable variation of delay in a given category of response time is that range within which the human user cannot detect differences under actual conditions of use. By "use" is meant the context of the human performing a task in which the delay of the response element occurs.

Some laboratory data* of indirect reference are available for making preliminary estimates of response time tolerances. Subjects judged intervals between clicks as "same" or "shorter" or "longer" than a comparison or reference interval between clicks. They gave their full attention to

making these judgments. When the duration of the interval was between 2.0 to 4.0 seconds, the subjects made 75% correct judgment of "same" or "different" at the limits of an interval between minus 8% of the stimulus and plus 8% of the stimulus. For example, 75% of the time an interval of 1.84 seconds was judged shorter than 2.0 seconds, and an interval of 2.16 was judged longer than 2.0 seconds. This is about the same as giving a "tolerance" range of 16% of the value of the stimulus. This value of 16% applies in the range of 2.0 to 4.0 seconds.

The most accurate judgments of time (under these experimental conditions) were between 0.6 and 0.8 seconds where the tolerance range is somewhat less than 10% of the value of the stimulus duration (e. g., 10% of 0.6 seconds delay between clicks). With intervals longer than 4.0 seconds such as 6.0 to 30 seconds, the equivalent tolerance ranges were shown to be 20 to 30%. Substantially the same relationships held where the interval was started and stopped with a pulse of light.

The foregoing results were based on carefully controlled stimuli and full attention to the interval by the subjects. Where the stimulus changes from one display to another, and where there is subjective variability introduced by the human operator making a control response that initiates a machine delay, it is likely that response time variations may exceed these tolerances substantially. By exactly how much would require empirical data from subjects in simulated task environments.

Of indirect significance to this report are the findings by a number of investigators (cited by Stevens) that the time interval that bounds what is subjectively felt as the "psychological present" is between 2.3 to 3.5 seconds, although under some special conditions the boundary may extend to 12 seconds. This interval contains "the physical time over which stimuli may be spread and yet all perceived as present . . . the maximal physical time over which may extend a temporal stimulus pattern . . . which is perceived as a whole."

*See in S. S. Stevens, *Handbook of Experimental Psychology*, Chap. 32, "Time Perception" by H. Woodrow.

Basis of response-time estimates

The estimates of delay times offered in the following pages are the best calculated guesses by the author, a behavioral scientist, who has specialized in task behavior, including thinking and problem solving. These estimates are based on rationales, some of which are cited above and others in context. They should, indeed, be verified by extended systems studies—not in artificial laboratories using abstract tasks—but in carefully designed, real life task environments and problems. The human subjects in these studies must have had many dozens of hours practice in acquiring relevant task skills (and not merely in manipulating the controls at the console) in order for the findings to be useful. Novices have their short-term memory registers heavily filled with what they are trying to learn; therefore, they are not guides as to what the problem-solving user (or other user) will be able to do and want to do when he is highly skilled. Traditional research practices in psychological laboratories would delay answers on these questions for years, however, and perhaps provide them after a new generation of large data-base systems are already on the market.

Nevertheless, the reader should accept the parameters cited as indicative rather than conclusive. It is relatively easy to arrange demonstrations for the skeptic about short response times that will impress him with how long four seconds can seem to be. The demonstration requires merely that he become absorbed, motivated, and emotionally aroused by the demonstration task.

Definitions of response time

Response to human inquiry, with few exceptions, serves as feedback to a continuity of thought. Human behavior occurs at a variety of information handling levels. Different kinds of response and response delay will be appropriate at different behavior levels. The definitions that follow depend in part for their time estimates on psychological rationales given in Part I of this report.

Topic 1. Response to control activation

This is the indication of action given, ordinarily, by the movement of a key, switch or other control member that signals it has been physically activated. The click of the typewriter key, or the

change in control force after moving a switch past a detent position are examples. They indicate responsiveness of the terminal as an object. This response should be immediate and perceived as a part of the mechanical action induced by the operator. *Time delay*: No more than 0.1 second. See also Topic No. 13, "Graphic Response from Light Pen."

A second form of feedback to the user at a keyboard is evidence of the key's being struck. In a typewriter, this is given by the printed character on the paper. This appears practically simultaneously (to the user) to striking or activating the key. Even if printed feedback of text being entered by the user goes through the computer before it is printed on the platen or CRT, the delay between depressing the key and the visual feedback should be no more than 0.1 to 0.2 seconds.

(Note that this delay in feedback may be far too slow for skilled keyboard users. These people are able to attend to the display, not the keyboard, while activating keys, and they will be aware of an out-of-synchronization relationship between eye and hand. Some adaptation can be made—the mechanical pipe organ had delays estimated at between 0.1 and 0.2 seconds. Part of the organist's skill was learning to adapt to this delay. Recognize, however, that the sense of hearing is more time-dependent than the sense of vision.)

If the light pen is used to select characters for a message, confirmation by brightening the selected character should be identifiable by the user within 0.2 second.

Topic 2. Response to "System, are you listening?"

The hum of the dial tone is the response the telephone gives to this implicit query. No dial tone means: "There's no point in trying to do anything further on this channel now."

Time delay: Up to three seconds. The time for onset of this response may be variable, but at some cost in user confidence. Confidence will, of course, be highest if the response signal begins within a second after activating the ON switch.

Comment: These statements apply only to the condition in which the user is becoming "initialized" in a session with the console. If he is actively engaged in a working conversation with the console, he must get immediate (as perceived by him) attention for making an input to the system, such as pressing a control key or other form of

entry. Having to wait four seconds, or even half a second for any reason when he wishes to enter information is violently disrupting to thinking.

This question has two levels. On the first level, the user wants to know if the system is available to work for him. After favorable acknowledgment, the user—depending on his task—will specify the programs and data he requires for his “private working area” at this session.

Topic 3 Response to “System, can you do work for me?”

A. Because in many cases a “yes” or “no” to the question of system availability may depend on the kind of work to be done, the user must key in a request for a given service. As the user does this, he is becoming psychologically locked into a conversation, and his capacity for annoyance with the quality of service is increased.

Time delay: For a routine request (as defined by the user performing a task) the acknowledgment should be within two seconds. A routine request is likely to be a demand for an information image in the store. For an impromptu, complex request, the delay may extend to five seconds.

B. The loading of the programs and data called for by the user should be within 15 seconds, although delays of up to one minute should be tolerable. The user will spend his time during this delay in arranging whatever notes he has, and in organizing his thoughts preparatory to work.

C. Response to the user requesting “Set up my job from where I left off yesterday” should be within 15 seconds for most favorable acceptance, up to one minute for acceptance.

Topic 4. Response to “System, do you understand me?”

This implicit query may precede Topic 3, or be concurrent with it. Assume the user has entered a 7-digit telephone number as a single, meaningful operation. If he has made an error that the system can detect, he should be allowed to complete his segment of thought before he is interrupted or told he is locked out. After two *seconds* and before four *seconds* following completion of keying in his “thought,” he should be informed of his error and either “told” to try again, or told of the error he made.

Comment: It is rude (i.e., disturbing) to be interrupted in mid-thought. The annoyance of the in-

terruption makes it more difficult to get back to the train of thought. The two-second pause enables the user to get his sense of completion following which an error indication is more acceptable.

Topic 5. Response to Identification

Assume a badge-reader type of terminal. The user is on his way to his work station or is at his work station.

He inserts the card, badge, or other identifying medium. Ideally, he should have two kinds of feedback.

1. *Feedback to correctly positioned card.* This should be in the order of direct mechanical response, such as activating a detent or producing a click or snap, with a delay of less than 0.4 to 0.5 second. If failure to position the card properly occurs rarely, this form of feedback is unnecessary.
2. *Feedback saying the equivalent of “OK, I’ve read you.”* This response time should be within two seconds, and be a fixed length of time. In general, people on their way to an activity experience mild annoyance at having their progress interrupted in order to be identified as an employee. The annoyance may be mitigated by making the interruption brief, simple, and standardized so that it can be accomplished practically by a series of reflex actions. That is why the confirmation of the identification should be made to the user in a standard length of response time. When a user clocks out, he is apt to be even more impatient with impediments. Then, a two-second delay will seem four times as long as a one-second delay.

Another factor in identification speed is the bottleneck likely to exist at entrances to work locations where many employees arrive at about the same time. Small lines of employees were informally observed as they punched in at time clocks. Cycle time per employee—when he had his time card in his hand—was about three seconds at the clock. The clock itself had a response time of about one second after the time card was seated. Cutting this response time to 0.5 second would reduce the cycle time per employee by 16%, assuming other factors remained constant. But if the response time

was four seconds, and it were to be cut to one second (and the other factors remained constant), people would pass through the line twice as fast with a one-second delay as with a four-second delay imposed by the action of the mechanism.

Comment: The delays proposed in this section are intended to apply only to that kind of identification implied by the statement, "Here I am and ready to go to work." Where the user sits at an inquiry terminal and says, symbolically, "This is who I am and I want to use your facility," a longer delay in acknowledgment is likely to be acceptable—say, up to five or seven seconds. (Note that this estimate is consistent with that of "Response to, 'System, can you do work for me?'" when the user is initiating an impromptu, complex request. See Topic 3.)

Topic 6. Response to "Here I am, what work should I do next?"

This inquiry is that of a production worker in a factory who has completed an assignment, acknowledged its completion, and requests from the terminal his next assignment. It is likely that this will be displayed to him in the form of a printed slip or card prepared at and by the terminal. Acceptable delays could range from 10 to 15 seconds.

This condition does not apply to the user in conversation with a terminal, such as in computer-assisted instruction. If the student has completed a segment of study and wishes to continue into another topic, the delay should be less than five seconds.

Topic 7. Response to simple inquiry of listed information

This form of inquiry presumes that the query addresses an existing record, or record-string, which can be directly retrieved and displayed. Example: Part #123456: give physical description. Or, Richard R. Roe: give man number. Or, Standard circuit #12345: give description.

If a terminal is frequently used by an employee for this kind of inquiry (say, more than once an hour), the response should be within two seconds. The employee is likely to have in mind some specific issue which the display response may resolve. It is also likely that the employee may have to scan several responses to his queries before hitting on the frame that fits his intent.

Topic 8. Response to simple inquiry of status

An example would be: "Current order status of inventory Part Number 123456." This is a simple inquiry because it asks for one category of information about an unambiguously identified object. The system may have to do some searching and processing from several storage locations to assemble the response. Where the user recognizes this requirement, the two-second delay limit may be relaxed to seven to ten seconds.

The user will be holding an idea in mind while waiting for the response, but it will be a single idea rather than a complex one. For example, "Can I or can't I take an order for 2000 items of this part number?"

Topic 9. Response to complex inquiry in tabular form

A complex inquiry is one which requires collecting and displaying data on the basis of logical relationships among categories. It assumes an "image" of the displayed response does not preexist in the system. An example: "How many orders for Product X, placed since January 1, 1967, have been cancelled to date?" Assume that master records are filed by customer name to which details of the order are added as attributes. These attributes include "date that order was placed," and "status" of which "cancelled" is a subcategory. The system must search these records (perhaps via indexes) and pull out the relevant items. (This is a simple example of complex inquiry.)

The user will certainly have a continuity of ideas in mind when he makes complex inquiries. This particular inquiry should get a complete response within four seconds.

Assume, however, the user had asked the same question for Product X, Y and Z. It would now be acceptable to display the answer about Product X within four seconds, about Product Y within four seconds after that, and about Product Z within the next four seconds.

The principle here is that it takes time for the user to assimilate the elements in a complex pattern. In many situations, four seconds per item would be longer than necessary, and two-second delays would in all cases be preferable.

If the display is graphic rather than tabular in format, additional considerations will apply. (See Topics 13 through 16 on graphics.)

Topic 10. Response to request for next page

Assume a graphic or high-speed printer output at the display. The user has completed reading or skimming a section of text which overruns into another "frame." The user activated the "Next Page" control.

Here, time delay should be no more than one second until (at least) the first several lines of text on the new page appears. You can test the annoyance of longer delays by becoming engrossed in some text and, when you are about to turn the page, be restrained from doing so to a slow count of four by an associate.

Delays of longer than one second will seem intrusive on the continuity of thought.

There is another page-turning condition. This is when the user is searching for some item of content which may lie on any of several pages or frames. A half second is a relatively long time, subjectively, for getting a page turned while searching for items of information.

A problem may be created when the user wants to scan through category indexes and therefore would like to flip pages quickly, unless the index already exists as an "image." In some cases, however, the index may have to be custom-built on the basis of the user's specific request. Where this occurs, the user must be informed that he can expect two-second delays when requesting the next frame of index terms.

If delays in advancing from a previous frame to a next frame in a viewing series are more than two seconds, it is increasingly unlikely that the user will use this medium for scanning and searching. It seems possible that adequate design of the application, however, can minimize the need for impromptu organizations of new indexes on immediate demand.

Skipping a number of pages or frames should be manageable with the help of a displayed index on one segment of the screen. The user should be able to skip ten pages all at once, as rapidly as the next page would appear.

Topic 11. Response to "Now run my problem."

Assume that an engineer or scientist has written a short program to solve a specific equation. He has written the program at the terminal. He presses the GO button.

(a) How long he will wait with patience will

be partly a function of how long he took to write the program and enter the data.

- (b) His patience will also depend on the number of additional data runs or changes he expects to make before selecting a particular set of parameters.
- (c) His patience will also depend on how anxious he is to get back to other work for which the calculated result is a step towards solution.

If the result is returned to him within 15 seconds, he may remain at the terminal "in the problem-solving frame of mind." If the delays are longer, he will, to a corresponding degree, tend not to think of the terminal and system as in-line with his thinking, and attempt to fill in the wait times with secondary activities—probably an unsatisfactory arrangement to him, but less so than staring at a blank screen, or waiting hours for a response from the Computation Center. These interruptions may also *tend* to make him satisfied with a result after less experimentation than if he could continue uninterrupted. (We assume he wants to see an "answer" before he tries another hypothesis.) This is a net loss to both system utilization and a user's problem-solving potential.

Topic 12. Response to delay following keyboard entry vs. light-pen entry of category for inquiry

Let us distinguish between light-pen entry of a category of information (such as a request for a given image or format by touching the light pen to a code name), and using the light pen as a stylus or drawing instrument. In this topic only the use of the light pen as category or function-selector is relevant.

Because it is easier for a nontypist to select instructions by light pen than by keyboard, he will expect a faster response to light pen. The difference may be that between the two-second response time to the light pen, and three-second response time to the keyboard. We can also expect a one to one-and-a-half second adaptation time required by the user for shifting his attention from the keyboard to the display.

This distinction disappears, however, when the user is activating a "page-turning" function on the display he is viewing. If he is continuing the reading of text (graphic or perhaps even tabular material) from one displayed frame to another, one-

second delay after activating the control (light pen or function key) is a maximum. This is too long if he is scanning pages while searching for some specific content. (See Topic 10 which calls for less than one-second response time.) The user who is scanning a series of frames will keep his finger (or the stylus) poised over the "Advance to Next Frame" control, and activate it without shifting his attention from the screen.

Topic 13. Graphic response from light pen

There are two major ways in which the light pen is used as a stylus (as contrasted with its use as a control selector or alphameric message composer). One is that of drawing lines on the scope face where the direction and shape of the line have significance. That is, the actual path travelled by the light pen is the input to the system.

Where the lines are drawn with deliberation by the user—relatively slowly as compared with slashing sketch strokes—a delay of up to 0.1 second seems to be acceptable. There must not be variability perceived by the user in this delay.

Another way of using the light pen for graphics is to compose an image from a "menu" of image parts. For example, a glossary of references at the side of the image frame may be symbols of resistor, diode, transistor, and so forth. The user places his light pen over one of these symbols and moves the light pen to the position on the frame that he wants the symbol to be. A copy of the symbol follows the light pen. The response delay in the image following the light pen may be as much as one second because the user is not tracing a line but positioning an image that, for him, is completed when his stylus touches the destination for the image.

Similar delays of up to one second would be acceptable when the user is constructing the format for a graphic display of, say, a bar chart or line graph from a menu of symbols.

Topic 14 Response to complex inquiry in graphic form

Assume the same kind of inquiry as described in Topic 9 "Response to Complex Inquiry in Tabular Form" except that the response will be a display of bar chart, schematic, or graph.

The graphical response should begin within two seconds and certainly be completed within ten seconds if the user is to maintain thought continuity

in an ongoing task—example, localizing the cause of an exception by means of category search. Other examples of such continuity in thinking would be the use of historical files during problem-solving sessions where the outcomes of these sessions would result in plans and hypotheses for organizational changes (operations research) or for growth (systems analysis).

Note: Many variables cited in previous topics also apply here.

Topic 15. Response to graphic manipulation of dynamic models

It is, of course, possible to animate a diagrammatic representation of a logical system (such as a computer), or a process system (such as a factory or inventory), or a topological system (such as transportation routings and flow). Pulses can simulate messages or transactions, and the thickness of a bar at the input to a symbolic work station may represent the size of a queue. Dynamic changes in the distributions of wait times at each of many stations can be shown on bar charts, whereas changes in the profiles of the bars show different patterns of queues or delays.

Experience with this kind of display is not sufficiently widespread to suggest the limits of analytical perception of human viewers of this kind of graphical simulation. We can expect that after many hundreds of hours of studious effort with this form of display, great improvements in perceptual sensitivity, retention, and interpretation will be achieved by at least some individuals with talent for it.

The problem-solving user will want at least three special properties in this kind of display. One is that of enlarging a segment of a display field. A second is that of selectively suppressing details in the representation of action or structure—similar in principle to going from lower level to higher level diagrams of a mechanism. A third will be an easy means of visually enhancing some given path or paths in a complex representation, while suppressing the remaining content into visual phantoms.

Response-time limits for these functions are not even readily conjectured. The serious problem solver will, of course, be prepared to spend many hours planning and executing the design, optimization, or simulated test of a complex system facility. Flexibility in his ability to get the display

to shift rapidly from one degree of time compression or expansion in simulated system behavior, or from one level of detail to another, will be important. This flexibility will determine how much and how well he can perceive, interpret, hypothesize, control, and modify. But putting minimum limits to the words "flexibility" and "shift rapidly" in the preceding sentences would be premature beyond the guess that whatever "scenario" of events the user must comprehend and work with should be compressible into 50-minute periods of time. Even this may be 10 times greater than the chunk of information that even a problem-solving specialist can hold in mind and work with as a designer or evaluator.

It is here that we need inventive, developmental studies somewhat similar to that conducted by the RAND Corporation in the early 1950's about how much a SAGE operator could assimilate—and under what conditions.

Statements about response times for graphic simulation of dynamic models will, therefore, not include even guesses at this point in knowledge.

Topic 16. Response to graphic manipulation in structural design

Examples of structural modelling are a highway engineer's designing a bridge, or an engineering architect's designing a building.

When the designer adds an element to the design, one system requirement is that of applying sets of algorithmic rules to that design element. For example, "Only one physical body can occupy a given space at one time," or "Building codes require that. . . ." Another system requirement is remembering what the designer has already done. A third requirement is translating sketch responses into the equivalent of appearance renderings and engineering renderings.

The intensity of design conceptualization demands rapid response from the medium on which the designer is working. But the designer will have to accept some constraints (disciplines) in how he attacks and sequences or stages his design effort in order to obtain reasonable system response (i.e., two-second response time, to be informed that he just sketched in a dimension that violates a rule, or type of rule).

During creative effort, idle time beyond a couple of seconds by the designer, while he waits to see the consequence of a unitary action, will be inhibit-

ing and intolerable. But, after the designer has completed working out an idea—a chunk made up of a number of individual actions—he will be inclined to wait a minute or two, while the system "catches up to him."

Comment: People engaged in creative activities recognize the relatively large amounts of work that can be executed during concentrated and continuous "mental heat" in a single session. This heat can cool off in interruptions lasting less than a minute. It is this heat of attention that the system should attempt to preserve.

Graphic motion that the designer perceives as relevant to the design task will help keep his attention and state of arousal, at least if it continues for no longer than ten seconds in consummating some design action. In other words, it is possible to present artifacts to the designer that will maintain his psychological "coupling" to the system. The concept precludes setting fixed response time limits to various response functions, except that their limits will be in seconds (usually) rather than in minutes.

Topic 17. Response to "Execute this command into the operational system."

An example of such a command is a manager's intervening in an automatic ordering process and designating an alternate vendor. Or, the manager may insert a command which, when effected, results in a change in scheduling of some manufacturing operation. Or, as a result of simulation and modelling of certain activities of the business, a revised operating budget is introduced and its implications for a number of affected departments are exploded and disseminated.

Although the user should be informed by the system within four seconds that it has understood and can interpret the command, its execution and final confirmation to the user that the command has been executed may have long and variable delays of minutes. The user has terminated one level of activity when he enters the command. It will be psychologically incomplete only to the degree that he expects a feedback telling him of interference with its execution. These delays, however, are partly dependent on operating activities outside the scope of the automatic system, such as a remote manager's being unable to accept a budget cut or change in schedule.

*Postscript 1***Discontinuity of waiting time at 15 seconds**

Assume an inquiry of any kind has been made. The user—and his attention—is captive to the terminal until he receives a response. If he is a busy man, captivity of more than 15 seconds, even for information essential to him, may be more than an annoyance and disruption. It can readily become a demoralizer—that is, a reducer of work pace and of motivation to work.

If, therefore, response delays of more than 15 seconds will occur, the system had better be designed to free the user from physical and mental captivity, so that he can turn to other activities and get his displayed answer when it is *convenient to him* to do so.

A possible, but doubtful, exception may arise when the user is in series with some process or continuity that demands (as soon as possible) the answer from him, which, in turn, depends on information he is trying to get at the terminal. In this case, the operating demands dictate acceptable time delays.

In any event, response delays of approximately 15 seconds, and certainly any delays longer than this, rule out *conversational* interaction between human and information systems.

*Postscript 2***Time recovery from errors and failures**

A dimension of response time is the question, "How quickly can I get going on my task again after something goes wrong?" What may have gone wrong could have been a machine failure, a failure in an operating program, an operator error, or an error by the user in mid-task.

The design of the system, including the application, should simplify both the effort and shorten the time required for recovery as perceived by the user.

If the user was in simple-inquiry mode, he will probably have a record of his last inquiry at the terminal, and can input the inquiry again.

If the user was in complex-inquiry mode, the last *index of categories* that he was using before the failure should have been retained and made available to him, so that he can pick up his inquiry from a position of good context.

If the user was in conversational problem-solving mode, there should have been retained a copy of all the parameters and starting structure of the model he constructed. Reconstructing this model would, from the user's standpoint, be the most arduous and unreliable of activities. (As an example of this almost universal dread of work getting lost, many writers and engineers save their yellow-sheet draft sketches in desk drawers until the job is entirely completed.) One can tolerate the loss of a machine run, which can be rerun later, but the loss of even an hour's creative work is obviously demoralizing. Rarely does one feel confidence that the reconstruction has all of the magic contained in the original.

When a system failure occurs, from whatever cause, the user is likely to feel an irrational sense of failure if his job has been lost. In some degree, it will be remembered as personal failure, and various psychological defenses will be inevitable. (One form of defense is to avoid the cause of the threat in the future.) It is therefore desirable, for motivational reasons as well as operating reasons, to attempt to restore the system as quickly as possible so that he can pick up and continue. "As quickly as possible" means "while he is still in dialogue (or work session) with the system"—and that means within 15 seconds, or failing that, within less than five minutes. The system should tell him how long he may have to be patient, and it should do so immediately after the failure, whatever it may be.

BIBLIOGRAPHY

- W BLAKELY
The discrimination of short empty temporal interval
PhD dissertation University of Illinois Library 1933
- J R NEWMAN
Extension of human capability through information processing and display systems
System Development Corp Santa Monica December 1966
- H SACKMAN
Experimental investigation of user performance in time-shared computing systems
System Development Corp Santa Monica May 1967
- H SIMON R K MELLON
Reflections on time-sharing from a users point of view.
In Computer Science Research Review
Carnegie Institute of Technology 1966
- S S STEVENS
Handbook of experimental psychology
Wiley N Y 1951

Linguistic methods in picture processing—A survey*

by W. F. MILLER and A. C. SHAW**

Stanford Linear Accelerator Center
Stanford, California

INTRODUCTION

By "picture processing" we mean the analysis and generation of pictures by computer, with or without human interaction; this definition includes both computer graphics and digital pattern recognition.

A number of people have advocated that picture processing problems be attacked with linguistic methods; perhaps the strongest early exponents were Narasimhan¹ and Kirsch.² The basic idea was to extend the notions of syntax and semantics to n -dimensional patterns ($n > 1$) and then apply some adaptation of the techniques of natural and artificial language processing. Several researchers have attempted to develop this concept during the last few years. While the work is still experimental, several practical uses have been demonstrated and ideas seem to be emerging that could form the basis of a picture theory.

This paper surveys research in linguistic methods for describing and processing pictures. The next section discusses the rationale and application area for a linguistic approach. We then present a general linguistic picture processing model as a basis for the survey discussion. The central idea within this model is that of a formal picture description. The survey itself is contained in section IV. In the concluding section we extract some common features and difficulties, and indicate directions for future research.

Models for picture processing

The term "model" denotes the general framework or "paradigm"³ within which workers pose and solve problems. Until recently, most theoretical work in picture *analysis* has, either implicitly or explicitly, been

based on the receptor/categorizer model (RCM) described in Marill and Green.⁴

The analysis of pictures (or pattern recognition) proceeds within the RCM: A picture is first reduced to a "feature set" by the receptor; this is a set of quantities which may range from the raw digitized values at one extreme to the results of a complex feature extraction process on the other. The feature set is then assigned to one of a finite number of classes or patterns by the categorizer. The assignment is the recognized pattern class to which the picture supposedly belongs. Most of the theory has dealt with the problem of categorization or classification. The principal technique is one of treating the feature or measurement set as a point in a multi-dimensional space. The task of the categorizer then becomes one of partitioning the space so that measurements from pictures belonging to the same pattern class are "close" (according to some metric) and measurements from pictures of different classes are far apart. (Sebestyen⁵ and Nilsson⁶ are references for the RCM.)

The RCM is the basis for a number of recognition systems, notably in character recognition.⁷ The model fails to be useful for analyzing complex pictures where the *structure and interrelationships* among the picture components are the important factors. To illustrate this point in a simple setting, consider the one-dimensional pattern recognition task required of a programming language translator. One purpose of the syntax analysis phase of the compiler is to categorize an input program into one of two mutually exclusive classes—the class of syntactically correct programs and its complement. Theoretically, one can envision a receptor which produces a feature vector from an input program; the categorizer then determines in which of the two possible subspaces the feature vector lies. While this can be done in principle, it is never considered seriously because of the complexities involved; for example, what is the feature set for a program? Even if this approach were practically feasible for program classification, it would

*Work supported by U.S. Atomic Energy Commission and National Science Foundation, Grant GP-7615.

**Present Address: Department of Computer Science, Cornell University, Ithaca, N.Y.

not produce the most important by/product of a successful analysis, i.e., a description of the structure of the input program.

Richly-structured pictures that are difficult, if not impossible, to analyze within the RCM include those produced in particle detector chambers by high-energy particle physics reactions; text and standard two dimensional mathematical notation (not isolated characters); line drawings, such as flow charts, circuits, and mechanical drawings; and complex biomedical pictures. What is required in these examples is a description of the pictures in which the meaningful relations among their subparts are apparent. The appropriate place to apply the RCM is for the recognition of the basic components of the pictures. In a series of papers, Narasimhan^{1,8,9,10} has forcefully stated this case:

“Categorization, clearly, is only one aspect of the recognition problem; not the whole of it by any means. It is our contention that the aim of any recognition procedure should not be merely to arrive at a ‘Yes,’ ‘No,’ ‘Don’t know’ decision confusion about aims might have been avoided if, historically, the problem had been posed as not one of pattern *recognition* but of pattern *analysis and description*.”¹¹

Much of the research in computer graphics* has been concerned primarily with data structures¹¹ and command and control languages. Picture descriptions are embedded in the data structures; in fact, the data structure *is* the description. This could be viewed as a linguistic specification of a picture since the structure (syntax) and values or interpretations of each structure (semantics) are explicitly contained in the data structure in most cases. However, the processing (analysis or synthesis) of the pictures is not directed *by* the data structure description but rather *towards* them through the mand and control languages.

In this survey we shall consider only those works where some attempt is made to describe pictures and classes of pictures, and use these descriptions to direct the processing. The analogy to linear language processing is evident and hence the term “linguistic model”¹² is employed.

A general linguistic picture processing model

The linguistic model for picture processing¹² is comprised of two parts:

*“Computer graphics” has usually referred to that set of techniques for computer processing of pictures using on-line displays and plotting equipment.

**Narasimhan¹ first used this term as applied to picture processing.

1. a general model within which pictures may be described (i.e., a meta-description formalism) and
2. an approach to the analysis and generation of pictures based directly on their descriptions.

The description, D , of a picture, α , will consist of two parts—a *primitive* or terminal symbol description T , and a *hierarchic* description H . T specifies the elementary patterns in the picture and their relationship to one another and H describes groupings of the elements into higher level structures. This can be written $D(\alpha) = (T(\alpha), H(\alpha))$. T and H , in turn, each have a *syntactical* (or *structural*) component T_s and H_s , and a *semantic* (*interpretation or value*) component T_v and H_v . That is,

$$T(\alpha) = (T_s(\alpha), T_v(\alpha))$$

$$H(\alpha) = (H_s(\alpha), H_v(\alpha)) .$$

$T_s(\alpha)$ names the elementary component classes or *primitives* in α and their relationship to one another; $T_v(\alpha)$ gives the values or meaning of the primitive components of α . The primitives in $T_s(\alpha)$ will denote classes; let $\mathcal{P}(T_s)$ be the set of all pictures with primitive structure T_s . We present a simple example of a primitive description T .

Example 1

Let ℓ name the set of all straight line segments and c name the set of all circles. ℓ and c are picture primitives. Let \odot denote the geometric relationship of intersection. Then, if a picture α contains a line segment α_1 intersecting a circle α_2 , its primitive description $T(\alpha)$ might be:

$$T_s(\alpha) = \ell \odot c \quad T_v(\alpha) = (v_\ell(\alpha_1), v_c(\alpha_2)) ,$$

where $v_\ell(x)$ is the pair of endpoint coordinates of the line x and $v_c(x)$ is the center coordinates and radius of the circle x . $\mathcal{P}(\ell \odot c)$ is the set of all pictures consisting of a line segment intersecting a circle.

Consider a set of rules or grammar \mathcal{G} generating a language $\mathcal{L}(\mathcal{G})$ whose “sentences” are primitive structural descriptions. Then, \mathcal{G} is said to describe the picture class $\mathcal{P}_{\mathcal{G}} = T_s \in \mathcal{L}(\mathcal{G}) \mathcal{P}(T_s)$. For a given picture $\alpha \in \mathcal{P}_{\mathcal{G}}$, the hierarchic structural description $H_s(\alpha)$ is the ordered set of rules of \mathcal{G} that were used to generate $T_s(\alpha)$; that is, $H_s(\alpha)$ is the “linguistic” structure or parse of $T_s(\alpha)$ according to \mathcal{G} . A one-to-one correspondence exists between the elements of a set \mathcal{I} of semantic or interpretation rules and the elements of \mathcal{G} . $H_v(\alpha)$ is

defined as the result of obeying the corresponding semantic rule for each rule of \mathcal{G} used in $H_s(\alpha)$.

Example 2

Let \mathcal{G} be the phrase structure grammar¹³:
 $\mathcal{G} = \{LC \rightarrow L, LC \rightarrow C, LC \rightarrow L \odot C, L \rightarrow \ell, C \rightarrow c\}$.
 Then $\mathcal{L}(\mathcal{G}) = \{\ell, c, \ell \odot c\}$ and $\mathcal{P}_{\mathcal{G}} = \mathcal{P}(\ell) \cup \mathcal{P}(c) \cup \mathcal{P}(\ell \odot c)$. We interpret the terminal symbols ℓ, c , and \odot as in Example 1 and let

$$\mathcal{I} = \{v_{LC} := v_L, v_{LC} := v_C, v_{LC} := \text{xsect}(v_L, v_C), v_L := v_\ell, v_C := v_c\}.$$

The k^{th} rule of \mathcal{J} corresponds to the k^{th} rule of \mathcal{G} for $k = 1, \dots, 5$. Within a rule, v_i designates the value associated with the syntactic unit i in the corresponding grammar rule; xsect is a function that computes the intersection(s) of a line with a circle, and v_ℓ and v_c are defined in Example 1. If $T_s(\alpha) = \ell \odot c$ for a given $\alpha \in \mathcal{P}_{\mathcal{G}}$, $H(\alpha)$ could be represented by the simple tree of Figure 1 where $\alpha = \alpha_1 \cup \alpha_2$, $\alpha_1 \in \mathcal{P}(\ell)$, $\alpha_2 \in \mathcal{P}(c)$, $v_\ell = v_\ell(\alpha_1)$, and $v_c = v_c(\alpha_2)$.

It is important to emphasize that the "meaning" of a picture will be expressed in *both* its primitive and hierarchic descriptions. Thus, several grammars may be used to generate the same class of primitive descriptions, but the hierarchic descriptions, and hence the meaning, may be different for different grammars. Even more generally, the same picture class may be described by totally different primitive and hierarchic descriptions; the intended interpretation of the picture dictates its description.

With the description model, our approach to picture processing can now be formulated:

1. The elementary components or primitives which

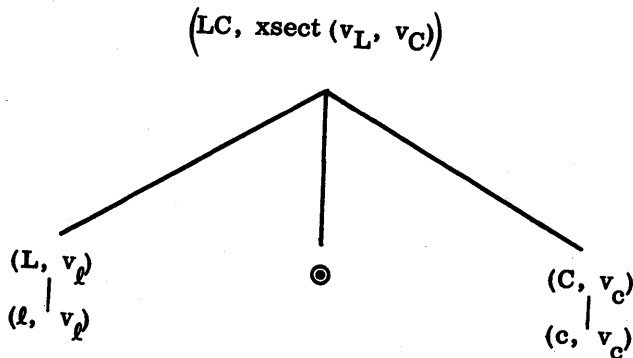


FIGURE 1—Hierarchic description of a picture

may appear in a class of pictures are named and defined.

2. The picture class is described by a generative grammar \mathcal{G} and associated semantics \mathcal{I} .
3. A given picture α is then *analyzed* by parsing it according to \mathcal{G} and \mathcal{I} to obtain its description $D(\alpha)$; that is, \mathcal{G} and \mathcal{I} are used explicitly to *direct* the analysis.
 Conversely, a picture α is *generated* by executing its description $D(\alpha)$.

Descriptions are then not only the results of an analysis or the input to a generation, but they also define the algorithms that guide the processing. This approach provides a framework in which picture processing systems may be implemented and theoretically examined. The arguments for treating analysis and synthesis problems together, i.e., using a common description scheme, are generality, simplicity, and the universal use of common description languages in science. We also note that most picture analysis applications have (and need) an associated generative system and vice versa; there are also many situations where both a synthesis and an analysis capability are equally important, for example, in computer-aided design.

Syntax-directed translation of programming languages^{14,15} can be interpreted within our model as the analysis of patterns of linear strings. In this case, the primitive description is obtained immediately—the input program corresponds to T_s , and the meaning of the basic symbols of the language to T_v . The grammar \mathcal{G} is generally a BNF grammar plus some constraints on the use of identifiers; the semantics \mathcal{I} is most often a set of code-generating rules. The analysis of a well-formed program yields the syntactic structure of the program and an equivalent program in some other language.

We find it most illuminating to evaluate picture processing research within the framework of the above model. In each case, the various components of the particular descriptive scheme— T_s , T_v , H_s , and H_v —are extracted and discussed in terms of their power and limitations. We are interested in the description mechanism both as a language of discourse about pictures and as a driver for analysis or generation systems.

The survey

The literature survey of Feder¹⁶ covers the few basic developments up to and including 1965; since then, there has been a relatively large surge of activity.

Early Developments

There are several early works that explicitly utilized primitive descriptions. Grimsdale *et al.*,¹⁷ produced geo-

metric descriptions of hand-drawn line figures, such as alphabetic characters; the description consisted of an encoded list of the picture curves, their connectivity, and geometric properties. Sherman¹⁸ reduced a hand-printed letter to a graph, and then built a character description out of the topological and geometric features of the abstracted picture. Neither T_s , nor T_v is defined formally in the above examples; picture analysis (recognition) occurs by comparing or matching picture descriptions with descriptions of standard patterns.

Eden^{19, 20} presented a formal system for describing handwriting. His primitive elements are a set of basic "strokes" or curves; the value of each stroke is a point pair (the endpoints) and a direction. Eden gives a set of rules \mathcal{G} for concatenating or collating strokes to form letters and words. The description T_s of a word of handwriting is then a sequence of n -tuples of strokes, each n -tuple representing a letter. This is one of the first works where the author recognizes the benefits of a generative description:

"Identification by a generative procedure leads to a clear definition of the set of permissible patterns. The class of accepted patterns is simply the set which can be generated by the rules operating on the primitive symbols of the theory."²

Eden did not report any attempts at using his scheme for recognition purposes; however, his descriptions were used for generation.

In Minsky,²¹ we find one of the earliest arguments for the use of "articular" or structured picture descriptions in pattern recognition. Minsky suggests a description language consisting of expressions of the form (R, L) , where L is an ordered list of subpictures or figures related to one another by the relation R . For example, $(\rightarrow, (x, y))$ might indicate that the figure y is to the right of the x . Expression composition within the elements of the list L permits the description of complicated structures; using the above notation, $(\rightarrow, ((\rightarrow, (a, b)), c))$ means that b is to the right of a , and c is to the right of the subpicture containing a and b . Although it is not explicitly linguistic, this work has influenced several later efforts (see discussion under Evans).

Narasimhan

The pioneering work in suggesting and applying a linguistic model for the solution of non-trivial problems in picture processing was done by Narasimhan.^{1, 8, 9, 10, 22, 23} He first proposed a general linguistic approach in 1962, calling it a "linguistic model for patterns"; he has since experimented with it in the analysis of bubble chamber

photographs using a parallel computer,^{1, 9, 10, 22} and in the generation of "handprinted" English characters.^{10, 23} Narasimhan restricts his model to the class of pictures containing only thin line-like elements.

We first discuss the analysis model in Narasimhan's 1962 paper.¹ Here, T_s is a list of the "basic sets" and their connectivity. Basic sets refer to neighborhoods on the picture having specified topological properties, for example, the neighborhood about the junction of two lines or the neighborhood about an endpoint of a line. Two sets are said to be connected if there exists a "road" or line-like element between them. T_v is the value of the sets (their topological meaning) and the geometry of the connecting roads. An informal set of rules \mathcal{G} then describes how strings of connected sets may be combined into other strings and phrases; phrases are of the form: $\langle \text{name} \rangle (\langle \text{vertex list} \rangle)$, for example, $ST(1, 2, 3)$, where the $\langle \text{vertex list} \rangle$ labels those points that may be linked to other phrases. Finally, there are additional rules of \mathcal{G} for combining phrases into sentences. The hierarchic description H_s of a picture is a list of sentences and phrases. Analysis proceeds from the "bottom up," first labeling all points as basic sets or roads, then forming phrases and, last of all, sentences.

Narasimhan does not define a general form for either \mathcal{G} or the description D . In the bubble chamber application, the hierarchic system of labeling imposed by \mathcal{G} is slightly different than above, starting with points at the most primitive level; \mathcal{G} is implicitly defined by the computer program itself. On the other hand, the generation of English "hand-printed" characters is explicitly directed by a finite-state generative grammar \mathcal{G} and an attribute list \mathcal{G} , the latter specifying some geometric properties of the characters, for example, position, length, and thickness. The primitives are simple geometric forms, such as straight lines or arcs; the definition of each primitive includes a set of labeled vertices to which other primitives may be attached. Productions or rewriting rules in \mathcal{G} are the form:

$$S(n_s) \rightarrow S_1 \cdot S_2(n_{s1s2}; n_{s1s}; n_{s2s}),$$

where S_1 is a terminal symbol (primitive name) or non-terminal symbol (phrase name), S_2 is terminal symbol, S is a non-terminal symbol—the defined phrase—, $n_{s_1 s_2}$ is a list of nodes of concatenation between S_1 and S_2 , n_{s1s} and n_{s2s} define the correspondence between the nodes of S_1 and S_2 that those of S , and n_s is a node list labeling the nodes of S . Figure 2 illustrates Narasimhan's rewriting rules for generating the letter "P," the primitives required, and the generated letters. All nodes of possible concatenation must appear in the description; this is cumbersome for simple pictures such as the English alphabet, and might be unmanageable for more

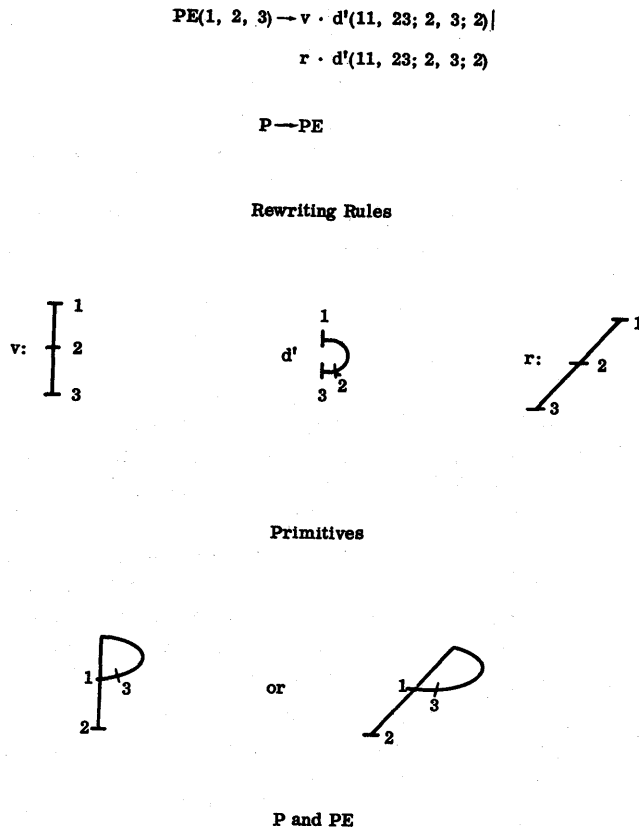


FIGURE 2—Narasimhan's generation of the letter "P"

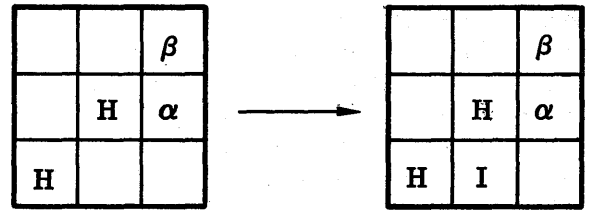
complex pictures. The system can only describe connected pictures and some other mechanism is required when dealing with pictures whose subparts are not connected. This scheme has been used successfully as part of an experimental system for the computer generation of posters.²³ To our knowledge, it has not been applied to other picture classes.

Kirsch

Kirsch,² in a stimulating article, argues that the proper way to view picture analysis is within a linguistic framework. Following this line of thought he poses several problems: How does one

1. express picture syntax or structure,
2. generalize the idea of concatenation to several dimensions,
3. describe geometric relations among picture components,
4. do syntax analysis of pictures, and
5. define picture primitives?

Kirsch gives a two-dimensional context-dependent grammar for 45° right triangles generated in a plane divided into unit squares; this is suggested as an illustration of the possible form of picture grammars.



Sample Production: $\alpha \in \{L, I\}$, $\beta \in \{H, W\}$

				W
			H	L
		H	I	L
	H	I	I	L
V	B	B	B	R

A Derived Triangle

FIGURE 3—Kirsch's right triangle description

Figure 3 contains a sample production and a derived triangle. Here, T_s is a two-dimensional 45° right triangle with labeled unit squares (the primitives); T_v is the meaning of the labels. There is no semantic portion corresponding to the grammar. As Kirsch admits, it is not evident how this approach may be generalized for other pictures; it is also a debatable point whether context-sensitive grammars are desirable since the analysis would be extremely complex. More recently, Lipkin, Watt, and Kirsch²⁴ have argued persuasively for an "iconic" (image-like or picture) grammar to be used for the analysis and synthesis of biological images with a large interactive computer system; however, the search for suitable iconic grammars continues. The work of Kirsch and his colleagues is notable for their clear and early recognition of the importance of a linguistic approach to picture processing problems and for their detailed enumeration of some of the difficulties.

Ledley

Ledley²⁵ and Ledley *et al.*²⁶ employed a standard BNF grammar to define picture classes. Their pub-

lished method for the analysis of chromosomes^{26,27} illustrates this approach. Here Ledley's "syntax-directed pattern recognition" is embedded in a large picture processing system that searches a digitized picture for objects, recognizes the primitives of an object, performs a syntax analysis of the object description, and finally computes further classifications and some statistics on all the chromosomes found. The object primitives consist of five types of curves from which chromosome boundaries can be generated. An edge-following program traces the boundary of an object in the picture and classifies each boundary segment into one of the primitive classes; since the boundary is a closed curve, a linear string or ordered list of its segment types is sufficient for the description T_s . If T_s represents a chromosome, the parse H_s will contain a categorization of it as, for example, submedian or telocentric in type; otherwise the parse fails, indicating the original object was not a chromosome. Figure 4 contains samples from the chromosome syntax, examples of the basic curve, types, and some chromosome descriptions. Ledley and Ruddle²⁷ state that the human complement of 46 chromosomes can be processed in about 20 seconds (on an IBM 7094) using this system—a factor of 500 as

compared to manual methods—but no data are given on the quantity of pictures examined and error rates, or how their methods compare with others, for example, chromosome classification by moment invariants.²⁸ Ledley's work is an example of a direct application of artificial language analysis methods to picture classification. It is difficult to generalize this approach to figures other than closed curves unless relational operators are included as part of T_s ; in the latter case, the most difficult task is obtaining T_s , not parsing the resulting string.

Guzmán

Guzmán^{29,30} describes pictures consisting of sets of isolated points and concatenated straight line segments using a figure description language (FDL). The primitive syntax T_s is given in FDL by listing every node in the figure and its immediate neighbors, and adjoining to this an arbitrary property list; T_v is a list of the actual coordinates of each node. Figure 5 contains two possible descriptions of an isosceles triangle and one of a quadrangle and a rectangle. Hierarchic descriptions and the equivalent of a grammar may be specified in FDL by assigning names to both primitive descriptions, and sets of names and descriptions. This is illustrated at the bottom of Figure 5, where a POLY is defined as either a RECT or an ISOS1. Several figures may be concatenated to form new ones by listing in a = TIE = statement the nodes of concatenation. The FDL language is used to drive some general scene analysis programs. A given scene is first preprocessed to produce a symbolic description in terms of points forming line segments and isolated points. A scene analysis program then accepts a series of "models" described in FDL and searches the scene for all or some instances of the models. Experiments with the system have served to pinpoint a number of extremely difficult problems associated with the analysis of two-dimensional projections of three-dimensional objects. While restricted to concatenated straight line segments and isolated points, the FDL language has some very desirable features. Chief among these is the ability to define "open" or bound variables (X, Y, and A1 in Figure 5) in the property list; this allows an elegant description of the relations among picture components.

Evans

The earlier work of Evans^{31,32} on solving geometric-analogy-intelligence test problems employed picture description methods similar to those suggested by Minsky.²¹ Recently, Evans³³ has developed a linguistic formalism for picture description and an associated pattern analyzer that is driven by a "grammar" \mathcal{G} written in the formalism. The syntax of a class of pictures is

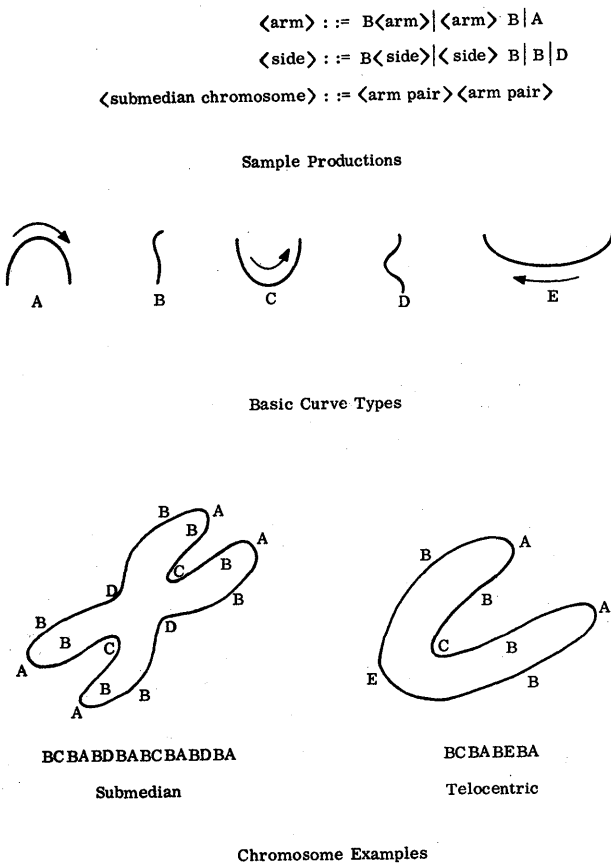
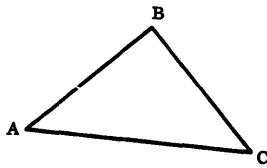
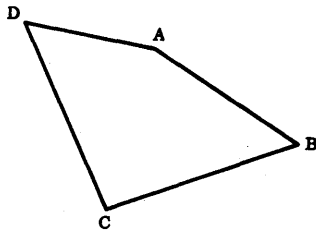


FIGURE 4—Ledley's chromosome description



```
(=DEF= ISOS1 (( A (B C) C (B A) B (A C)) where ((LENG A B X)
              (LENG B C X) (VARIABLES X))))
(=DEF= ISOS2 (( A (B C) C (B A) B (A C)) where ((ANGLE B A C A1)
              (ANGLE B C A A1) (VARIABLES A1))))
```



```
(=DEF= QUADR ( A (B D) B (C A) C (D B) D (A C)))
(=DEF= RECT (QUADR where ((LENG A B X) (LENG D C X)
                          (LENG A D Y) (LENG C B Y) (ANGLE D A B 90°)
                          (VARIABLES X Y))))
(=DEF= POLY (=OR= (RECT ISOS1)))
```

FIGURE 5—Guzmán's FDL notation

given by a set of rules, each of which has four components: (L R P I) (our notation). An example of a rule that we will use in the discussion below is:

```
(TRIANGLE (XYZ) (( VERTEX X) (VERTEX Y)
                 (VERTEX Z) (ELS X Y) (ELS Y Z) (ELS X Z)
                 (NONCOLL X Y Z)) (( VERTICES (LIST X Y Z))))
```

The first component, L, names the construct or pattern whose components are defined by R and P; in the example, the pattern TRIANGLE is named. R is a list of "dummy" variables, one of which is associated with each constituent of the defined pattern. P is a list of predicates which names the pattern type represented by each dummy variable, and describes the relationships that must exist among these patterns. X, Y, and Z are named as type VERTEX; ELS is a predicate which tests for the existence of a line segment between two points, and NONCOLL tests for noncollinearity among 3 points. The last part I of the syntax rule can specify any computation over the properties of the pattern components; during analysis, it assigns the result to the new construct defined by the rule. After a successful analysis TRIANGLE will have attached to it the name VERTICES followed by a list of the values of X, Y, and Z. These attached properties can then be

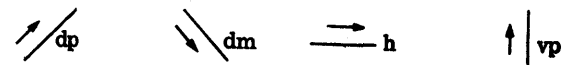
used by predicates in subsequent syntax rules. In terms of our model, the I component can be viewed as part of the syntax in some instances or as an interpretation or semantic rule of \mathcal{L} in others.

Evan's pattern analyzer assumes that a picture is first preprocessed to produce a list of its primitive elements and their properties; this is the primitive description T. The pattern analyzer (a LISP²⁴ program) accepts a preprocessed picture and a grammar, and parses the picture to produce hierarchic descriptions of all patterns satisfying the grammar; the library of predicates may first have to be extended if new relational predicates appear in the grammar. While the description and analysis systems are very general, they have only been tested on simple examples and it is too early to predict how useful they will be.

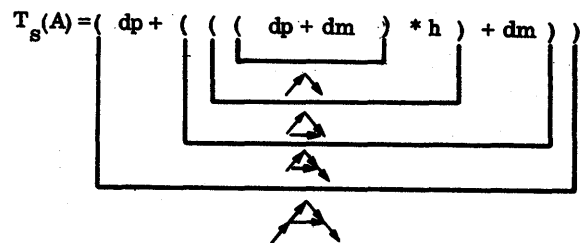
Shaw, Miller, and George

In the Shaw papers^{12, 35} a picture description language (PDL) is presented and applied. PDL is a language for expressing the primitive structural description T_s of a picture. The basic components or primitives may be any pattern having two distinguished points, a tail and a head; primitives can be concatenated together only at these points. The PDL language can describe the concatenations among any connected set of primitives. By allowing the definition of blank (invisible) and "don't care" primitives, a large class of pictures may be described in terms of concatenations and simple relations among their primitive elements; these include photographs produced in high energy particle physics experiments, characters, text, flow charts, and line drawings of all varieties.

Figure 6 illustrates the use of PDL to describe a sim-



Primitive Classes



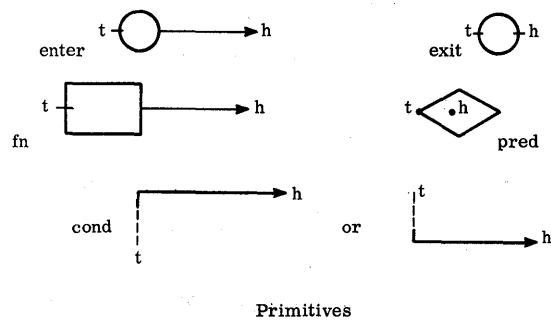
$T_s(F) = (vp + (h \times (vp + h)))$

FIGURE 6—Shaw's PDL language

ple "A" and an "F." For each primitive, the figure contains its class name, a typical member, and an arrow pointing from its tail to head; for example, h denotes the set of all horizontal line segments of a restricted length, with tail at the left endpoint and head at the right endpoint. h can be defined more precisely either theoretically or pragmatically by an equation, an attribute list, a recognition program, or a generation program. The tree beneath the "A" indicates how the letter is generated from its description. The operators +, x, and * describe particular combinations of tail/head concatenations of their operands. Each PDL expression, and the pictures they describe, has a tail and head defined respectively as the tail of the first element and head of the last element in the expression. Thus (S₁ + S₂) has a tail equal to the tail of S₁ and a head equal to the head of S₂; the "+" describes the concatenation of the head of S₁ to the tail of S₂. One more binary operator (—), a unary tail/head reversal operator (~), and a "rewriting" convention complete the description scheme. PDL has a number of useful formal properties that permit descriptions to be transformed into more convenient forms for processing, and forms the basis of a picture calculus discussed in Miller and Shaw.³⁶ The primitive semantic description T_s consists of a list of the primitives and their attributes.

A hierarchic structure is imposed on a class of pictures by means of a restricted form of context-free grammar G generating sentences in PDL. Figure 7 contains several productions from a flow chart grammar for a small ALGOL-like language. The tail and head of each primitive are labelled t and h respectively. The line segments with arrow heads leading from *enter*, *fn* and *cond* may be any sequence of concatenated segments thus allowing the head of these primitives to be placed anywhere in a picture relative to the tail. The box in *fn* is a function box and *pred* represents a predicate or test. *cond* may be either the true or false branch of the predicate; the initial blank (dotted) part at its tail carries it to one of the vertices of the diamond. In the syntax, the / and superscript labels indicate "rewriting" so that both appearances of TEST in the STEPUNTIL rule refer to exactly the same entity. The hierarchic structural description H_s is defined as the parse of T_s according to G; no mechanism for attaching arbitrary semantics to G has been developed yet.

A goal-oriented picture parser (analyzer) (Shaw¹²) accepts a pattern recognition routine for each primitive class and a grammar and uses the latter to direct the recognizers over pictures and produce their primitive and hierarchic descriptions; tail and head pointers are moved over the two- or three-dimensional picture space in a manner analogous to the movement of a string pointer in linear language analysis. An implemented sys-



Primitives

STMNT → BASIC | CNDTNL
 BASIC → ASSIGN | FOR | BLOCK^b
 FOR → STEPUNTIL | WHILE
 STEPUNTIL → (INIT + (((TEST^{su} + cond) + STMNT^{su})
 *(~ INC) × (/TEST^{su} + cond)))
 INIT → fn
 INC → fn
 TEST → pred

Partial Flow Chart Syntax

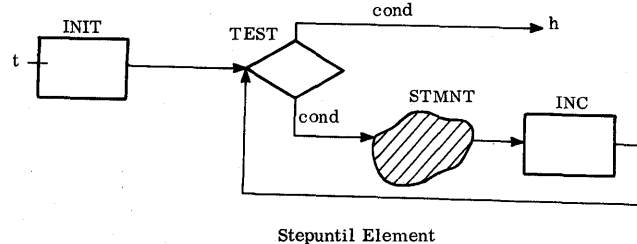


FIGURE 7—Example of Shaw's flow chart descriptions

tem has been applied to the analysis of some digitized spark chamber film. Each picture consisted of a data box with 22 identification digits; 4 fiducial markers ("X"s); and 2 views of 6 spark chambers containing sets of isolated and collinear sparks. 39 syntax rules were used to describe the possible contents of all pictures. The description D(α) of each picture α was produced in approximately 7 seconds on an IBM 360/50. With the picture parser available it took less than 2 man months to put together the spark chamber system. The spark chamber application, even though experimental, has demonstrated certain pragmatically useful advantages of the above methods. These may be summarized as follows: There can be significant simplifications in implementing and modifying picture analysis systems

and one need not pay an exorbitant price in computer processing time when compared with the more ad hoc systems in use in various physics laboratories.

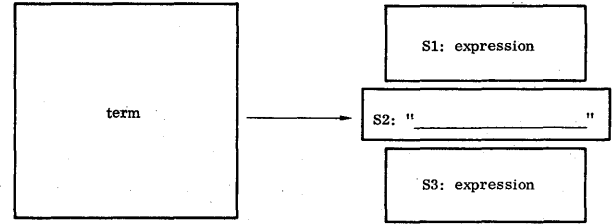
George³⁷ and George and Miller³⁶ employ PDL as the basis of an interactive graphics system. Pictures are generated and modified on-line by manipulating PDL descriptions. Pictures can be stored and retrieved by assigning names to their descriptions; the picture data structure is the PDL description itself so that the machine always contains a structured representation. Any changes to a named subpicture are immediately reflected in all pictures that refer to it as a component. The chief limitations of the descriptive scheme are the restricted set of relations that may be expressed, the practical constraints resulting from only two points of concatenation for a primitive and the absence of a general mechanism for hierarchic semantics.

Anderson

Anderson^{39,40} syntactically analyzes standard two-dimensional mathematical notation after the primitive elements or characters have been classified by conventional pattern recognition techniques. The value T_v of a primitive is its name and 6 positional coordinates: $X_{min}, X_{center}, X_{max}, Y_{min}, Y_{center}, Y_{max}$, where $(X_{min}, X_{max}, Y_{min}, Y_{max})$ define the smallest enclosing rectangle of the character and the point (X_{center}, Y_{center}) is its typographic center. Each syntax rule consists of four structural parts (elements of \mathcal{G}) and one semantic part (element of \mathcal{J}). Figure 8 contains a typical syntax rule. The meaning of the notation is as follows:

- Si: the i^{th} syntactic unit of the right part of the rule.
- Pi: a partitioning predicate that Si must satisfy. c_{ij} is the j^{th} positional coordinate of Si; the positional coordinates above are numbered from 1 to 6 so that c_{13} represents the 3rd coordinate (X_{max}) of syntactic unit S1. c_{0j} refers to the j^{th} coordinate of an arbitrary character in the syntactic unit.
- R: a predicate testing the spatial relationship among successfully parsed elements of the right part of syntax rule.
- Ci: each higher level structure (syntactic unit) is given 6 positional coordinates similar to those of a primitive. C_i $i = 1 \dots 6$ defines the 6 coordinates assigned to the left part of the syntax rule in a successful parse.
- M: the semantic rule indicating an action to be taken or the meaning to be given to the rule.

The mathematical expression $\frac{a^2 + b}{c}$ satisfies the syntax of "term" in the figure; the typographic center is



Graphical Form of Replacement Rule

S1: expression	P1: $c_{01} > c_{21}$ and $c_{03} < c_{23}$ and $c_{04} > c_{26}$	C1: c_{21} C2: c_{22}
S2: horizline	P2: \emptyset	C3: c_{23}
S3: expression	P3: $c_{01} > c_{21}$ and $c_{03} < c_{23}$ and $c_{06} < c_{24}$	C4: c_{34} C5: c_{25}
R: \emptyset		C6: c_{16}
M: $(s_1)/(s_3)$		

Tabular Form of Replacement Rule

FIGURE 8—Example of Anderson's syntax rules

(C2, C5) which is defined in the replacement rule as (c_{22}, c_{25}) , the center of the primitive "horizline." Anderson has described several non-trivial classes of pictures in this notation including two-dimensional arithmetic expressions, matrices, directed-graphs, and a proposed form for a two-dimensional programming language.

A top-down goal-directed method is used for analysis; the basic idea is to use the syntax directly to partition picture space into syntactical units such that the predicates Pi and R are satisfied. The analysis algorithm has been implemented in several experimental systems and tested with hand-printed arithmetic expressions in an interactive mode. He assumes that the primitive characters are correctly classified by recognition routines and that the expressions satisfy some reasonable constraints on their form, for example, the limits above and below an integral sign must not extend further to the left than the leftmost edge of the integral sign. Simple expressions can then be parsed successfully in a reasonable amount of computer time. The expression $\int_1^N |x| dx$ takes approximately 5 seconds to analyze on an IBM 360/50 with an unoptimized PL/I version of the general system; a program optimized especially for mathematical notation and running on the Digital

Equipment Corporation PDP-1 takes less than half a second to recognize the expression $\frac{1 + Z - Z^2}{1 - 1}$.

One of the virtues of Anderson's model is the provision for arbitrary predicates to test spatial relationships as part of the syntax. In order to handle this generality, the analysis algorithm must test a large number of possible partitionings of the picture space before rejecting an inapplicable syntax rule. However, in the case of mathematical notation, increased efficiency can be obtained by taking advantage of its normal left-to-right flow. While adequate for driving a picture analyzer for a restricted class of pictures, the descriptive scheme does not appear suitable for synthesis problems. (Anderson argues that these two aspects of picture processing are fundamentally different and should be treated by entirely different methods.) Anderson has demonstrated the feasibility of interactive mathematics using the above concepts; he concludes, and the authors concur, that future efforts could be directed towards engineering such systems.

Other Works

We conclude the survey by noting several other works which employ either linguistic methods or closely related techniques.

Clark and Miller⁴¹ use the language of graph theory to describe spark linkages and the topology of physics "events" appearing in spark chamber film. These descriptions are embodied in computer programs that apply elementary graph theory to assist in the decision-making process and perform the film analysis. The primitive elements of the pictures are sparks; a multi-list structural provides the description T , and T , of the spark connectivities. Hierarchic descriptions result from combining sparks according to their geometric and graph properties to form tracks and events. While an explicit linguistic approach is not employed, the underlying graph model acts as a formal description language much as in the work of Sherman and Narasimhan. The above program formed the basis for a practical production system that was used for several physics experiments.

Clowes⁴² employs a set \mathcal{G} of Boolean functions on pictures to define the syntactic classes for hand-written numerals; the successive execution of these functions from the bottom up serves to analyze and describe the pictures. More recently, he⁴³ has been working on a scheme based on transformational grammars and Chomsky's model for natural language syntax. Other efforts which are explicitly linguistic in nature include Feder,⁴⁴ Watt,^{45,46} Inselberg and Kline,⁴⁷ Inselberg,⁴⁸ Breeding,⁴⁹ Knot and Wiley,⁵⁰ and Nir.⁵¹

A related area of research has been pursued by Kirsch,² and, more recently, by Coles⁵² and others. Natural language statements about pictures are translated into some formal notation, usually the predicate calculus; the predicate calculus statement then describes a set of pictures—those for which the statement has the truth value of *true*. The natural language statement "Each polygon smaller than a black triangle is a square," could be translated into the predicate calculus as " $(\forall x) (P(x) \wedge (\exists y) (B(y) \wedge T(y) \wedge Sm(s,y)) \supset Sq(x))$ " which may be read as "for all x , if x is a polygon and if there exists a y such that y is black and y is a triangle and x is smaller than y , then x is a square." The predicate calculus expression directs a picture analyzer to determine the truth value of the statement with respect to a given picture. By these methods, Coles⁵² is able to recognize some fairly complicated electric circuits and chemical molecules drawn on a computer-controlled display. One of the aims of this research is to provide interactive question-answering systems with a pictorial data base. A principal, and extremely difficult, problem is that of translating natural language input to the formal notation. In terms of our description model, the predicate calculus statement is the primitive structural description T_s ; hierarchic descriptions do not exist in these schemes.

CONCLUSIONS

As this survey indicates, there has been a great deal of research in picture description methods and associated processing systems; most of this is recent and is still in progress. A principal reason for this research has been the lack of adequate techniques for dealing with complex richly-structured pictures; we felt that relevant techniques are now emerging. All of the reported work is experimental in the sense that, to our knowledge, there do not exist any "production systems" that employ linguistic methods to any large extent. However, in several instances, notable in the work of Anderson⁴⁰ and the authors,^{12,36} the benefits and practicality of these methods have been demonstrated.

With the exception of Kirsch's triangle example,² all of the descriptive schemes are basically linear. One suspects that the development of explicit two- and three-dimensional picture languages would lead to much greater insight into picture processing problems (and, quite possibly, human perception); we are still waiting for a breakthrough in this direction. Regardless of the detailed forms of the descriptive notation and grammars in the various systems, each syntax rule essentially specifies a list of patterns and a set of relations satisfied by them. Practical analysis systems will clearly have to restrict the class of pictures and the types of relations that may exist among the elements of a picture. This

is entirely analogous to the linear language situation where extremely efficient parsers exist when the grammar form and class of languages are restricted, for example, in simple precedence grammars.⁵³ One of the most difficult problems in pattern analysis is the classification of primitive patterns; in many situations, ambiguities and recognition failures can be resolved by examining the picture field surrounding the pattern in question, i.e., by using contextual information. Most of the surveyed works assume that the primitive elements have been classified before entering the analysis; in Shaw,¹² the grammar \mathcal{G} directs the primitive recognizers about the picture and assists the classification process by using the contextual information embedded in \mathcal{G} . Work in this direction should be pursued further. Finally, we note that, with the exception of Eden^{19,20} Narasimhan,^{10,23} and Shaw, Miller, and George,^{12,35,36,37,38} the research has been concerned only with the analysis of pictures. As we argued in section III, there are advantages in treating both analysis and synthesis problems within the same formalism. However, picture generation using formal description schemes has not yet been examined in depth and remains a fruitful area for future work.

REFERENCES

- 1 R NARASIMHAN
A linguistic approach to pattern recognition
Report No 21 Digital Computer Laboratory University of Illinois July 1962
- 2 R A KIRSCH
Computer interpretation of English text and picture patterns
IEEE Transactions on Electronic Computers EC-13 4 August 363-376 1964
- 3 T S KUHN
The structure of scientific revolutions
The University of Chicago Press Chicago 1962
- 4 T MARILL D M GREEN
Statistical recognition functions and the design of pattern recognizers
IRE Transactions on Electronic Computers EC-9 4 December 472-477 1960
- 5 G S SEBESTYEN
Decision-making processes in pattern recognition
The Macmillan Company New York 1962
- 6 N J NILSSON
Learning machines
McGraw-Hill New York 1965
- 7 *Character recognitions*
British Computer Society London 1967
- 8 R NARASIMHAN
Syntactic descriptions of pictures and gestalt phenomena of visual perception
Report No 142 Digital Computer Laboratory University of Illinois July 1963
- 9 R NARASIMHAN
Labeling schemata and syntactic description of pictures
Information and Control 7 151-179 1964
- 10 R NARASIMHAN
Synta-directed interpretation of classes of pictures
Comm ACM 9 3 March 166-173 1966
- 11 J C GRAY
Compound data structure for computer aided design: a survey
Proc of 22nd National Conference of ACM Thompson Book Co Washington 1967 355-365
- 12 A C SHAW
The formal description and parsing of pictures
PhD Thesis Computer Science Department Stanford University Stanford California Also published as CS 94 Computer Science Department Stanford University and SLCA Report No 84 Stanford Linear Accelerator Center Stanford California 1968
- 13 N CHOMSKY
Syntactic structures
Mouton and Co London 1957
- 14 J FELDMAN D GRIES
Translator writing systems
Comm ACM 11 2 February 77-113 1968
- 15 A C SHAW
Lectures notes on a course in systems programming
Report No CS 52 Computer Science Department Stanford University December 1966
- 16 J FEDER
The linguistic approach to pattern analysis—a literature survey
Technical Report 400-133 Department of Electrical Engineering New York University February 1966
- 17 R L GRIMSDALE F H SUMNER C J TUNIS T KILBURN
A system for the automatic recognition of patterns
Paper No 2792 M The Institution of Electrical Engineering December 210-221 1958
- 18 H SHERMAN
A quasi-topological method for machine recognition of line patterns
Proceedings of the International Conference on Information Processing UNESCO Paris 1959 232-238
- 19 M EDEN
On the formalization of handwriting
Proceedings of Symposia in Applied Mathematics American Mathematical Society 12 83-88 1961
- 20 M EDEN
Handwriting and pattern recognition
IRE Transactions on Information Theory IT-8 2 160-166 1962
- 21 M MINSKY
Steps toward artificial intelligence
Proceedings of the IRE 49 1 January 8-30 1961
- 22 R NARASIMHAN
A programming system for scanning digitized bubble-chamber negatives
Report No 139 Digital Computer Laboratory University of Illinois June 1963
- 23 R NARASIMHAN V S N REDDY
A generative model for handprinted English letters and its computer implementation
Technical Report No 12 Computer Group Tata Institute of Fundamental Research Bombay India June 1966
- 24 L E LIPKIN W C WATT R A KIRSCH
The analysis synthesis and description of biological images
Annals of the New York Academy of Sciences 128 3 January 984-1012 1966
- 25 R S LEDLEY

- Programming and utilizing digital computers*
McGraw-Hill New York 1962 Chapter 8
- 26 R S LEDLEY L S ROTOLO T J GOLAB
J D JACOBSEN M D GINSBERG J B WISLON
FIDAC: film input to digital automatic computer and associated syntax-directed pattern recognition programming system
Optical and Electro-Optical Information Processing Tippee J Berkowitz D Clapp L Koester C and Vanderburgh Jr A (Eds)
MIT Press Cambridge Massachusetts 1965 Chapter 33
- 27 R S LEDLEY F H RUDDLE
Chromosome analysis by computer
Scientific American 40-46 April 1966
- 28 J W BUTLER M K BUTLER A STROUD
Automatic classification of chromosomes
Proceedings of the Conference on Data Acquisition and Processing in Biology and Medicine Pergamon Press New York 1963
- 29 A GUZMÁN
Scene analysis using the concept of a model
AFCLR-67-0133 Computer Corporation of America Cambridge Massachusetts 1967
- 30 A GUZMÁN
Some aspects of pattern recognition by computer
MAC-TR-37 Project MAC Massachusetts Institute of Technology February 1967 M S thesis
- 31 T G EVANS
A program for the solution of a class of geometric-analogy intelligence-test questions
PhD Thesis Department of Mathematics Massachusetts Institute of Technology Cambridge Massachusetts 1963
Available as Physical and Mathematical Sciences Research Paper No 64 Air Force Cambridge Research Laboratories L G Hanscom Field Massachusetts
- 32 T G EVANS
A heuristic program to solve geometric-analogy problems
Proceedings of the AFIPS Spring Joint Computer Conference Spartan Books Inc Washington D C 1964 327-338
- 33 T G EVANS
A description-controlled pattern analyzer
Proceedings of the IFIP Congress Edinburgh 1968
- 34 J McCARTHY P W ABRAHAM D J EDWARDS
T P HART M I LEVIN
LISP 1.5 programmers manual
The MIT Press Cambridge Massachusetts 1962
- 35 A C SHAW
A proposed language for the formal description of pictures
GSG Memo 28 Computation Group Stanford Linear Accelerator Center Stanford California February 1967 internal report
- 36 W F MILLER A C SHAW
A picture calculus
Emerging Concepts in Graphics University of Illinois November 1967 in press
- 37 J E GEORGE
Picture generation based on the picture calculus
GSG Memo 50 Computation Group Stanford Linear Accelerator Center Stanford California December 1967 internal report
- 38 J E GEORGE W F MILLER
String descriptions of data for display
SLAC-PUB-383, Stanford Linear Accelerator Center Stanford California Presented at 9th Annual Symposium of the Society for Information Display 1968
- 39 R H ANDERSON
Syntax-directed recognition of hand-printed two-dimensional mathematics
Proceedings of the ACM Symposium on Interactive Systems for Experimental Applied Mathematics to be published 1967
- 40 R H ANDERSON
Syntax-directed recognition of hand-printed two-dimensional mathematics
PhD Thesis Applied Mathematics Harvard University Cambridge Massachusetts 1968
- 41 R CLARK W F MILLER
Computer-based data analysis systems at Argonne
Methods in Computational Physics Adler B Fernbach S and Rotenberg M (Eds) Volume 5 Academic Press New York 1966 47-98.
- 42 M B CLOWES
Preception picture processing and computers
Machine Intelligence 1 Collins N and Mitchie D Eds Oliver and Boyd London 1967 181-197
- 43 M B CLOWES
A generative picture grammar
Seminar paper No 6 Computing Research Section Commonwealth Scientific and Industrial Research Organization Canberra Australia April 1967
- 44 J FEDER
Linguistic specification and analysis of classes of patterns
Technical Report 400-147 Department of Electrical Engineering New York University October 1966
- 45 W C WATT
Morphology of the Nevada cattlebrands and their blazons—Part One
National Bureau of Standards Report No 9050 US Department of Commerce 1966
- 46 W C WATT
Morphology of the Nevada cattlebrands and their blazons—Part Two
Department of Computer Science Carnegie-Mellon University Pittsburgh Pennsylvania 1967
- 47 A INSELBERG R KLINE
A syntactic and contextual pattern recognizer: a preliminary study
Technical Memo 45 Computer Systems Laboratory Washington University St Louis Missouri October 1967
- 48 A INSELBERG
An approach to the syntax-directed analysis of graphic data
Technical Memo 52 Computer Systems Laboratory Washington University St Louis Missouri January 1968
- 49 K BREEDING
Grammar for a pattern description language
Report No 177 Department of Computer Science University of Illinois May 1965 M S Thesis
- 50 P J KNOKE R G WILEY
A linguistic approach to mechanical pattern recognition
Digest of the First Annual IEEE Computer Conference Chicago Illinois September 142-144 1967
- 51 M NIR
Recognition of general line patterns with application to bubble chamber photographs and handprinted characters
PhD Thesis Electrical Engineering University of Pennsylvania Philadelphia Pennsylvania 1967
- 52 S COLES
Syntax directed interpretation of natural language
PhD Thesis Carnegie Institute of Technology Pittsburgh Pennsylvania 1967
- 53 N WIRTH H WEBER
Euler—a generalization of Algol and its formal definition: Part I
Comm ACM 9 1 January 13-25 1966

Decomposition of a visual scene into three-dimensional bodies

by ADOLFO GUZMÁN

*Massachusetts Institute of Technology
Cambridge, Massachusetts*

INTRODUCTION

We consider visual scenes composed by the optical image of a group of bodies. When such a scene is "seen" by a computer through a film spot scanner, image dissector, or similar device, it can be treated as a two-dimensional array of numbers, or as a function of two variables.

At a higher level, a scene could be meaningfully described as a conglomerate of points, lines and surfaces, with properties (coordinates, slopes, ...) attached to them.

Still a more sophisticated description could use terms concerning the bodies or objects which compose such a scene, indicating their positions, inter-relations, etc.

This paper describes a program which finds bodies in a scene, presumably formed by three-dimensional objects. Some of them may not be completely visible. The picture is presented as a line drawing.

When SEE—the pretentious name of the program—analyzes the scene **TRIAL** (see Figure 1 'TRIAL'), the results are:

(BODY 1. IS :6 :2 :1)

(BODY 2. IS :11 :12 :10)

(BODY 3. IS :4 :9 :5 :7 :3 :8 :13)

SEE looks for three-dimensional objects in a two-dimensional scene. The scene itself is not obtained from a visual input device, or from an array of intensities or brightness. Rather, it is assumed that a preprocessing of some sort has taken place, and the scene to be analyzed is available in a symbolic format (to be described in a later Section), in terms of points (vertices), lines (edges), and surface (regions).

SEE does not have a pre-concieved idea of the form or model of the objects which could appear in a given

scene. The only supposition is that the bodies are solid objects formed by plane surfaces; in this way, it can not find "cubes" or "houses" in a scene, since it does not know what a "house" is. Once SEE has partitioned a scene into bodies, some other program will work on them and decide which of those bodies are "houses."

Thus, SEE is intended to serve as a link between a pre-processor^{1,2} which transforms intensity pictures into point or line pictures,⁵ and a recognizer (such as TD³ or DT⁴), which handles this line picture and finds bodies, objects or zones matching with certain patterns or models. Instead of searching through the whole scene looking for parts to match its models, the work of the recognizer becomes simpler after SEE has partitioned the scene into bodies, because the data to be searched (matched) are smaller and better organized.

The analysis which SEE makes of the different scenes generally agrees with human opinion, although in some ambiguous cases it behaves rather conservatively. Distributed over these pages, the reader will find examples of scenes analyzed by SEE, and the peculiarities and behavior of the program will become clear.

The program SEE, written in LISP, has been tested in the PDP-6 machine of the Artificial Intelligence Group, Project MAC, at Massachusetts Institute of Technology. A preliminary version, written in CON-VERT,⁶ was used extensively for a quick test of ideas which shaped the program to its actual form. The analysis of a scene takes from 30 to 90 seconds, with the program running interpreted under the interpreter of the LISP programming system.

A more technical description of SEE can be found in an unpublished memorandum.⁷

Related work

Rudd H. Canaday⁸ in 1962 analyzed scenes composed of two-dimensional overlapping objects, "straight-

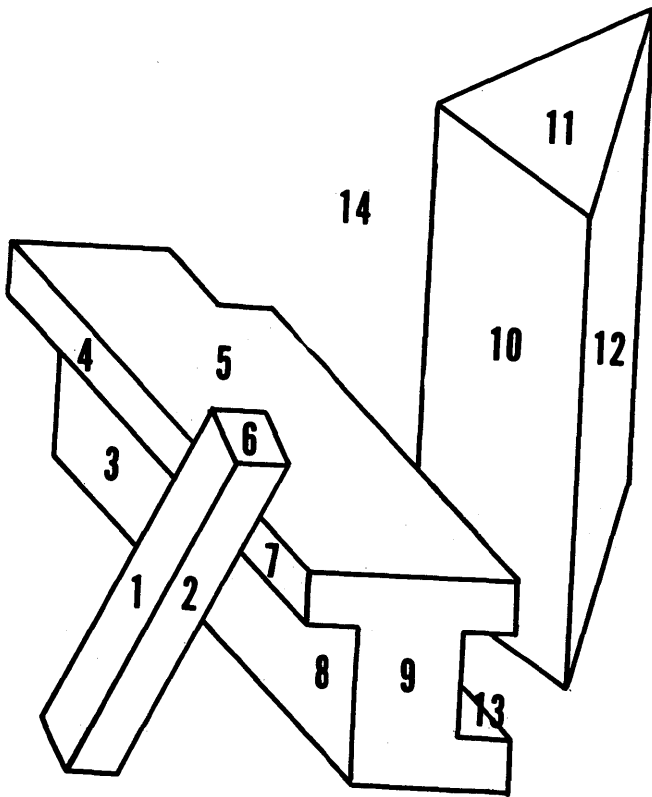


FIGURE 1—TRIAL

The program analyzes this scene and finds 3 bodies:

```
(BODY 1 IS :6 :2 :1)
(BODY 2 IS :11 :12 :10)
(BODY 3 IS :4 :9 :5 :7 :3 :8 :13)
```

sided pieces of cardboard." His program breaks the image into its component parts (the pieces of cardboard), describes each one, gives the depth of each part in the image (or scene), and states which parts cover which.

Roberts [9] in 1963 described programs that (1) convert a picture (a scene) into a line drawing and (2) produce a three-dimensional description of the objects shown in the drawing in terms of models and their transformations. The main restriction on the lines is that they should be a perspective projection of the surface boundaries of a set of three-dimensional objects with planar surfaces. He relies on perspective and numerical computations, while SEE uses a heuristic and symbolic (i.e., non-numerical) approach. Also, SEE does not need models to isolate bodies. Roberts' work is probably the most important and closest to ours.

Actually, several research groups (at Massachusetts Institute of Technology, ¹⁰ at Stanford University, ¹¹ at Stanford Research Institute ¹²) work actively to-

wards the realization of a mechanical manipulator, i.e., an intelligent automata who could visually perceive and successfully interact with its environment, under the control of a computer. Naturally, the mechanization of visual perception forms part of their research, and important work begins to emerge from them in this area.

Organization of the paper

It is formed by the following headings:

- Introduction and related previous work.
- Input Format. The representation of the scene as it is entered to the computer.
- Format of a Scene. The representation of the scene as SEE expects.
- Type of Vertices. Classification of vertices according to their topology.
- The program. Analysis of the algorithm, description of heuristics.
- Interesting examples. Discussion. Future work.

Input format

For testing purposes, the scenes are entered by hand in a simplified format (called input format), and then some routines convert this to the form required by SEE. Eventually, the data will come from a visual input device, through a preprocessor.^{2,5}

Examples of a scene

Suppose we want to describe the scene 'CUBE.' We begin by giving (in LISP) a value to 'CUBE.' (See Figure 2 'Cube')

```
(SETQ CUBE (QUOTE (A 1.0 1.0 (:1 B :4 G)
                    B 1.0 5.0 (:1 E :2 C :4 A)
                    C 3.0 7.0 (:2 D :4 B)
                    D 8.0 7.0 (:2 E :3 F :4 C)
                    E 6.0 5.0 (:2 B :1 G :3 D)
                    F 8.0 3.0 (:3 G :4 D)
                    G 6.0 1.0 (:1 A :4 F :3 E)
                    )))
```

Thus we associate with each vertex its coordinates and a list, in counterclockwise order, of regions and vertices radiating from that vertex.

The conversion of the scene, as just given, into the form which SEE expects, is made by the function LLENA; thus, (LLENA CUBE) will put in the prop-

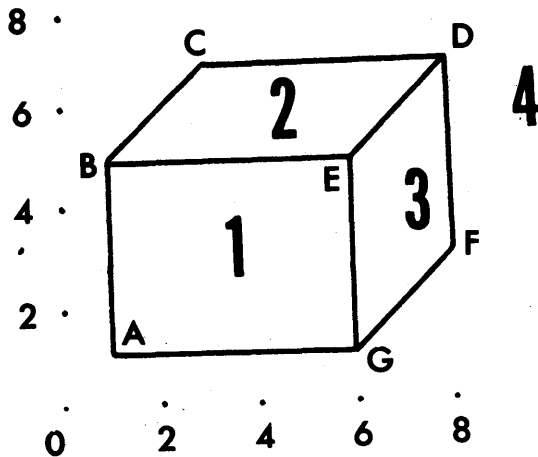


FIGURE 2—'CUBE'—A scene.

erty list of CUBE the properties REGIONS and VERTICES; in the property list of each vertex, the properties XCOR, YCOR, NREGIONS, NVERTICES and KIND are placed; and in the property list of each region, it places the properties NEIGHBORS and KVERTICES. It also marks region :4 as part of the background.

In other words, LLENA converts a scene from the 'Input Format' to the 'Format of a Scene' described in the next section.

Format of a scene

A scene is presented to our program as a scene in a special symbolic format, which we now describe. Essentially, it is an arrangement of relations between vertices and regions.

A scene has a name which identifies it; this name is an atom whose property list contains the properties 'REGIONS,' 'VERTICES' and 'BACKGROUND.' For example, the scene ONE (see Figure 3 'ONE') has the name 'ONE.' In the property list of 'ONE' we find

- REGIONS — (:1 :2 :3 :4 :5 :6)
Unordered list of regions composing scene ONE.
- VERTICES — (A B C D E F G H I J K)
Unordered list of vertices composing the scene ONE.
- BACKGROUND— (:6)
Unordered list of regions composing the background of the scene ONE.

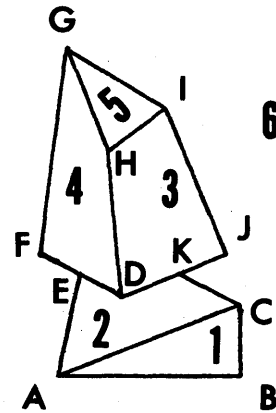


FIGURE 3—'ONE.' A scene. Vertices and surfaces (regions) are the main components of a scene.

Region

A region corresponds to a surface limited by a simple connected curve. For instance, in ONE, the surface delimited by the vertices A B C is a region, called :1, but G I, J K D H is not.

Each region has as name an atom which possesses additional properties describing different attributes of the region in question. These are 'NEIGHBORS,' 'KVERTICES,' and 'FOOP'. For example, the region in scene ONE formed by the lines AE, ED, DK, KC, CA, has ':2' at its name. In the property list of :2 we find:

- NEIGHBORS — (:3 :4 :6 :1 :6)
Counterclockwise ordered list of all regions which are neighbors to :2. For each region, this list is unique up to cyclic permutation.
- KVERTICES — (D E A C K)
Counterclockwise ordered list of all vertices which belong to the region :2. This list is unique up to cyclic permutation.
- FOOP — (:3 D :4 E :6 A :1 C :6 K)
Counterclockwise ordered list of alternating neighbors and kvertices of :2. This list is unique up to cyclic permutation.

The FOOP property of a region is formed by a man who walks on its boundary always having this region to his left, and takes note of the regions to his right and of the vertices which he finds in his way.

Vertex

A vertex is the point where two or more lines of the scene meet; for instance, A, G, and K are vertices of the scene ONE.

Each vertex has as name an atom which possesses additional properties describing different attributes of the vertex in question. These are 'XCOR,' 'YCOR,' 'NVERTICES,' 'NREGIONS,' and 'KIND.' For example, the vertex H (see scene ONE) has in its property list:

- XCOR — 3.0
x-coordinate
- YCOR — 15.0
y-coordinate
- NVERTICES — (I G D)
Counterclockwise ordered list of vertices to which H is connected. This list is unique up to cyclic permutation.
- NREGIONS — (:3 :5 :4)
Counterclockwise ordered list of regions to which H belongs. This list is unique up to cyclic permutation.
- KIND — (:3 I :5 G :4 D)
Counterclockwise ordered list of alternating nregions and nvertices of H. This list is unique up to cyclic permutation.

The KIND property of a vertex is formed by a man who stands at the vertex and, while rotating counterclockwise, takes note of the regions and vertices which he sees.

NREGIONS and NVERTICES are then easily derived from KIND: take the odd positioned elements of KIND, and its even-positioned elements, respectively.

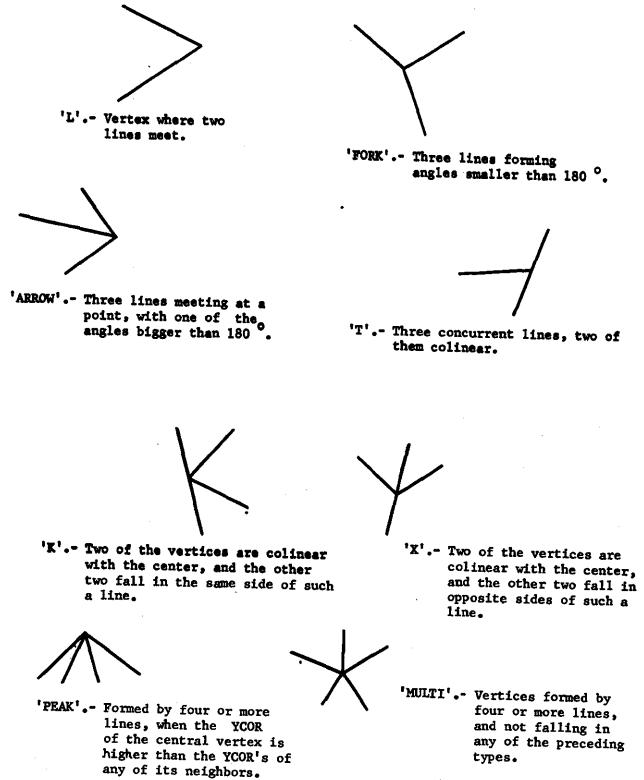
Types of vertices

Vertices are classified according to the slope, disposition and number of lines which form them. The function (TYPEGENERATOR L), where L is a list of vertices, performs this classification, putting in the property list of each one of the elements of L, the type to which it belongs.

The TYPE of a vertex is always a list of two elements; the first is the *type-name*: one of 'L,' 'FORK,' 'ARROW,' 'T,' 'K,' 'X,' 'PEAK,' 'MULTI'; the second element is the *datum*, which generally is a list, whose

form varies with the *type-name* and contains information in a determined order about the vertex in question. (See Table I 'VERTICES').

TABLE I—'VERTICES' Classification of vertices of a scene.



The program

The program named SEE accepts a scene expressed in the notation described above and produces outputs lists identifying and describing the bodies present in the scene.

In this section we describe the program, and how it achieves its goals, by discussing the procedures, heuristics etc., employed and the way they work. We begin with several examples.

Example 1. Scene 'STACK' This scene (see Figure 4 'STACK') is analyzed by SEE, with the following results:

- (LOCAL ASSUMES (:5) (:13 :14) SAME BODY)
- (BODY 1. IS :1 :3 :2 :18 :17)
- (BODY 2. IS :4 :16 :15)
- (BODY 3. IS :7 :6 :11 :12)
- (BODY 4. IS :9 :8 :10)
- (BODY 5. IS :13 :14 :5)
- (BODY 6. IS :20 :19)

Results for scene STACK

Example 2. Scene 'BRIDGE.' With this example,

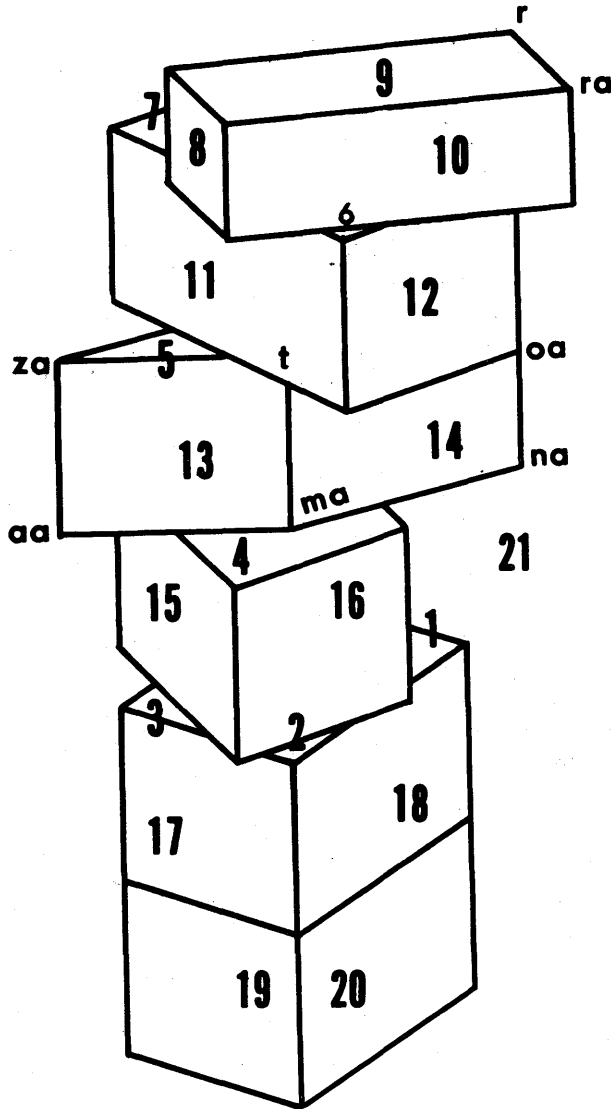


FIGURE 4—'STACK.' This scene is analyzed by SEE, with results detailed in Example 1. All bodies are correctly found. Some of the vertices appear in the drawing with their names; in other drawings we will omit the names; we are also omitting the coordinate axes.

we will give details of the program's operation. We start by loading into LISP the programs for SEE and its auxiliary functions.

Then, we evaluate:

(UREAD BRIDGE SCEN3) ↑Q

This causes scene BRIDGE (see Figure 5 'BRIDGE'), to be read (in the format described previously) and transformed to the proper form which SEE expects (also described before).

Finally, we evaluate

(SEE (QUOTE BRIDGE))

This calls SEE to work on BRIDGE.

Results appear in Table II, 'RESULTS FOR BRIDGE.'

(LOCAL ASSUMES (:18) (:19) SAME BODY)
 (LOCAL ASSUMES (:28) (:29) SAME BODY)
 (LOCAL ASSUMES (:10) (:8 :11 :5 :6 :4) SAME BODY)
 (LOCAL ASSUMES (:7) (:8 :11 :5 :6 :4 :10) SAME BODY)
 (SINGLEBODY ASSUMES (:19 :18) (:16) SAME BODY)

RESULTS

(BODY 1. IS :24 :9 :21 :12 :25)
 (BODY 2. IS :22 :26 :23)
 (BODY 3. IS :17 :3 :20)
 (BODY 4. IS :1 :2)
 (BODY 5. IS :14 :15 :13)
 (BODY 6. IS :19 :18 :16)
 (BODY 7. IS :29 :28)
 (BODY 8. IS :8 :11 :5 :6 :4 :10 :7)

Results for scene 'BRIDGE'

TABLE II—RESULTS FOR BRIDGE

SEE finds eight bodies in scene 'BRIDGE' (See Figure 5 'BRIDGE'). Unnecessary detail has been removed from the drawings; the reader must bear in mind that the complete scene contains vertices (whose names do not appear in the drawings) and their coordinates (which are also not shown), as well as edges and regions.

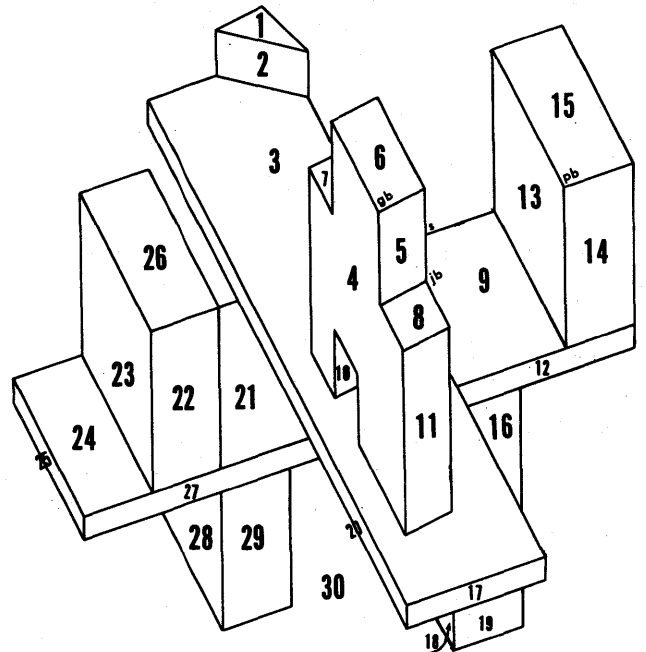


FIGURE 5—'BRIDGE.' The long body :25 :24 :27 :21 :9 :12 is correctly identified. (See Table II)

SEE and its parts

The operation of SEE is quite straightforward; the program is not recursive and does not do any tree search. Its main parts, which are executed one after another, unless otherwise indicated, are:

FILLSCENE. The properties SLOP and TYPE are generated for each vertex, if they were not already present.

CLEAN. Old properties used internally by SEE are removed from vertices and regions.

EVIDENCE. An analysis is made of vertices, regions and associated information, in search of clues that indicate that two regions form part of the same body. If evidence exists that two regions in fact belong to the same body, they are linked or marked with a "gensym" (both receive the same new label). There are two kinds of links, called strong (global) or weak (local).

LOCALEVIDENCE. Some features of the scene will weakly suggest that a group of regions should be considered together, as part of the same body. This part of the program is that which produces the 'local' links or evidence.

GLOBAL. The 'strong' links gathered so far are analyzed; regions are grouped into "nuclei" of bodies, which grow until some conditions fail to be satisfied (a detailed explanation follows later).

LOCAL. Weak evidence is taken into account for deciding which of the unsatisfactory global links should be considered satisfactory, and the corresponding nuclei of bodies are then joined to form a single and bigger nucleus. Assumptions made by LOCAL are printed (see output of SEE). LOCAL may call GLOBAL again, or go on.

SINGLEBODY. If a single region does not belong to a larger nucleus, but is linked by one strong evidence to another region, it is incorporated into the nucleus of that other region. This part of the program may call GLOBAL or LOCAL again, if necessary, or continue. SINGLEBODY also prints its assumptions.

RESULTS. The regions belonging to the background are screened out, and the results are printed.

Operation

Here is explained in considerable detail each of the parts of SEE that have been sketched above. This will help the reader understand the behavior of the program, its strength and deficiencies.

Example. Scene **'TOWER.'** First, we begin by showing a typical analysis of SEE with a somewhat complicated scene (see Figure 6 'TOWER'). Most of the scenes contain several "nasty" coincidences: a vertex of an object lies precisely on the edge of another object; two nearly parallel lines are merged into a single one, etc.

This has been done on purpose, since a non-sophisticated preprocessor will tend to make this kind of error.

The output is given in Table III, 'RESULTS FOR TOWER.'

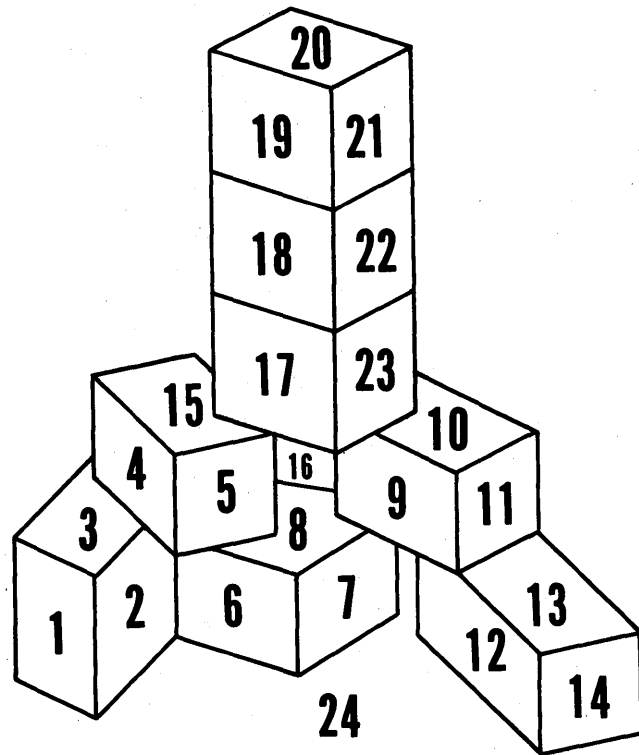


FIGURE 6—'TOWER.' Neither LOCAL nor SINGLEBODY are necessary to correctly parse this scene into the bodies which form it. There are many 'strong' links in this scene and SEE makes good use of them. (See Table III)

- (BODY 1. IS :3 :2 :1)
- (BODY 2. IS :5 :15 :4)
- (BODY 3. IS :23 :17)
- (BODY 4. IS :6 :7 :8)
- (BODY 5. IS :10 :11 :9)
- (BODY 6. IS :13 :14 :12)
- (BODY 7. IS :18 :22)
- (BODY 8. IS :20 :19 :21)

Results for TOWER

TABLE III—Results for Figure 6 'TOWER'

FILLSCENE, CLEAN. These two parts of SEE are simple; if necessary, FILLSCENE calls SLOPGENERATOR and TYPEGENERATOR; CLEAN removes some unwanted properties.

EVIDENCE. Strong or global links are placed by this part of the program. Two functions are used: EVERTICES and TJOINTS.

EVERTICES. This function considers each vertex of the scene under the following rules:

L. Vertices of the type 'L' are not considered.

FORK. If three regions meet at a vertex of the FORK type (Table IV (b)), and none of them is the background, links between them will be formed. For instance in Figure 6 'TOWER,' we establish the links :19— :20, :19— :21, and :20— :21. Nevertheless, some of these links may not be produced: in Figure 7 'FORK,' the link between :3 and :2 is *not* produced because Q is a "passing T"; the link between :1 and :3 is *not* generated because R is an 'L.' The link between :1 and :2 *is* generated.

This heuristic is powerful, and (see Figure 5 'BRIDGE') allows us, for instance, while working on vertex *jb*, to omit the link between regions :5 and :9 in scene 'BRIDGE.' Nevertheless, this same heuristic puts a link between regions :8 and :9 of the same scene. As we shall see later, this situation is not too bad.

ARROW. This type of vertex (Table IV (c)) causes two of its regions (the 'left' and the 'right' one) to be joined; in Table IV (c), we will put a link between :1 and :2, which counts as evidence that :1 and :2 belong to the same body. Nothing is said about :3.

For instance, this type of vertex joints 1: and :2 in Figure 5 'BRIDGE.'

X. Two cases are distinguished:

- (a) The X is formed by the intersection of two lines. No evidence is formed.
- (b) Otherwise (Table IV (f)), links :1— :2 and :3— :4 are formed. This type of X occurs when we have piles of objects; for instance, in Figure 6 'TOWER,' :18 and :22 are considered as belonging to the same body, due to this type of vertex.

PEAK. All its regions (table IV (h)), except the one containing the obtuse angle, are linked to each other. See also Figure 14 'CORN.'

T. A search is made for another T to match the vertex currently analyzed; two T's match if they are colinear and "facing each other;" if there are several pairs, the closest is chosen. For instance (Table IV (d)), A and

B are paired. An indicator 'NEXTE' is placed in such vertices.

- (a) Once the NEXTE is determined, EVERTICES establishes a link between :1 and :2 (Table IV (d)), and another between :3 and :4. These links will not be produced if the results of them is to associate the background with something that is not the background.
- (b) The following test is done (Figure 8 'PARALLEL'): If neither :1 or :2 is the background, but both have it as neighbor, and in some part of the boundary of :1 (and the same holds for :2) with the background, the segment them (M — K in Figure 8 'PARALLEL') is parallel to the central segment (O—P) of the T, then :1 and :2 are linked. For instance, in Figure 4 'STACK,' this analysis applied to the vertex T will produce evidence that :13 and :14 belong to the same body (since ZA—AA, T—MA and OA—NA are parallel). This is a rather global heuristic although only useful for bodies with parallel faces. Also, EVERTICES classifies T's according to the slope of their central segments.

A summary of the strong links put by EVERTICES is found in Table IV 'GLOBAL EVIDENCE.'

TJOINTS. This function actuates on T's and established global evidence as described in part (a) of T (Table IV (d)).

LOCALEVIDENCE. An arrow is a leg if one of its left or right vertices is a corner (if necessary, through a chain of matched T's) which has a side parallel to the central segment of the arrow. The function LEGS finds legs in the scene, and stores this information in a list of 'weak' links (see Figure 9 'LEGS').

GLOBAL. Strong evidence is analyzed in this part of the program. When this section is entered, several links (strong and weak) exist among the different regions of the scene. These links are shown pictorially in Figure 10 'LINKS,' for the scene 'BRIDGE' (see both). All the links to the background (:30) are deleted: the background cannot be part of any body.

Definition: a *nucleus* (of a body) is either a region or a set of nuclei which has been formed by the following rule.

TABLE IV—'GLOBAL EVIDENCE' The strong links are represented by dotted lines

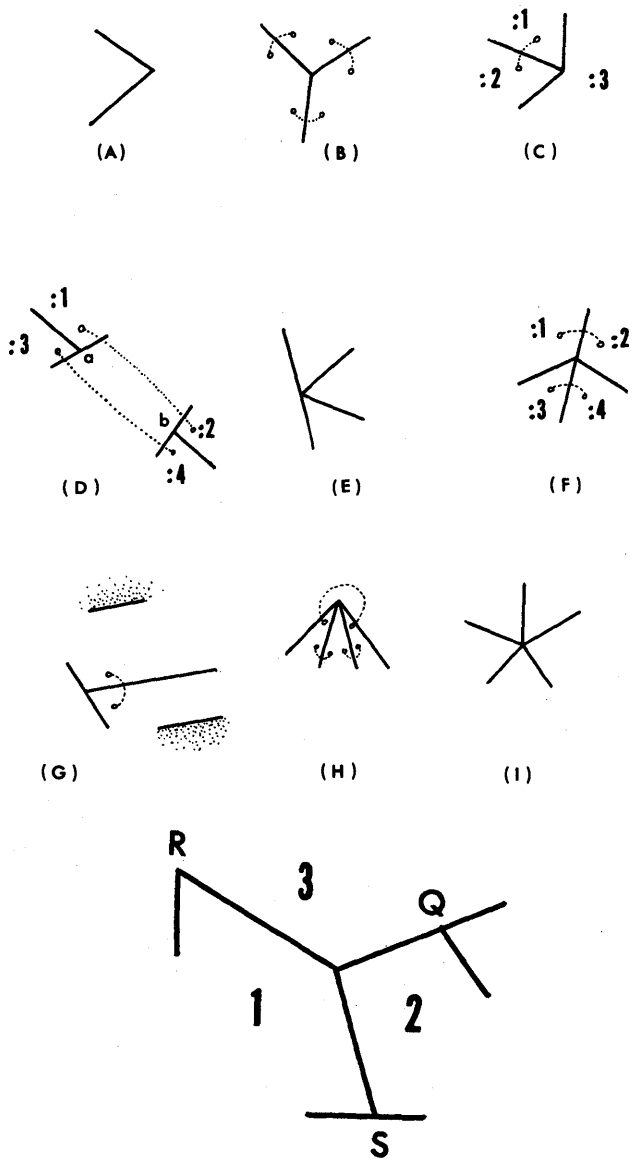


FIGURE 7—'FORK.'

Rule: If two nuclei are connected by two or more links, they are merged into a larger nucleus by concatenation.

(Two nuclei A and B are linked if regions a and b are linked where $a \in A$ and $b \in B$). For instance, regions :8 and :11 are put together, because there exists two links among them, to form the nucleus :8—11. Now, we see that region :4 (see Figure 10 'LINKS') has two links with this nucleus :8—11, and therefore the new nucleus :8—11 = 4 is formed.

We let the nuclei grow and merge under the former rule, until no new nuclei can be formed. When this is the case, the scene has been partitioned into several "maximal" nuclei; between any two of these there are zero or,

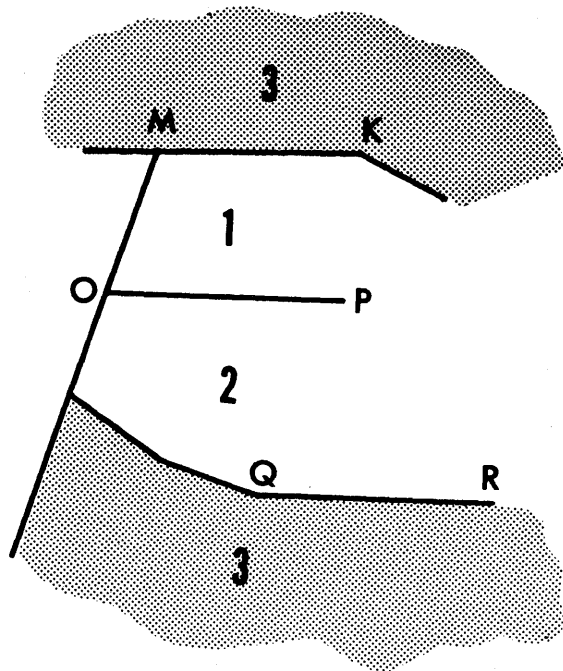


FIGURE 8—'PARALLEL.' A link is established between :1 and :2 because they do not belong to the background, but the background is a neighbor of both of them, and the segment that separates :1 from the background (and the same is true for :2) is parallel to OP, the central segment of the T in question (:3 is the background).

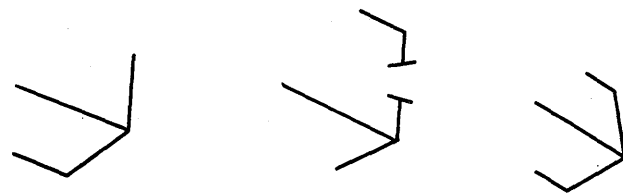


FIGURE 9—'LEGS.' Three different types of legs.

at most, one link. For example, figure 10 'LINKS' will be transformed into Figure 11 'NUCLEI.'

LOCAL. If some strong link joining two "maximal" nuclei is also reinforced by a weak link, these nuclei are merged.

For example, in scene BRIDGE (see Figure 5 'BRIDGE') the following local links exist (among others): :9 to :4, :10 to :4, :28 to :29, :18 to :19. Therefore, the corresponding nuclei are merged and now figure NUCLEI is transformed into figure 'NEW NUCLEI.'

A weak link does not cause the regions which it links to be merged or considered as belonging to the same body unless there is, in addition, one strong evidence between such regions. LOCAL may call GLOBAL again, if necessary.

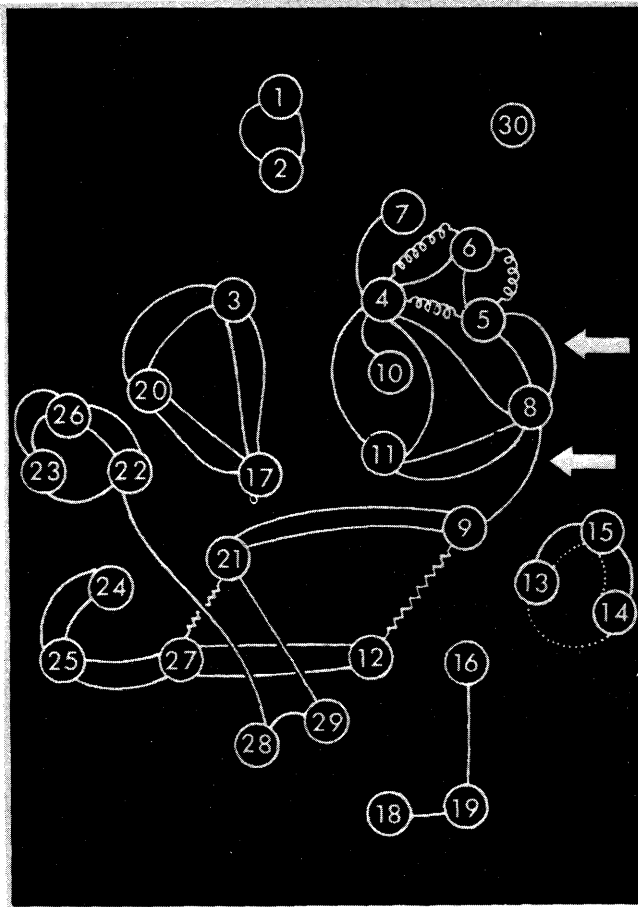


FIGURE 10—'LINKS.' This figure represents scene 'BRIDGE,' with the strong links between its regions (represented here by circles) shown. The dotted links represent the evidence generated by the vertex PB (see Figure 5 'BRIDGE'). The short arrows show the links put by vertex JB; note that a link between :5 and :9 is not put, because S (see Figure 5 'BRIDGE') is a passing t-joint. Zig-zag links are produced by the mechanism described in part (b) of T. (See Figure 8 'PARALLEL'). Curled links are produced by vertex GB; even if this vertex were occluded, and the links were missing, there is still enough evidence to identify regions :4 :5 and :6 as belonging to the same body. Weak links are not shown.

SINGLEBODY. A strong link joining a nucleus and another nucleus composed by a single region is considered enough evidence and the nuclei in question merged, if there is no other link emanating from the single region. For instance, in Figure 12 'NEW NUCLEI,' nucleus :16 is merged with nucleus :18—:19 (see Figure 13 'FINAL'). Nucleus :28 - 29 is not joined with :26—22—23 or with :24—25—27—12—21—9. Even if nucleus :28—29 were composed by a single region, it still will not be merged, since two links emerged from it: two nuclei claim its possession.

RESULTS. After having screened out the regions

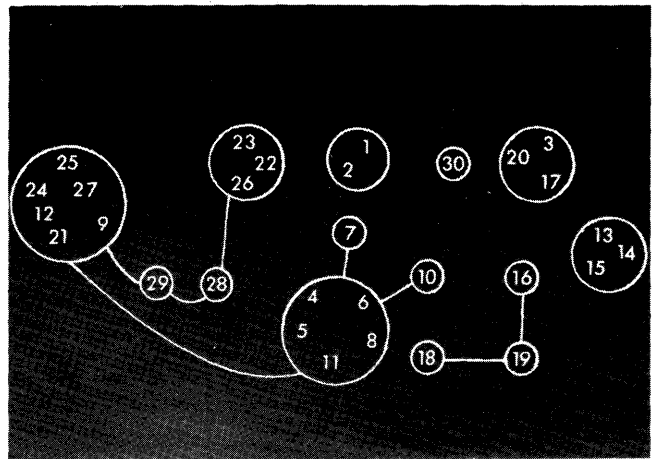


FIGURE 11—'NUCLEI.' After joining all nuclei having two or more links in common, the representation for the scene 'BRIDGE' changes from that shown in figure 10 'LINKS' to the one shown here.

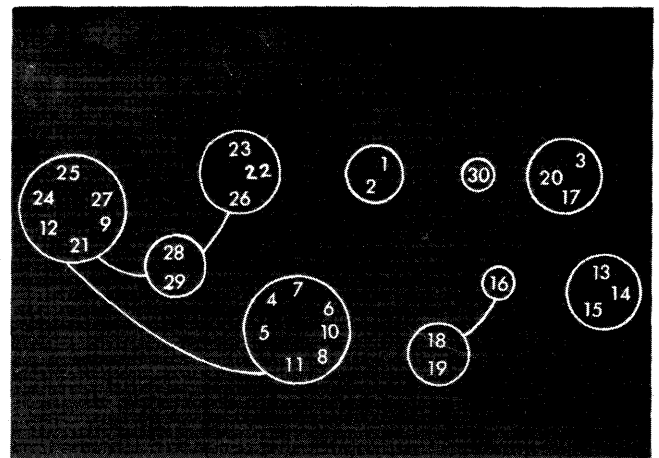


FIGURE 12—'NEW NUCLEI.' The figure shows the scene 'BRIDGE,' after LOCAL transforms it from the representation in Figure 11 'NUCLEI' to the one shown here.

that belong to the background, the nuclei are printed as "bodies."

In this process, the links which may be joining some of the nuclei are ignored: RESULTS considers the links of Figure 13 'FINAL', for instance, as non-existent.

Summary

SEE uses a variety of kinds of evidence to "link" together regions of a scene. The links in SEE are supposed to be general enough to make SEE an object-analysis system. Each link is a piece of evidence that suggests that two or more regions come from the same object, and regions that get tied together by enough evidence are considered as "nuclei" of possible objects.

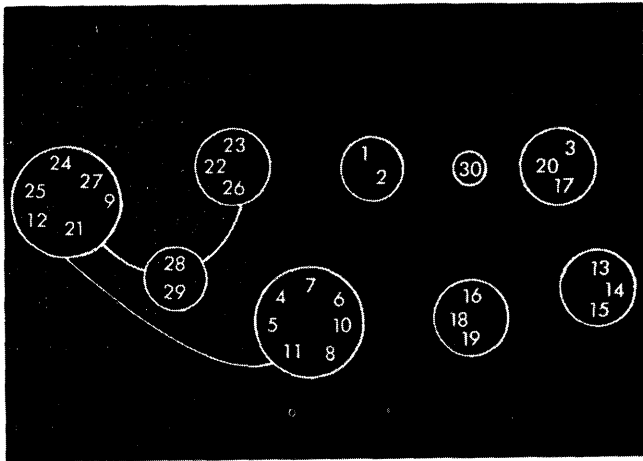


FIGURE 13—'FINAL.' SINGLEBODY joins the lonely region :16 with the nucleus :18-19.

Some interesting examples

We present in this section several scenes with the results obtained by SEE.

Example. 'CORN.' The program analyzes the scene CORN (see figure 14 'CORN') obtaining the following identification of bodies:

- (SINGLEBODY ASSUMES (:2 :3 :1) (:4) SAME BODY)
- (BODY 1. IS :2 :3 :1 :4)
- (BODY 2. IS :7 :6 :5)
- (BODY 3. IS :13 :11 :12)
- (BODY 4. IS :15 :16 :14) Results for 'CORN'
- (BODY 5. IS :19 :18 :17)
- (BODY 6. IS :21 :20)
- (BODY 7. IS :8 :9 :10)

The region :4 got a single link with :3, and SINGLEBODY had to join it with the nucleus :1 :2 :3. Note that :4 and :12 did not get joined. The pyramid at the top was easily identified because its peak produces many links.

Example. 'MOMO'. (See Figure 15 'MOMO'). The results are as follows:

- (LOCAL ASSUMES (:17) (:9) SAME BODY)
- (LOCAL ASSUMES (:9 :17) (:18) SAME BODY)
- (BODY 1. IS :3 :2 :1)
- (BODY 2. IS :32 :33 :27 :26)
- (BODY 3. IS :28 :31)
- (BODY 4. IS :19 :20 :34 :30 :29)
- (BODY 5. IS :36 :35)
- (BODY 6. IS :24 :5 :21 :4)
- (BODY 7. IS :25 :23 :22) Results for 'MOMO'
- (BODY 8. IS :14 :13 :15)

- (BODY 9. IS :10 :16 :11 :12)
- (BODY 10. IS :18 :9 :17)
- (BODY 11. IS :7 :8)
- (BODY 12. IS :38 :37 :39)

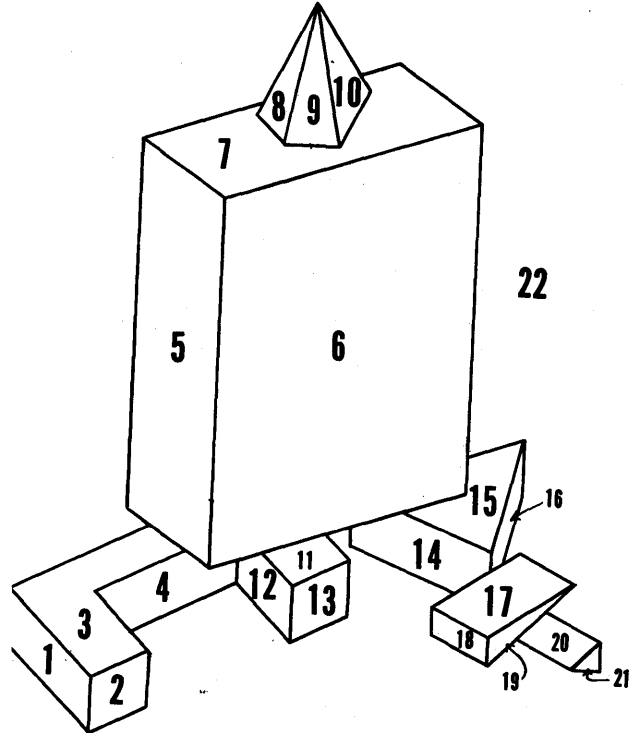


FIGURE 14—'CORN.' Since a link between :4 and :12 is not established, the bodies found in that part of the scene are :1 :2 :3 :4 and :11 :12 :13.

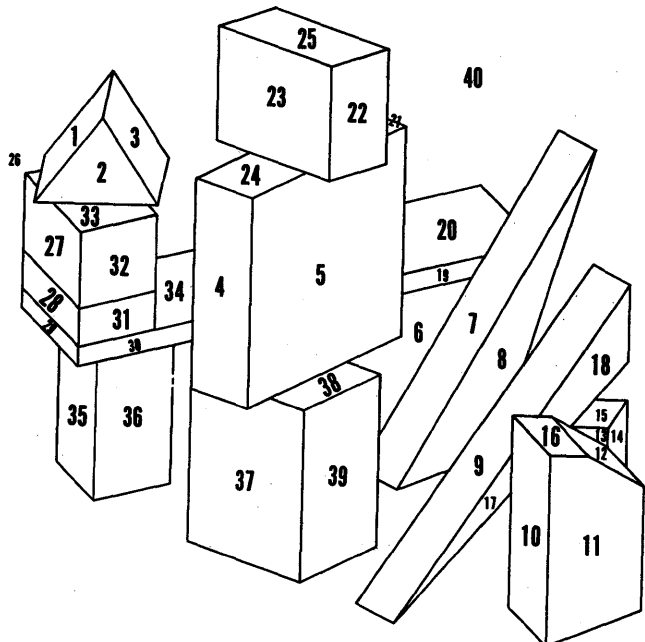


FIGURE 15—'MOMO.'

Comments on the solution for 'MOMO': The central cubes are easily isolated. :21 gets a link with :5 and another with :24, so there is no need to use LOCAL in it. The same is true of :26 with :27 and :33.

There is enough strong evidence to joint :28 and :31.

The links for :29 :30 :34 :20 :19 and :9 :17 :18 are indicated in Figure 16 'MOMO-LINKS.'

The dotted links between regions :30 and :34, and between :20 and :19 (see Figure 15 'MOMO') are due to the heuristic of parallel lines (of Figure 8 'PARALLEL'). These links made it possible to join the otherwise disconnected nuclei :29- :30- :19 and :34- :20. In particular, if :34 or :20 did not have parallel sides, SEE would have failed to merge these nuclei, and would have reported two bodies. The disposition of regions in MOMO is such that no link is present between :29 and :34, since :28 and :31 fall "exactly" parallel to :29 and :30. In a less extreme scene, some of these links would be present, allowing correct identification even if :29-:30-:34-:20-:19 were not a prism. Anyway, SEE did not make any mistakes.

The triangle of links formed by :9, :18 and :17 normally would have produced 3 separate bodies (since we need at least one double link to merge two regions); nevertheless, in this case, LOCAL makes some assumptions and saves the situation. Again, if :9 were not a parallelogram, there would be no weak links between :9 and :18 or between :9 and :17, and 3 bodies would have been reported instead of :9-:17-:18. But we should notice that, in general, two links instead of one would be between :17 and :18.

Links were established between :12 and :13, without serious consequences, because we need two mistakes, two wrong strong links, to fool the program. But that could happen.

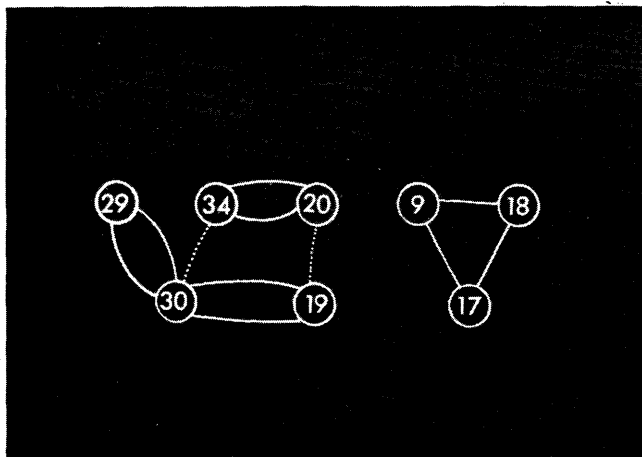


FIGURE 16—MOMO—LINKS'. Some links of figure 15 'MOMO'

Example. 'HOME'.—(see scene HOME). The results are in Table V 'RESULTS FOR HOME.'

```
(SINGLEBODY ASSUMES (:38) (:39) SAME BODY)
(SINGLEBODY ASSUMES (:34) (:33) SAME BODY)
(SINGLEBODY ASSUMES (:25 :24 :22) (:23) SAME BODY)
(SINGLEBODY ASSUMES (:20) (:21) SAME BODY)
RESULTS
(BODY 1. IS :3 :6 :10 :2)
(BODY 2. IS :14 :13 :11 :12)
(BODY 3. IS :9 :8)
(BODY 4. IS :18)
(BODY 5. IS :15 :5)
(BODY 6. IS :38 :39)
(BODY 7. IS :7 :26 :27)
(BODY 8. IS :20 :21)
(BODY 9. IS :29 :28)
(BODY 10. IS :34 :33)
(BODY 11. IS :31 :32 :30)
(BODY 12. IS :25 :24 :22 :23)
(BODY 13. IS :16)
(BODY 14. IS :19)
(BODY 15. IS :17)
(BODY 16. IS :36 :37 :35)
```

Results for HOME

TABLE V—Results for HOME

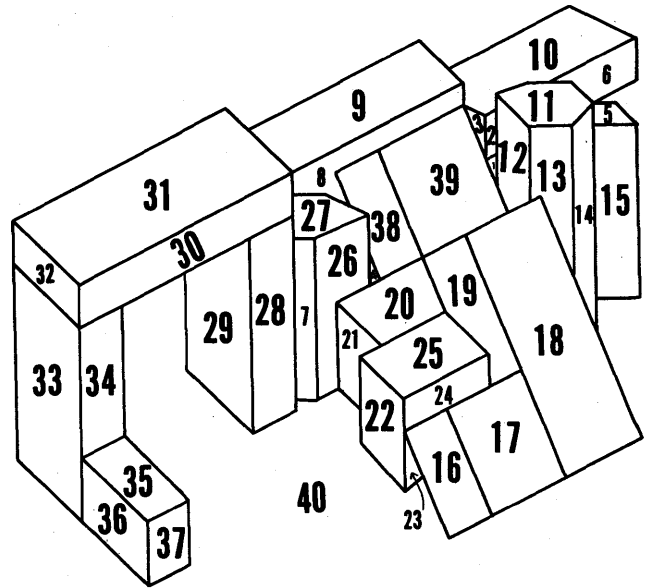


FIGURE 17—'HOME.' The hexagonal prisms did not create difficulties; SINGLEBODY was needed to group :34 with :33. The body :38-39 was identified as a single one, but :16-17 was reported as two. Note that there does not exist local link between :21 and :20; nevertheless, SINGLEBODY makes the correct identification. (See Table V).

Comments on the solution to HOME: There is a certain degree of ambiguity in this scene which human beings tend to resolve by assuming parallelepipeds. :16 and :17

are reported as two separate bodies, instead of one, which is probably a more natural answer.

In the picture from which 'HOME' was drawn, :19 and :18 were the two faces of a single parallelepiped leaning on :38-:39; SEE reports :19 as one body and :18 as another.

Nevertheless, SEE reports :38 and :39 as forming part of the same body (which was in fact the case in the picture in question), due to the fact that :4 and :1 are background.

Example. 'SPREAD.'—(see Figure 18 'SPREAD'). The results are in Table VI 'RESULTS FOR SPREAD'.

(LOCAL ASSUMES (:36) (:4) SAME BODY)
 (LOCAL ASSUMES (:30) (:31 :32 :29) SAME BODY)
 (LOCAL ASSUMES (:16) (:17) SAME BODY)
 (LOCAL ASSUMES (:5) (:20) SAME BODY)
 (LOCAL ASSUMES (:3 :11 :9) (:12 :10) SAME BODY)
 (SINGLEBODY ASSUMES (:23) (:22) SAME BODY)
 (SINGLEBODY ASSUMES (:4 :36) (:37) SAME BODY)
 (SINGLEBODY ASSUMES (:28 :26 :27) (:25) SAME BODY)

RESULTS

(BODY 1. IS :39 :40 :38)
 (BODY 2. IS :23 :22)
 (BODY 3. IS :42 :41)
 (BODY 4. IS :4 :36 :37)
 (BODY 5. IS :24)
 (BODY 6. IS :28 :26 :27 :25)
 (BODY 7. IS :31 :32 :29 :30)
 (BODY 8. IS :20 :5)
 (BODY 9. IS :12 :10 :3 :11 :9)
 (BODY 10. IS :13 :7 :1)
 (BODY 11. IS :21 :6)
 (BODY 12. IS :8 :18)
 (BODY 13. IS :17 :16)
 (BODY 14. IS :45 :43 :44)
 (BODY 15. IS :19)
 (BODY 16. IS :15 :14)

Results for SPREAD

TABLE VI—Results for SPREAD

Comments on the solution to SPREAD: The body :22-23-24, due to insufficiency of links, was split in two: :22-23 and :24.

Since there is only one link between :6 and :5, this body gets split into two: :6-21 and :5-20. Note that :21 is not the same face as :20, and there is where SEE gets confused and refuses to see evidence toward linking :21 with :20.

The long body :9-10-11-12-3 gets properly identified. Example. 'HARD.'—This scene is analyzed with the following results:

(LOCAL ASSUMES (:11) (:12) SAME BODY)
 (LOCAL ASSUMES (:15) (:16) SAME BODY)
 RESULTS
 (BODY 1. IS :12 :11)

(BODY 2. IS :16 :15)
 (BODY 3. IS :32 :31 :30)
 (BODY 4. IS :9 :10 :8)
 (BODY 5. IS :18 :19 :20)
 (BODY 6. IS :13 :17 :14) Results for 'HARD'
 (BODY 7. IS :5 :4)
 (BODY 8. IS :1 :2 :33)
 (BODY 9. IS :24 :23 :22 :3 :21 :28 :29)
 (BODY 10. IS :25 :26 :27)
 (BODY 11. IS :7)
 (BODY 12. IS :6)

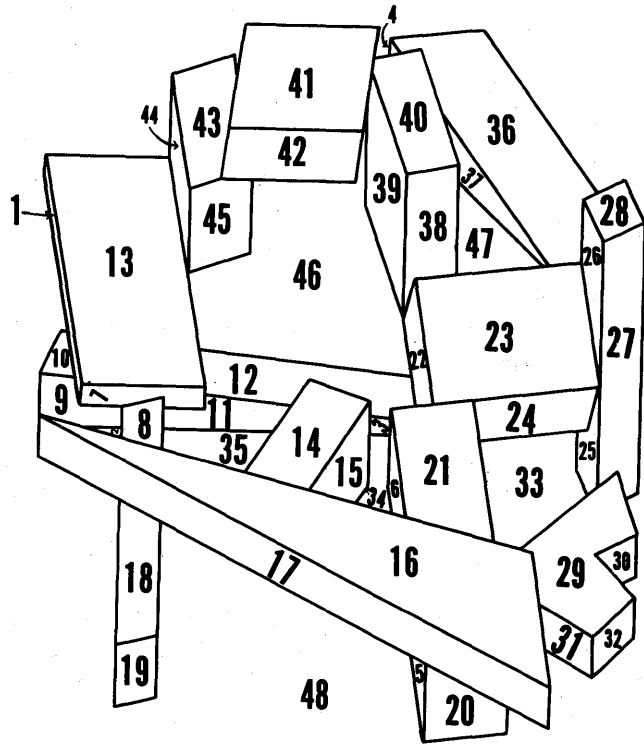


FIGURE 18—'SPREAD.' :41 and :42 are identified as a single body. Nevertheless, :8-18-19 gets broken into :8-18 and :19. :28-27-26-25 gets correctly identified, as well as the funny looking :29-30-31-32. (See Table VI).

Comments on the solution to HARD: :15 and :16 have to make use of weak evidence to get joined and recognized as forming part of the same body. Nevertheless, this is not necessary with :28 and :29, because, through a chain of Ts, there is enough evidence in :3, :21 and :22 to join successfully all that long and twice occluded body.

There is one serious bug in this identification: regions :7 and :6 get identified as two separate bodies, and not as a single one, as one would normally do. This is caused by the fact that neither :7 nor :6 have visible 'useful' vertices, and there are not enough parallel lines in them to use the heuristic of Figure 8 'PARALLEL.'

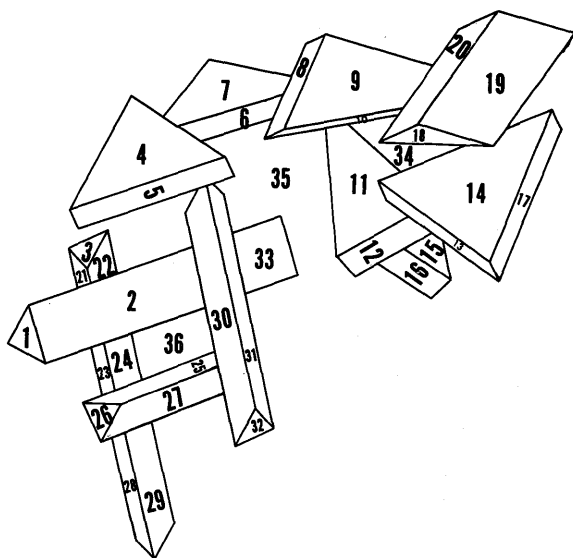


FIGURE 19—'HARD.' Body :21-22-3-23-24-28-29 is reported as a single object, which is correct. Nevertheless, regions :7 and :6 get reported as two bodies.

:33 was recognized as part of :1-2, as it should be.

DISCUSSION

We have described a program that analyzes a three-dimensional scene (presented in the form of a line drawing) and splits it into "objects" on the basis of pure form. If we consider a scene as a set of regions (surfaces), then SEE partitions the set into appropriate subsets, each subset forming a three-dimensional body or object.

The performance of SEE shows to us that *it is possible to separate a scene into the objects forming it, without needing to know in detail these objects*; SEE does not need to know the 'definitions' or descriptions of a pyramid, or a pentagonal prism, in order to isolate these objects in a scene containing them, even in the case where they are partially occluded.

The basic idea behind SEE is to make global use of information collected locally at each vertex: this information is noisy and SEE has ways to combine many different kinds of unreliable evidence to make fairly reliable global judgments.

The essentials are:

- (1) Representation as vertices (with coordinates), lines and regions
- (2) Types of vertices.
- (3) Concepts of links (strong and weak), nuclei and rules for forming them.

The current version of SEE is restricted to scenes pre-

sented in symbolic form. It does not have resources for dealing with curves, shadows, noise, missing lines, etc. So it represents a "geometric theory of object identity" at present.

Since SEE requires two strong evidences to join two nuclei, it appears that its judgments will lie in the 'safe' side, that is, SEE will almost never join two regions that belong to different bodies. From the analysis of scenes shown above, its errors are almost always of the same type: regions that should be joined are left separated. We could say that SEE behaves "conservatively," especially in the presence of ambiguities.

Divisions of the evidence into two types, strong and weak, results in a good compromise. The weak evidence is considered to favor linking the regions, but this evidence is used only to reinforce evidence from more reliable clues. Indeed, the weak links that give extra weight to nearly parallel lines are a concession to object-recognition, in the sense of letting the analysis system exploit the fact that rectangular objects are common enough in the real world to warrant special attention.

Most of the ideas in SEE will work on curves too. However, we do not yet have a good idea of how sensitive the system will be to "symbolic noise," i.e., missing misplaced, and false region boundaries. As indicated in the scenes above, the system seems to have good resistance to "accidents" in which objects happen to to "line up" unusually well. This feature may be necessary if the preprocessor that (eventually) will feed data SEE decides to report two nearly colinear lines as one alone, or if it lumps several vertices into one, because they lie close to each other.

Extensions. (None of this incorporated in the actual program.) More heuristics could be added to increase the number of links; in particular, given a good number of "link proposers," parameters set outside (by the user) would tell SEE which set of heuristics to use; for instance, if we knew that the scene is formed by prisms, we could use the heuristics that ask for parallel lines having a given configuration, we could check the length of certain edges, etc.

Different kinds of links could also be established; in this way, 'contradictory' links (such as the three links of Figure 13 'FINAL' which SEE just ignores) could be studied further, in order to discover ambiguities. In particular, a "conditional link" would be useful: regions :2 and :3 belong to the same body if region :4 does not.

SINGLEBODY could be improved so as to analyze in more detail the regions that by themselves form bodies (that is, the bodies formed by only one region); in this way, we could hope to joint regions :6 and :7 of scene 'HARD.'

ACKNOWLEDGMENTS

The author is grateful to Professors Marvin Minsky and Seymour Papert for several helpful discussions, and to the referees of this paper for their constructive recommendations.

Michael Speciner made valuable suggestions; Cornelia Sullivan helped to codify most of the scenes.

"The author was partially supported by a scholarship from the Instituto Nacional de la Investigación Científica (México)."

The work reported here could not have been produced had the author not had access to the computational facilities of Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01). This paper, having had this support from MAC, may be reproduced in whole or in part for any purpose of the United States Government.

REFERENCES

- 1 R GREENBLATT J HOLLOWAY
Sides 21
Memorandum MAC-M-320 A I Memo 101 Project MAC
MIT August 1966 A program that finds lines using gradient measurements
- 2 K K PINGLE
A program to find objects in a picture
Memorandum No 39 Artificial Intelligence Project Stanford
University January 1966 It traces around objects in a picture
and fits curves to the edges
- 3 A GUZMÁN
Scene analysis using the concept of model
Technical Report No AFCRL-67-0133 Computer Corpora-
tion of America Cambridge Massachusetts January 1967
- 4 A GUZMÁN
A primitive recognizer of figures in a scene
Memorandum MAC-M-342 AI Memo 119 Project MAC
MIT January 1967
- 5 A GUZMÁN
Some aspects of pattern recognition by computer
MS Thesis Electrical Engineering Department MIT February
1967 Also available as a Project MAC Technical Report
MAC-TR-37 This memorandum discusses many methods
of identifying objects of known forms.
- 6 A GUZMÁN H V McINTOSH
CONVERT
Communications of the ACM
9,8 August 1966 pp 604-615 Also available as Project MAC
Memorandum MAC-M-316 AI Memo 99 June 1966
- 7 A GUZMÁN
Decomposition of a visual scene into bodies
Memorandum MAC-M-357 AI Memo 139 Project MAC
MIT September 1967 unpublished This memorandum is a
more technical description of SEE
- 8 R H CANADAY
The description of overlapping figures
MS Thesis Electrical Engineering Department MIT January
1962
- 9 L G ROBERTS
Machine perception of three-dimensional solids
Optical and electrooptical information processing
pp 159-197 J T Tippett et al eds MIT Press 1965
- 10 M L MINSKY S A PAPERT
Research on intelligent automata
Status report II
Project MAC MIT September 1967
- 11 J McCARTHY
Plans for the Stanford artificial intelligence project
Stanford University April 1965
- 12 C A ROSEN N J NILSSON eds
Application of intelligent automata to reconnaissance
Third Interim Report Stanford Research Institute December
1967

A limited speech recognition system*

by DANIEL G. BOBROW and DENNIS H. KLATT†

Bolt Beranek and Newman Inc
Cambridge, Massachusetts

INTRODUCTION

A computer system which identifies words in continuous speech of an unknown speaker is beyond the current state of the art in speech recognition. LISPER, is a successful *limited speech recognition* system based on a set of assumptions which greatly simplify the recognition problem; within these restrictions it allows experimentation on the usefulness of a voice insertion system on the computer.

LISPER operates within limitations along a number of dimensions. Rather than use continuous speech in which segmentation is a problem, we work with messages with easily delimited beginning and termination points. The set of messages is limited in number; at any one time the vocabulary to be distinguished can contain up to about 100 items. However, an item need not be a single word, but may be any short phrase. A message list from a NASA mission context, shown in Table 1, was one of three used in testing the system. Note that LISPER recognizes each of these messages as a unit, and does *not* segment a multiword utterance into individual words for recognition. The system is not designed to work well simultaneously for a number of different speakers, or achieve good recognition scores for an unknown speaker. The system is useable by any male speaker, but must be first trained by him. The training period consists of a period of closed loop operation in which the speaker says an input message, the system guesses what he says, and he responds with the correct message. In this training phase, the system will learn the idiosyncratic variations of the speaker's set of input messages. In this closed loop system, it is not unlikely that the speaker will also learn something.

*This research was supported principally by the National Aeronautics and Space Administration, and in part by the Advanced Research Projects Agency.

†Also at Massachusetts Institute of Technology, Cambridge Mass.

The LISPER system was designed as a research vehicle, as well as a pattern recognition system. For this

one	distance to dock
two	fuel tank content
three	time to sunrise
four	time to sunset
five	orbit apogee
six	orbit perigee
eight	revolution time
nine	closing rate to dock
point	midcourse correction time
plus	micrometeoroid density
stop	radiation count
zero	what is attitude
seven	remaining control pulses
minus	alternate splashdown point
pressure	weather at splashdown point
negative	sea and wind at splashdown point
what is yaw	visibility at splashdown point
what is pitch	temperature at splashdown point
end repeat	skin temperature
affirmative	power consumption
inclination	fuel cell capacity
distance to earth	repeat at intervals

TABLE 1—Message list from NASA context

reason, flexibility for data access and program modification were important. For this reason, LISPER was built in an extended LISP system.^{1,2} This decision allowed an easy transfer during the research from a DEC PDP-1 to an SDS 940 computer.

Any pattern recognition system must have three basic components: preprocessing hardware to extract a representation of the input; programs utilizing this raw data to compute properties of the unknown input (data reduction); and a recognition or decision algorithm. Figure 1 shows a block diagram of the organization of the LISPER system. The input speech signal may be obtained from either a microphone or tape recorder. The

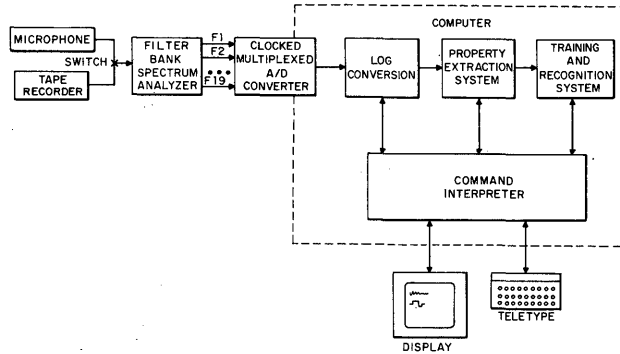


FIGURE 1—Block Diagram of the Lisper system

basic parameters of the input are extracted by a spectrum analyzer which consists of a pre-emphasis network followed by nineteen bandpass filters. Bandpass filter outputs are rectified, low-pass filtered, sampled, converted into logarithmic units, and stored on digital tape, or in the memory of the computer. The 19 filters consist of 15 filter spaced uniformly up to 3000 Hz, the range encompassed by the first three resonances of the male voice; and 4 filters covering the range from 3000–6500 Hz in which there is information about the noise component of speech. The four-pole Lerner bandpass filters each have a bandwidth of 360 Hz. An advantage of this filter system is that one does not see spectral peaks due to individual harmonics of the fundamental frequency of a male speaker; a spectral maximum in the filter outputs indicates the presence of one or more formants (resonances of the vocal tract transfer function). The spacing between filters makes resolution of individual formants poor, and we do not use formant tracking in our recognition system.

The spectral representation of a word as derived from our input system consists of 200 spectral samples, corresponding to 2 seconds of speech material and approximates the log of the short term power spectrum of the speech signal. A spectral sample consists of the outputs of the 19 bandpass filters, all sampled at a particular instant of time. A filter output, in log units, is an integer ranging from 0 to 63, covering a 45 decibel range of intensity.

The use of the logarithm of the short-time spectrum is well established as one approach to speech analysis and has often been used as the basis for recognition programs.³ The principal contributions of this research have been the development of a set of interesting algorithms for extracting features which characterize speech utterances, and coupling this set with a recognition algorithm capable of high quality message identification in the presence of redundant, inconsistent, or incorrect information from these properties. The recognition algorithm does not directly utilize the

spectral data, nor is an intermediate phonemic transcription of the word produced. However, we have made use of the present day body of knowledge about acoustic phonetics, and the distinctive feature approach to phonological description.^{4,5,6} We discuss below some problems of these two approaches and indicate how a number of these problems have been alleviated by techniques used in LISPER.

Our system has been extensively tested, and recognition scores have been obtained on three vocabularies with three different speakers. Typical of our results is attainment of recognition scores that approach 97% correct on a 54 word vocabulary after three rounds of training for a single speaker. These results compare favorably with the word recognition scores of other investigators.^{7,8,9,10}

Property extraction system

An unknown word is represented in the computer by a matrix of integers (filter number vs. sampled time vs. log-amplitude). The purpose of a property or feature extraction system is to reduce the information content of the input signal to a level that makes it possible for the decision or recognition algorithm to operate reasonably. The critical thing is that the information eliminated should be information which is irrelevant to the decision that the recognition algorithm must make.

The techniques used in LISPER can best be appreciated by first exploring some problems involved in identifying a speech utterance on the basis of the 19×200 array representing a 2 second sampling of its energy spectrum. Pattern recognition schemes operating on data of this dimensionality are theoretically tenable (Sebestyen,¹¹), but only if repetitions of words cluster properly in the resultant vector space. It is by now well-known that speech patterns as initially stored in a computer do not cluster according to the word spoken. Some of the reasons for this are:

1. The range of intensities encountered will vary because the overall recording level is not fixed. Recording level depends on vocal effort and the distance of a speaker from the microphone.
2. An unknown word is difficult to register at the same place within 200 spectral samples because word onset time is not a simple feature to detect reliably. For example, initial voiceless fricatives may be missed, prevoiced stops are hard to treat consistently, etc. We insure that all of an utterance is contained within the two second interval by sampling the input line continuously, storing spectral samples in a computer-simulated digital tape loop, and stopping 1.8 seconds after the data first exceed a threshold.

3. The total duration of a word is highly variable. In addition an increase in speaking rate is not manifested by a linear compression of the time dimension. Final syllables are often prolonged. Some transitions (for example, the release of a stop consonant) are not as greatly affected by changes in speaking rate as the steady-state portions of vowels. If shortened enough, vowels are likely to be reduced and consonants may not be carefully articulated, with resultant losses in spectral distinctiveness.
4. A speaker attempts to generate an utterance so that it has a particular set of perceptual attributes (or features). We do not know in detail what the acoustic correlates of these attributes are. There is a great deal of variation allowed in the acoustic properties of the signal that will still give rise to the same utterance. There is also sufficient redundancy in the message to permit a speaker to leave out certain attributes entirely. For example, the degree of stress placed on a syllable will determine the extent to which the vowel *may* be reduced (Lindblom, ¹²). Consonants in unstressed syllables may contain less frication noise, a weak stop burst release, or incomplete stop closure. Vowels may be nasalized in nasal environments. The substitution of one incomplete gesture for a consonant cluster is also common in unstressed syllables of natural speech. None of these effects would necessarily produce word recognition difficulties if they appeared consistently in the data. Unfortunately, they do not.
5. If a speaker is instructed to speak distinctly and not rapidly, some surprising and unfortunate variability in speaking habits has been experimentally detected. In an attempt to help the system, our speakers released final stops, increased the length of some syllables, and articulated unstressed syllables more carefully than they would normally. Unfortunately, our speakers appear to have found these speaking habits unnatural, and could not remember from repetition to repetition exactly what they had done to "help." For example, final voiced stop releases gave trouble by producing short vowel segments that varied greatly in amplitude, and the words *four* and *core* were sometimes pronounced as if they had two syllables.
6. Individual speakers have vocal tracts of different sizes and shapes. It is physically impossible for two speakers to produce identical spectra for a given phone or word. A speaker makes an articulatory gesture that we, as listeners, interpret; possibly with respect to our knowledge of the

spectra that he is capable of producing (Gerstman, ¹³). The nature and importance of the normalization process of a listener are not well understood.

7. Different speakers have articulatory habits (idiolects) that may be quite distinct. Habits include the timing and dynamics of articulatory movements and the features that a particular speaker employs to manifest a phonemic distinction. Whether recognition difficulties can be attributed to individual speech habit structures is not known. Very little quantitative data are available on the characteristics that distinguish speakers.

The variability of the spectrum of a word has led to the search for data reduction techniques to eliminate irrelevant information. An approach favored by several investigators has been to develop rules for detecting phonemes (Fry and Denes, ¹⁴; Martin et al., ¹⁵; Reddy, ¹⁶). We believe that phoneme recognition is a much more difficult problem than word recognition because it presupposes a good understanding of the cues that distinguish phonemes in arbitrary phonetic environments. For example, allophones of the phoneme /p/ may be:

1. normally aspirated, as in the word "peak" [p^hik]
2. weakly aspirated, as in the word "supper" [s]∧'p^hə^∧]
3. non-aspirated, as in the word "spin" [spin]
4. non-released, as in the word "top" [t^hap⁷]

There is no need for a word recognition program to attempt to group this disparate set of physical signals together into one phoneme. However, to the extent that algorithms for deriving a detailed phonetic-feature description of an utterance can be found, they can be of considerable help to a practical speech recognition system. Examples of feature approaches that are relevant include the work of Hughes, ¹⁷; Hemdal and Hughes, ⁶; Gold, ⁷; and Martin et al. ¹⁵.

The properties of speech which are used by the recognition program are based on the energy measures derived from the input system. The output of each filter is an elementary function of the speech input signal. We use the notation $F_n(i)$ for the output of filter n at sample interval i ; that is $F_1(i)$, $F_2(i)$, . . . , $F_{19}(i)$ are used for the output of filters 1 through 19 at sample interval i . The filter number n ranges from 1 for the low frequency filter to 19 for the high frequency filter; and $i = 1$ for the first sample interval to $i = 200$ for the last time sample of a two second utterance.

More complicated functions of the speech input signal can then be defined in terms of these elementary (base) functions, F_1 , . . . F_{19} , in the LISP system. For

example, the following function has been found to correlate with perceived loudness, independent of vowel quality.

$$\text{Loud}(i) = F1(i) + F2(i) + F3(i) + F4(i) + F12(i) - F7(i)$$

The output of the seventh filter is subtracted from the sum of the first four filters and filter 12 to compensate for the fact that low vowels are inherently more intense when produced with the same vocal effort. We will indicate later how a modified version of this function Loud(i) can be used to correct for differences in recording level between repetitions of the same word.

Loud(i) is useful in reducing the information that must be processed by the decision algorithm, since it helps to normalize the input with respect to a variable which is not important to recognition, namely, the recording level of the input signal. We describe in this section a number of techniques which are used to reduce the information content of the incoming signal in ways that preserve the invariance of message identity over ranges of parameters which are irrelevant to the decision about the identity of the message.

One important way we have found to reduce the information content of a function is to reduce its range of values. Thus, for example, we may define a reduced information property Amp2(i) based only on the output of filter F2.

$$\text{Amp2}(i) = \begin{array}{l} +1 \text{ if } F(2) > 40; \\ 0 \text{ if } F(2) > 20; \\ -1 \text{ otherwise} \end{array}$$

Amp2(i) is thus a 3 valued function, where we ascribe no significance to the names of the three values. As described later, a set of non-linguistic threshold properties similar to Amp2 can be used as a basis for recognition. However, we note the following problem. If the signal in F2 were varying around either of the thresholds, 20 or 40, then there would be little significance of the change of state from -1 to 0 or 0 to 1 and back; it would be much more likely due to noise than to a real variation in the input signal.

To make such properties less sensitive to noise, we introduce a hysteresis region around the thresholds to insure that a change of state is significant. A revised definition of Amp2(i) is

$$\text{Amp2}(i) = \begin{array}{l} +1 \text{ if } [F2(i) > 40] \text{ or } [F2(i) > 37 \text{ and } \\ F2(i) - 1] = 1]; \\ 0 \text{ if } [F2(i) > 20] \text{ or } [F2(i) > 17 \text{ and } \\ F2(i) - 1] \neq -1]; \\ -1 \text{ otherwise} \end{array}$$

Most of the features we use are functions of sampled time having a range limited to a small number of distinguishable states. These functions are very sensitive to slight changes in time scale and origin; these variations in the data are irrelevant for recognition.

The time dimension is removed from a feature by transforming the time function into a sequence of transitions of states, as is illustrated in the following example:

i	1	2	3	4	5	6	7	8	9	...
Voice(i)	0	0	1	1	1	1	1	1	0	...
Voice =	0	1	0							...

This transformation reduces the amount of data that must be manipulated by the program. Due to the nature of spoken language, the exact time when features change value will vary from repetition to repetition of the same word, but the essence of the word remains in the sequence of state transitions of an appropriate set of features.

No information about the word onset time, speaking rate, and speaking rhythm can be recovered from the sequence unless these parameters have an effect on the actual states that are reached. To this extent, recognition will be unperturbed by variations in word onset time, speaking rate, and speaking rhythm.

A problem that arises from collapsing the time dimension is an inability to tell whether two features were in specific states at the same time. Time removal assumes that features independently characterize a word. This is obviously false. A clear example is provided by the words "sue, zoo" which can only be distinguished by knowing whether the strident is simultaneously voiced or not. It is possible to retain some timing information by the inclusion of features that count the number of time samples between temporal landmarks in the data and change state if the count exceeds a threshold, or to include states which are entered only upon simultaneous satisfaction of two conditions.

For example, the state "-1" corresponds to voiceless segments in the definition of the spectral quality [r]-like(i). In this way we indicate in which voiced segment an [r]-like phone occurred. The sequence produced for the word "ratio" should be -1, 1, 0, -1, 0, -1. A more detailed localization of the [r] is not necessary for limited vocabulary word recognition, and would certainly be more difficult. We find that very gross timing information is satisfactory for recognition. Two features are used to make a preliminary division of a word or message into one or several segments. One feature divides a word into voiced and voiceless segments and the other divides it into syllables. Voice(i)

and Syllable(i) are then used to introduce time markers in the definitions of other features.

Features which classify the vowel quality of the stressed syllable of an utterance into one of 10 categories have been found to be very useful in recognition. The stressed syllable is identified with the aid of a function Loud(i) that computes an approximation to the perceived loudness of a vowel. These features of a stressed syllable are less likely to be affected by the natural variability that characterizes the speech process (Stevens¹⁸).

Recognition algorithm

The recognition algorithm is a program that learns to identify words by associating the outputs of various property extractors with them. During learning, the vocabulary of words is presented a number of times, and information is accumulated about the different ways that the speaker may pronounce each word. For example, the result of the training procedure applied to the feature sequence Voice after 5 presentations of the 4-word vocabulary "one, two, subtract, multiply" might be:

Word	Sequence	Number of times sequence occurred
one	010	5
two	010	5
subtract	01010	5
multiply	0101010	3
multiply	01010	2

The two versions of the word "multiply" exemplify a common problem. No matter where we place threshold boundaries, there are some words that are treated inconsistently by the feature detectors.

However, our recognition algorithm is powerful enough to deal with ambiguity in the voicing assignment of some vocabulary items, and thus we need not develop a more sophisticated voicing algorithm. We made no attempt to use rate of voicing onset, offset, time between burst and energy build-up for stops, etc., as additional cues to a better feature definition. In all of the features used, the ambiguity remains at a level tolerable to the recognition algorithm. The redundancy inherent in the complete set of feature definitions allows recognition of variations of the same message. Another way of putting it is to say that there appear to be no absolute boundaries along the property dimensions we have chosen. The recognition algorithm takes this fundamental limitation into account and makes a best guess given the imperfect nature of the properties.

In the training process the program reorganizes the data for each property into a list of those sequences

which have occurred. The list for Voice for our example is shown below:

Sequence	(Word, frequency)
010	one 5, two 5
01010	subtract 5, multiply 2
0101010	multiply 3

If a new utterance is presented to the program for recognition, and the feature Voice(i) produces the sequence 01010, then Voice will register one vote for the word "subtract" and one vote for the word "multiply." The unknown word is identified as the vocabulary word eliciting votes from the most features. Typically, the identified word receives a vote, signifying a perfect match for that property, from only about 80 percent of the features.

In case of ties, the program makes use of information concerning the number of times a word appears at a node. Thus, if "subtract" and "multiply" tie for first place in the voting from all the features and Voice = 01010, then Voice will register 5 votes for "subtract" and 2 votes for "multiply" in the run-off between these two candidates.

The final decision procedure is an attempt to find the message from the set of possible inputs which is most similar to the current input message. Because there is a wide variation in the way people say things, the decision procedure does not insist that the current input must be like one of the prototype input strings in all ways that it was categorized; that is, it need not be suggested by all properties. In this sense, the decision procedure allows a generalization of the original training and learning by looking for a best fit without putting a bound on the goodness of this fit.

The property sequences have been considered as independent characterizations of the input message. In this case, by independent we do not mean that the computations themselves are necessarily independent, but that these descriptors of the input message are treated independently in the decision process. With the assumption of independence, the voting procedure is similar to a maximum likelihood estimate. The a priori probabilities of all messages are the same. Therefore, the most likely message is the one for which the product of the a posteriori probabilities (or the sum of their logarithms) is a maximum. For each property, every message, M_i , previously associated with the current input sequences S_i is given a (scaled) log probability of $100 + N_i$, where $N_i > 1$ is the frequency with which M_i was seen for S_i . Messages not associated with S_i in training are given log-probability 0 and can therefore be ignored in the summation. These assignments of

probabilities achieve in one step a score which allows a single search to determine the most likely message.

It has been experimentally determined that the voting scheme works as well or better than a number of other measures that make use of the same information. For example, we tried the adjusted weight used by Teitelman¹⁹ in ARGUS, a hand printed character recognizer (a similarly structured pattern recognition system); and $\log(1 + N_i)$ which is a good small sample probability estimate of the a posteriori probability of message i .

Functions and properties used in recognition

The principal results of this research have been the development of properties which characterize speech input signals in a way to make recognition possible. Two different sets of recognition properties are included here. The first is a set of properties and functions which tend to describe the speech signal in more linguistic terms. Transitions of these properties describe features of the input message which can be understood in terms of the ordinary linguistic descriptions of such messages. In this way, they tend to be more speaker independent than the second, more abstract set of properties which are described.

The second set of properties extract features of the spectral shape of a message. Their extreme simplicity makes them seem ideal for hardware implementation, and their high accuracy in recognition suggests that they capture most of the invariant qualities of our input message set.

Linguistic features .

We distinguish *functions*, such as Loud(i), which have a domain equal to the input spectral representation of an unknown word; *features*, such as Voice(i), which are functions that are used in recognition and have a range limited to a small number of states; and *feature sequences*, such as Voice, which are the result of removing the time dimension from a feature. The following functions are used in the definitions of a set of 15 features.

Functions

$$1. \text{ Loud}(i) = F1(i) + F2(i) + F3(i) + \text{MAX}[F2(i), F4(i)] + \text{MAX}[2(F3(i) - F4(i)), 0].$$

This function is intended to provide a measure of the perceptual loudness of vowels. Perceptual loudness is related to vocal effort whereas the acoustic energy in a vowel depends in part on the vocal tract configuration as well as vocal effort. For example, the low vowel, [a], for which the vocal tract is relatively open will have a greater natural intensity than the high

vowel, [i], for which the vocal tract is more closed (Lehiste and Peterson, 1959; Fant, 1960). The function is intended to compensate for the reduced level associated with a low first formant.

$$2. \text{ ah}(i) = F3(i) + F4(i) - F1(i) - F2(i).$$

This function and the following two functions are used to characterize the spectral quality of the vowel nuclei of a word. The vowels [i, a, u] represent limiting articulatory positions for the tongue body. The vowel [a], as in "pot," is produced by placing the tongue as far back and as low as possible without producing a constricted vocal tract (and therefore a consonant). An acoustic correlate of this articulation is a high first formant, resulting in a greater output in filters 3 and 4 than in filters 1 and 2. The function ah(i) will therefore be a maximum for vowels with a high first formant.

$$3. \text{ ee}(i) = F9(i) + F10(i) + F11(i) + F12(i) - 2F6(i) - 2F7(i).$$

The vowel [i] as in "beet" is produced by positioning the tongue body as high and as forward in the mouth as possible. An acoustic correlate of this articulation is a low first formant and a high second and third formant, resulting in an energy minimum in filters 6 and 7, and an energy concentration between filters 9 and 12. The function ee(i) will therefore be a maximum for vowels similar to [i].

$$4. \text{ oo}(i) = F3(i) + F4(i) + F5(i) - F8(i) - F10(i) - F12(i) - F14(i) - F16(i).$$

The vowel [u] as in "boot" is produced by positioning the tongue body as high and as far back in the mouth as possible. An acoustic correlate of this articulation is a low first and second formant, resulting in a major energy concentration below F7 and reduced energy above F7. The function oo(i) will therefore be a maximum for vowels similar to [u].

$$5. \text{ er}(i) = F7(i) + F8(i) - F4(i) - F12(i).$$

This function is similar to the vowel functions in form, but has the task of detecting spectra characteristic of the consonantal and syllabic allophones of [r]. The low third formant of [r] and [é] (as the vowel in "Bert") produces a distinct spectral shape with an energy concentration centered in filters 7 and 8, a dip in

energy at filter 4 between the first and second formants, and an absence of energy above F10 due to the low third formant. The function $er(i)$ will therefore be a maximum when phones similar to [r, ê] are produced.

$$6. \text{str}(i) = F16(i) + F17(i) + F18(i) + F19(i).$$

This function is a maximum when high frequency frication energy is present. The strident phones produce intense high frequency energy, whereas a non-strident fricative produces a small energy peak in filter 19. The function $\text{str}(i)$ will therefore be a maximum for many allophones of [s, z, š, ž, č, j, t, k], which are the first consonants in [sin, zen, shin, azure, chin, gin, tin, kin] respectively.

$$7. \text{dspect}(i) = |ah(i) - ah(i - 1)| + |ee(i) - ee(i)| \\ + |oo(i) - oo(i - 1)|$$

This function is a computationally inexpensive approximation to a spectral derivative. The function will be a maximum when the spectrum is changing rapidly, as for example in a consonantal transition. $\text{Dspect}(i)$ is used as an aid in delimiting syllable boundaries. It is also used to distinguish ee-like vowels from some consonant-vowel transitions that produce a momentary peak in the function $ee(i)$.

$$8. \text{cv}(i) = \text{sum of } F1(i) \text{ through } F16(i).$$

This function tends to be greater for vowels than for adjacent consonants. This is because a constricted or closed vocal tract configuration results in a reduced acoustic output in the frequency range spanned by these filters. This function is used to detect syllable boundaries.

$$9. \text{damp}(i) = \text{sum of } F1(i) \text{ through } F19(i) \text{ minus sum} \\ \text{of } F1(i - 1) \text{ through } F19(i - 1).$$

This function indicates a sudden increase in energy in all filters. It will be a maximum for stop releases and sudden voicing onsets. It is used to detect stop bursts.

Features

The preceding functions are used in the computation of the following features. The usual interpretation of the three feature values (states) is: “-1” implies that the property is irrelevant for this time interval; “0” means that the property is relevant but not present; and “1” means that the property is relevant and present.

$$1. \text{Voice}(i) = 1 \text{ if } [F2(i) > t1] \text{ or } [F2(i) > t1-5 \text{ and}$$

$$\text{Voice}(i-1) = 1]$$

0 otherwise

$$t1 = 34$$

A hysteresis region of 5 units is employed about the threshold, $t1$, to ensure that a change of state is significant and not due to random fluctuations in level. The effect of the second conditional part of the definition is to keep $\text{Voice}(i)$ in its current state for values of $F2(i)$ between 30 and 34, the hysteresis region.

An unknown word may vary considerably in overall recording level if the speaker moves with respect to the microphone or changes his vocal effort. It is much easier to compensate for recording level by modifying the threshold, $t1$, than by attempting to modify the raw data. The maximum value attained by the time function $\text{Loud}(i)$ is used for this normalization:

$$\text{Maxloud} = \text{Max}_i [\text{Loud}(i)]$$

$\text{Loud}(i)$ attains its maximum value near the midpoint of the stressed vowel nucleus. For a speaker positioned correctly in front of a microphone and speaking at a comfortable level, a typical value for Maxloud is 200. Threshold normalization is accomplished by computing a new threshold, $t1'$, for each unknown word according to the following formulae:

$$t'_1 = t1 * (200/\text{Maxloud})$$

The method fails if the recording level increases to saturation or decreases below background noise. The effective range for recording level insensitivity is somewhat less than the full 45 decibel range of the input system.

The fundamental voice frequency of a male speaker will from about 90 to 180 Hz. The greatest energy due to voicing will appear in the low frequency filters because energy in the harmonics of the laryngeal source falls off at about 12 db per octave. $\text{Voice}(i)$ looks at the energy in filter 2. If this energy exceeds a threshold, voicing is present, otherwise not. This simple definition has worked surprisingly well. The greatest difficulty has been in treating final voiced stop releases consistently, and in detecting voicing energy in some voiced fricatives.

$$2. \text{Syllable}(i) = 1 \text{ when the function } \text{cv}(i) \text{ is signifi-} \\ \text{cantly increasing}$$

0 when the function $\text{cv}(i)$ is signifi-
cantly decreasing

otherwise, set to same value as in
previous time sample.

The precise computation that implements this definition is too lengthy to be given here. The feature usually gives an accurate count of the number of syllables in a word and provides a rough segmentation. However, the times at which the feature changes state do not delimit precise syllable boundaries. Examples of difficulties with Syllable(i) include a frequently missed third syllable in "binary" and the segmentation into two syllables of some repetitions of the words "four" and "core."

3. $\text{Stress}(i) = 1$ if $\text{Loud}(i) = \text{Maxloud}$ for the first time.
 0 if $\text{Syllable}(i) = 1$
 -1 otherwise
 $\text{Maxloud} = \text{Max}_i [\text{Loud}(i)]$

This feature is designed to indicate which syllable of a word is the one which has the maximum loudness and is therefore the stressed syllable. Stress assignment has worked very well. The only word for which stress assignment has not consistently agreed with the expected stress assignment is "overflow," and this may be due to variations in the actual stress given the word by our speakers.

4. $[\text{a}]\text{-like}(i) = -1$ if $\text{Voice}(i) = 0$
 1 if $[\text{ah}(i) > 1]$ or $[\text{ah}(i) > -2]$ and
 $[\text{a}]\text{-like}(i-1) = 1$
 0 otherwise

This feature indicates the presence of the vowels [a, o^u, ɔ, ae, ʌ] in many phonetic environments (the vowels in [pot, boat, bought, bat, but] respectively).

5. $[\text{i}]\text{-like}(i) = -1$ if $\text{Voice}(i) = 0$
 1 if $[\text{ee}(i) > 16]$ or $[\text{[i]}\text{-like}(i-1) = 1]$
and $[\text{ee}(i) > 8]$ and $[\text{F19}(i) < 25]$
and $[\text{Dspect}(i) < 30]$
 0 otherwise

This feature indicates the presence of the vowels [i, I, e^v, e] (the vowels in [beat, bit, bait, bet]) in many phonetic environments. The two additional clauses involving $\text{F19}(i)$ and $\text{Dspect}(i)$ are utilized to eliminate false responses to stridents and consonantal transitions which are momentarily [i]-like.

6. $[\text{u}]\text{-like}(i) = -1$ if $[\text{Voice}(i) = 0]$
 1 if $[\text{oo}(i) > \text{t2}]$ or $[\text{[u]}\text{-like}(i-1) = 1]$ and $[\text{oo}(i) > \text{t2}-7]$
and $[\text{F2}(i) > \text{F1}(i)]$
 0 otherwise
 $\text{t2} = 32$

This feature indicates the presence of the vowels [u, v, o^u, ɔ] (the vowels in [boot, book, boat, bought]) in many phonetic environments. The additional clause involving $\text{F1}(i)$ and $\text{F2}(i)$ is intended to eliminate responses to consonants such as [m, n, l, w]. It has been found necessary to ignore any transitions to the "1" state in the features [i]-like(i) and [u]-like(i) if they last less than 6 time samples. Even with this additional constraint, a number of consonants are assigned one of these vowel-like qualities. In addition, there appear to be no natural threshold boundaries in the vowel-quality space so that a certain percentage of the vocabulary are treated inconsistently by each feature.

7. $[\text{r}]\text{-like}(i) = -1$ if $[\text{Voice}(i) = 0]$
 1 if $[\text{[er}(i) > 18]$ and
 $[\text{F3}(i) - \text{F4}(i) > 5]$
or $[\text{[er}(i) > 14]$
and $[\text{[r]}\text{-like}(i-1) = 1]$
 0 otherwise

This feature indicates the presence of [r, ê] in many phonetic environments. The clause involving $\text{F3}(i)$ and $\text{F4}(i)$ is intended to ensure that the first formant frequency is not high. The low third formant of these phones produces a spectrum that is distinct and relatively easily differentiated from all other speech spectra. The feature works consistently except following voiceless stops and in some unstressed intervocalic positions.

8. $\text{Strident}(i) = 1$ when the function $\text{Str}(i)$ is significantly increasing
 0 when the function $\text{Str}(i)$ is significantly decreasing
 -1 otherwise set to same value as in previous time sample

This feature indicates the presence of [s, z, š, ž,] and also [t, d, k, g] in certain phonetic environments (generalal in stressed position and followed by a front vowel). A floating threshold is employed to divide a sequence of two successive stridents (e.g., [st]) into two strident segments separated by the state "-1."

$$9. \text{ Strid-stop}(i) = \begin{cases} 0 & \text{if } \text{Lstrident}(i) = 1 \\ 1 & \text{if } \text{Strident}(i) = 1 \\ -1 & \text{otherwise} \end{cases}$$

Where $\text{Lstrident}(i)$ is the same as $\text{Strident}(i)$ except that the state "+1" must occur for more than 6 time samples or it is transformed to the state -1. This property computes the length of a strident segment and places it in one of two categories. In general, a long strident is a continuant and a short strident is a stop. However, in some phonetic environments, voiceless stops may have an aspiration duration that is longer than the frication duration of an intervocalic fricative.

$$10. \text{ Fricative}(i) = \begin{cases} 1 & \text{if } [[\text{F19}(i) > t4] \text{ or } [[\text{F19}(i) > \\ & t4 - 4] \text{ and } [\text{Fricative}(i-1) \\ & = 1]] \text{ and } [\text{Loud}(i) < 120]] \\ 0 & \text{if } \text{Voice}(i) = -0 \\ -1 & \text{otherwise} \\ & t4 = 6 \end{cases}$$

This feature looks for the high frequency energy characteristic of [f, v, θ, ð], (the initial consonants of [fin, vend, thin, then]). Strident phones generally exceed the threshold and are also detected by $\text{Fricative}(i)$. The upper bound on $\text{Loud}(i)$ is used to prevent a response to a loud vocalic sound in which energy is spread over the entire spectrum. A feature like $\text{Fricative}(i)$ or $\text{Strident}(i)$ sends a great deal of information about a word to the recognition algorithm, but neither feature is as consistent in its analysis of an unknown word as a simple feature such as $\text{Voice}(i)$.

$$11. \text{ Stop-burst}(i) = \begin{cases} 1 & \text{if } [[\text{damp}(i) > 75] \\ & \text{and } [\text{F19}(i-1) < 30] \\ & \text{and } [[\text{F2}(i) < 20] \\ & \text{or } [\text{F2}(i) - \text{F2}(i-1) < 1]]] \\ 0 & \text{if } \text{Voice}(i) = 1 \\ -1 & \text{otherwise} \end{cases}$$

This feature detects the sudden onset of energy in all filters that is characteristic of the release of a stop. The stop burst is distinguished from a rapid voicing onset by requiring the energy in $\text{F2}(i)$ to either be low or not increasing. The "-1" state following a burst is removed if it is less than 3 time samples long. This transformation is an only partially successful attempt to signal voiced

stops by the sequence "1, 0" and voiceless stops by the sequence "1, -1, 0."

$$12. \text{ Nasal}(i) = \begin{cases} -1 & \text{if } \text{Voice}(i-2) = 0 \\ & 1 \text{ if } [\text{Loud}(i) - \text{Loud}(i-1) > 5 \\ & \text{or } [\text{Nasal}(i-1) = 1 \\ & \text{and } \text{Loud}(i) - \text{Loud}(i-1) > 1]] \\ & \text{and } [\text{ah}(i) - \text{ah}(i-1) > 5 \\ & \text{or } \text{ah}(i) - \text{ah}(i-2) > 8] \\ 0 & \text{otherwise} \end{cases}$$

This feature looks for a voiced segment followed by a sudden increase in loudness and a rising first formant frequency. A nasal followed by [i] does not satisfy this criterion, but nasals in many other environments are detected. Initial [ɛ] is frequently also detected by the feature. Features that look for dynamic properties of the input spectra are not easy to define efficiently. This relatively simple algorithm does not give a very satisfactory definition of nasality.

13-15. There are three other features of a somewhat different type. These features are intended to give a fairly precise characterization of the stressed vowel in the unknown word. The features, called [a]-stress, [i]-stress, and [u]-stress respectively, characterize the spectrum at a single time sample, that point during the stressed vowel when $\text{Loud}(i)$ reaches a maximum for the word.

The ranges of the functions $\text{ah}(i)$, $\text{ee}(i)$, and $\text{oo}(i)$ have been divided into 10 regions corresponding to the 10 possible states of the three special features:

UPPER BOUND ON REGION

State:	1	2	3	4	5	6	7	8	9	10
ah(i)	none	14	9	4	0	-3	-6	-11	-16	-27
ee(i)	none	80	60	40	20	0	-20	-40	-60	-80
oo(i)	none	80	60	40	20	0	-20	-40	-60	-80

A typical stressed [i] phone might be characterized by the states [a]-stress = 9, [i]-stress = 1, and [u]-stress = 6. It has been found that this characterization is relatively stable for a given word. Most words fall into two or at most three states for each of the features.

Non-Linguistic Properties

A set of abstract properties was defined near the end of the research project in order to provide a comparative base for the performance of the linguistically motivated

features just described. These properties take advantage of some of the constraints that apply to speech spectra, but do not reflect the detailed phonetic content of the utterances.

There are two distinct types of spectral shape properties that were used in this set. The first nineteen properties crudely categorize the shape over time of the output of each of the nineteen filters, individually, with no attention paid to correlation between filters. There are three regions of "interest" in the output. The spectral property is given the value "-1" if the output is in the lowest region, "0" in the middle region, and "1" in the highest region. A hysteresis region around the transition threshold is used to prevent many "uninteresting" transitions when the filter output is near the border of these regions.

Formally, we define a set of 19 spectral amplitude properties, $S_n(i)$, as follows, where the notation $F_n(i)$ denotes the output of filter n at the i^{th} sample interval:

$$S_n(i) = \begin{array}{l} 1 \text{ if } [F_n(i) > B] \text{ or } [[F_n(i) > B - 8] \text{ and} \\ \quad [S_n(i - 1) = 1]] \\ 0 \text{ if } [F_n(i) > A] \text{ or } [[F_n(i) > A - 8] \text{ and} \\ \quad [S_n(i - 1) \neq -1]] \\ -1 \text{ otherwise} \end{array}$$

A and B, as a function of the filter number, are given in the following table:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	37	34	30	25	20	30	25	20	15	25	21	18	15	20	15	17	20	17	13
B	50	47	44	40	35	45	40	35	30	38	34	31	27	35	30	30	30	25	30

The threshold values in this table should be modified if there is a change in recording level. This can be done in the same way as previously described for thresholds involving linguistic features.

Another ten properties are used which roughly describe the correlation between adjacent filters for the lower ten filters. These properties categorize the differences between energies in adjacent filters. The three alternatives are: the next higher band has significantly more energy, significantly less energy, or about the same energy, as its lower neighbor. These properties are given values "1," "-1," and "0" respectively in these cases. Formally, we define a set of 10 spectral difference properties, $D_n(i)$, as follows, where $n = 1, 2, \dots, 10$ and $m = n + 1$:

$$D_n(i) = \begin{array}{l} 1 \text{ if } [[F_m(i) - F_n(i) > 4] \\ \quad \text{or } [[F_m(i) - F_n(i) > -1] \\ \quad \text{and } [D_n(i - 1) = 1]] \end{array}$$

$$\begin{array}{l} -1 \text{ if } [[F_m(i) - F_n(i) < 4] \\ \quad \text{or } [[F_m(i) - F_n(i) < 1] \\ \quad \text{and } [D_n(i - 1) = -1]] \\ 0 \text{ otherwise} \end{array}$$

A change of state for one of these properties indicates that the spectral shape of the current time sample differs from the spectral shape of the previous time sample. A change of state is not produced by a simple change in level since it is the difference in energies which is significant. The large hysteresis region around the thresholds also prevents a property from changing state when there is no energy present in either filter.

The thresholds for the two types of properties were selected so that each state could be reached by at least some of the vocabulary items. The 29 properties produce sequences of states used in recognition. We can compare directly these recognition scores with those produced by state sequences from the 15 linguistic features.

Results

As benchmark experimental data, we recorded a word list, Table 2, used by Ben Gold in other speech recognition experiments. This list was recorded in a quiet room (S/N ratio > 35 db) as spoken 10 times by two speakers, KS and CW. Both had been used as subjects by Gold. KS and CW were also recorded 5 times each on two other lists, an augmented International Word Spelling alphabet, Table 3, and a message list typical of one that might be used in a NASA mission context, shown earlier in Table 1.

The messages were recorded on high quality magnetic tape at $7\frac{1}{2}$ ips, with approximately two seconds gap between words. For almost all experiments we used a digital tape containing the 6 bit logarithms of the filter bank output sampled every 10 milliseconds. Only in the noise degradation experiment was the analog input used directly. No difficulty was encountered in finding the beginning and end of each message.

A training round consists of one repetition of the entire word list in which the computer makes an identification attempt, is told the correct response, and stores the characterization of this message for use in future identification. With four rounds of training utilizing the linguistic properties, the system recognized correctly (gave as unambiguous first choice) 51 and 52 out of 54 words of the Ben Gold list for KS and CW respectively. Early experiments indicated that additional training would not increase the recognition scores after reaching its asymptote, and these 95 + % correct scores were available after only 3 rounds of training. Further experiments gave speaker KS 38 out of 38

insert	name
delete	end
replace	scale
move	cycle
read	skip
binary	jump
save	address
core	overflow
directive	point
list	control
load	register
store	word
add	exchange
subtract	input
zero	output
one	make
two	intersect
three	compare
four	accumulate
five	memory
six	bite
seven	quarter
eight	half
nine	whole
multiply	unite
divide	decimal
number	octal

TABLE 2—Word list from Ben Gold

zero	hotel
one	India
two	Juliet
three	kilo
four	Lima
five	Mike
six	November
seven	Oscar
eight	papa
niner	Quebec
affirmative	Romeo
negative	Sierra
alpha	tango
bravo	uniform
Charlie	Victor
delta	whiskey
echo	x-ray
foxtrot	yankee
golf	zulu

TABLE 3—International word spelling alphabet and numbers

correct on the International Word Spelling Alphabet and 43 out of 44 correct on the NASA message set, indicating the usefulness of these linguistic properties on other message sets. Recognition rates for the Gold list as a function of training rounds are shown in Table 5. Note the lack of improvement (and oscillation of scores) with more than 4 rounds of training. Part of the

difficulty is that these later rounds were recorded at a different time. This indicates that some degradation in performance may be expected under less well controlled conditions.

Some incorrect responses are listed in Table 6 (for KS on the Gold list after 4 training rounds). These confusions give an indication of the types of phonetic information that is not well represented by the linguistic features. For example, the “four-core” confusion suggests that frication noise often falls below the lower thresholds of the filters, and that formant transitions are not sufficiently distinct to trigger different state transitions for this word pair. It is expected that some improvements could be made in the threshold settings by studying the data of Table 6.

Previous Training	1	2	3	4	5	6	7	8	9	Perfect
Ben Gold List	43	44	49	51	47	45	52	49	51	54
Alphabet	30	36	34	38						38
NASA List	35	37	43	43						44

TABLE 5—Typical recognition scores (KS)

Correct Answer	Guess
Weather at Splashdown Point	Alternate Splashdown Point
Four	Core, or Whole
Binary	Memory
Load	One

TABLE 6—Typical mistakes

These asymptotic recognition scores also indicate the separability of this 54 word vocabulary with these linguistic properties. The three vocabularies were combined into a single vocabulary of 114 distinct messages (duplications between vocabularies were considered only one message). On this larger vocabulary, the recognition score was a remarkable 110 out of 114 for KS. All those messages identified correctly in the individual lists were also identified in this larger context after the same four training rounds. This indicates very little interference between messages on these different lists; and implies that fairly large noninterfering (phonetically balanced) vocabularies might be constructed while maintaining this high recognition rate. It should be emphasized that while messages were repeated in training and testing, the same speech utterance was never seen more than once by the system.

To test the consistency of speakers uttering messages, we devised the spectral threshold properties described earlier. We tested these properties for recognition only

on the Ben Gold word list. After four rounds of training, the system was able to identify correctly 52 out of the 54 input messages presented to it for KS; and 51 out of 54 for CW. This rate is as good as that discussed for the linguistic features.

The spectral threshold properties appear to converge more rapidly to a higher recognition score on early training rounds. However, the linguistic properties may be slightly less speaker dependent. Neither of these tendencies is statistically significant so we are unable to make a definitive comparison.

There are 29 spectral threshold properties and only 15 linguistic properties. In addition, the spectral threshold properties tend to produce longer sequences. The comparable recognition rates of the two property sets suggests that the total information sent to the recognition algorithm by each set is roughly the same order of magnitude. If this is true, then it can be concluded that the linguistic sequences embody a more concise and more consistent representation of the vocabulary.

DISCUSSION

We have outlined the reasons why word recognition based on spectral input data is not a straightforward problem in pattern recognition. The invariants in the speech code appear to be relatively complex functions of the spectral input. We have also outlined the reasons why phoneme recognition is not an appropriate intermediate step in a word recognition algorithm. Unsolved problems include the segmentation of speech spectra into phoneme sized chunks and the derivation of rules for recognizing all of the allophones of a given phoneme. Our approach is intermediate between pattern recognition and phoneme recognition.

A set of acoustic feature detectors have been defined. The time dimension is removed from each feature so that the learning program deals only with the sequences of states produced by the feature definitions. The feature sequence characterization of an input message is sufficiently redundant so that the recognition algorithm is able to deal successfully with errors and inconsistencies in some fraction of the features.

The advantages of this approach are:

- (1) No precise segmentation of the utterance is required; features can change state in arbitrary time segments because the time dimension is removed from a feature in a way that is independent of the changes of state of other features.
- (2) A word need not be registered at exactly the same place in the 2 second speech sample; state sequences are independent of time origin.
- (3) Feature definitions can be made insensitive to the recording level by adjusting threshold

settings according to a measure of the loudness of the unknown word.

- (4) Feature definitions can be made less sensitive to noise by employing hysteresis regions about thresholds.
- (5) State transition sequences weight changes in spectra as the important cues to word recognition (simple pattern-matching schemes will give undue weight to a sustained vowel).
- (6) Features may be defined to take advantage of natural acoustic boundaries between phones (Stevens, 1968b) and thereby minimize the variability in state transition sequences produced by a feature.
- (7) Features may be added to the system to provide redundancy for those decisions that are difficult. There is a direct control over the amount of redundancy sent to the recognition algorithm.
- (8) The feature approach permits the introduction and testing of linguistic hypotheses, such as placing greater emphasis on the properties of the stressed syllable.

The disadvantages of this approach are:

- (1) Removal of the time dimension discards all information concerning the simultaneous occurrence of specific states for two or more features. We have used the voicing feature to reintroduce some timing information. There exist pairs of English words for which this "time with respect to the voiced segments" is not enough to disambiguate the pair. Special features can always be defined to treat special cases, but our approach is not easily generalized to yield segmentation of an utterance into phone-sized chunks.
- (2) The features currently implemented are not speaker independent. Each speaker will have to train the system and this requires approximately 3 or 4 repetitions of the vocabulary.
- (3) Our system will degrade in performance as the length of the vocabulary is increased or as the number of speakers that it can simultaneously recognize is increased. This property is of course true of any recognition program; however, it should be noted that, with our current simple-minded set of features, there is a high error rate in any feature characterization and we rely heavily on redundancy to select the most likely input message.
- (4) For our limited objectives, the current implementation is computationally fast and gives satisfactory results. However, more sophisticated and more reliable features would be desirable.

The exponent in the function relating computation time and feature performance is not known but may be restrictively large.

A set of simple sum-and-difference properties was compared with the linguistically motivated features in the later stages of research. The comparable performance of this set of properties has led to several conclusions concerning the desirable attributes of features that operate within this framework.

A good feature includes a maximum amount of information in the sequences that it produces. Factors that influence information content are consistency of characterization and the number of vocabulary items that can be differentiated on the basis of the feature. A voicing feature that works perfectly does not contain as much information about the unknown word as any of the relatively inconsistent spectral properties. In other words, a moderate amount of inconsistency is tolerable if accompanied by increased word separability (and additional features to provide redundancy).

The spectral properties compare favorably on the basis of recognition scores because the linguistic features are fewer in number and computationally similar in form to the spectral properties. We believe that, even for limited vocabulary word recognition, a set of phonetically oriented features exist which are better in some sense than simple pattern features. The reasons for this faith consist of arguments that:

- (1) there exist natural acoustic boundaries between phones (Stevens, ²⁰). Thresholds placed at these boundaries will produce features with more consistent sequence assignments. Features selected on this basis will be less sensitive to the free variations that occur in speech spectra.
- (2) there exist invariant attributes in acoustic waveforms from different speakers. These invariants, when incorporated into feature definitions, will produce recognition scores that are less sensitive to the individual speaker.

The arguments in favor of phonetically oriented features are offset by the computational simplicity of the spectral properties. Not enough is currently known in acoustic phonetics to take advantage of these theoretical benefits without additional basic research. We argue that a carefully selected set of properties like our spectral properties represent a practical word recognition solution that may not be superseded for some time to come.

The so-called spectral shape features were not selected primarily on the basis of their pattern recognition potential. They were carefully selected on the basis of what is known about the information bearing parameters of speech. An arbitrary set of parameters

will not work (for the reasons outlined in the introduction). In that sense, the spectral shape features resemble acoustic phonetic parameters and are distinct from a number of other types of pattern characterizing functions that could have been chosen. Our results should not be interpreted to mean that a fundamental knowledge of speech is not needed when working on restricted problems such as limited vocabulary word recognition. On the contrary, it was only after developing a set of linguistic feature definitions within the context of our chosen recognition algorithm that we were able to devise a set of spectral shape functions possessing the necessary attributes.

ACKNOWLEDGMENTS

The authors wish to acknowledge the invaluable assistance of Lucille Darley who did most of the programming of LISPER, and Kenneth N. Stevens and Gottfried von Bismark, ²¹ who designed and built the filter bank spectrum analyzer.

REFERENCES

- 1 D G BOBROW D L MURPHY W TEITELMAN
The BBN LISP system
BBN Report No 1677 April 1968
- 2 J MCCARTHY et al
The LISP 1.5 programmers manual
MIT Press 1964
- 3 J L FLANAGAN
Speech analysis synthesis and perception
New York Academic Press 1965
- 4 N CHOMSKY M HALLE
The sound pattern of English
New York Harper and Row 1968
- 5 R JAKOBSON G M FANT M HALLE
Preliminaries to speech analysis
The MIT Press Cambridge 1963
- 6 J HEMDAL G HUGHES
A feature based computer recognition program for the modeling of vowel perception
In W Wath-enDunned Models for the Perception of Speech and Visual Form pp 440-453 Cambridge The MIT Press 1964
- 7 B GOLD
Word recognition computer program
Technical Report 452 Research Laboratory of Electronics
Cambridge MIT Press 1966
- 8 P DENES M MATHEWS
Spoken digit recognition using time-frequency pattern matching
The Journal of the Acoustical Society of America 32 pp 1450-1455 1960
- 9 H DUDLEY S BALASHEK
Automatic recognition of phoentic patterns in speech
The Journal of the Acoustical Society of America 30 pp 721-732 1958
- 10 K H DAVIS H R BIDDOLPH S BALASHEK
Automatic recognition of spoken digits
The Journal of the Acoustical Society of America 24 pp 637-642 1952

- 11 G S SEBESTYEN
Recognition of membership in classes
IRE Transactions on Information Theory IT-6 pp 44-50 1961
- 12 B LINDBLOM
Spectrographic study of vowel reduction
The Journal of the Acoustical Society of America 35 pp 1773-1781 1963
- 13 L J GERSTMAN
Classification of self-normalized vowels
1967 Conference on Speech Communication and Processing
6-8 November Office of Aerospace Research United States
Air Force pp 97-100 1967
- 14 D B FRY P DENES
The solution of some fundamental problems in mechanical speech recognition
Languages and Speech 1 pp 35-58 1958
- 15 T B MARTIN A NELSON H ZADELL R COX
Continuous speech by feature abstraction
DDC No AFAL-78-66-189 Camden N J RCA 1966
- 16 D R REDDY
An approach to computer speech recognition by direct analysis of the speech wave
Technical Report C549 Palo Alto Cal Stanford University
Computer Science Department 1966
- 17 G W HUGHES
The recognition of speech by machine
Technical Report 395 Research Laboratory of Electronics
Cambridge MIT Press 1961
- 18 K N STEVENS
Study of acoustic properties of speech sounds
BBN Report 1669 AFCRL Report 68-0446 Bolt Beranek and
Newman Inc 1968
- 19 W TEITELMAN
Real time recognition of hand printed characters
Proc FJCC Spartan Press 1964
- 20 K N STEVENS
The quantal nature of speech: evidence from articulatory-acoustic data
In E E David and P B Denes eds *Human communication A
Unified View* New York McGraw-Hill in press
- 21 K N STEVENS G von BISMARCK
*A nineteen-channel filter bank spectrum analyzer for a speech
recognition system*
NASA Scientific Report No 2 1967 Contract NAS 12-138

Computer models for speech and music appreciation

by P. B. DENES and M. V. MATHEWS

Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

INTRODUCTION

Computers have been used extensively in speech research for over 10 years now; they have been applied to the synthesis of musical sounds for a slightly shorter period. The results have produced models of sound production and perception which are intimately related to the synthesis rules programmed in the computer and indeed the program is often the best available model of the production or perception process.

The most difficult problem encountered in both speech synthesis and music composition is the introduction of the right kind of long range dependencies. In speech, excellent isolated sounds can be easily generated. Transition rules which synthesize high quality examples of two sequential sounds tax both the state of our acoustic knowledge and our computer programs. Longer range effects such as sentence stress have barely begun to be studied. In music, we can produce single notes which are either excellent imitations of existing instruments or interesting new tones, we can closely imitate a complicated style for a few notes with a composing algorithm, but we cannot generate minutes of sound which approaches the cohesiveness of human compositions.

In order to understand the underlying problems of long range structure, the rest of this paper will survey speech synthesis procedures and composing algorithms with special emphasis on this structure. As will become clear, although there are some similarities between the speech and music structures, the differences far outweigh the similarities.

Speech synthesis

Research on generating artificial speech has been going on more or less continuously for sev-

eral centuries.¹ More recently, a significant part of this effort has been devoted to "speech synthesis by rule," the process which converts sentences specified on a typewriter keyboard into readily understandable speech. Synthesizers of this kind are constructed to serve as models of the speech mechanism and are used for the study of the human speech process. They could also have practical applications, for example, for automatic voice read-out from computer-based information retrieval systems and from teaching machines, and for other ways of improving man-computer interaction. Unfortunately, however, nobody has yet been able to design a satisfactory method of speech synthesis by rule. This is not due to any lack of effort, but is more an indication of some unexpected complexities of the speech process.

The basic speech generating process is deceptively simple: natural speech is produced by the vibration of our vocal cords and this buzzy sound is transmitted through the tube formed by our throat and mouth. This tube, the vocal tract, has a number of resonances, called the "formants," and the frequency of the acoustic resonances depends on the shape of the vocal tract. The quality of the vocal cord buzz is modified by the formants when the sound is transmitted through the vocal tract. As speech sound after speech sound is pronounced, the movements of tongue and lips change the shape of the vocal tract; this will alter the frequency of the formants, and thereby the quality of the sounds produced. Intensive research in the 1940's and 1950's resulted in satisfactory explanations of what determines the acoustic resonances of non-uniform tubes of shapes like those of the vocal tract. The acoustics of speech production is therefore fairly well understood and a variety of working models are avail-

able. The understanding of the relevant acoustics, however, is insufficient for explaining the speech process as a whole. Experiments carried out at Haskins Laboratories in New York,^{2,3} by Ohman⁴ and Lindblom,⁵ and by others indicate that the acoustic features typical for a phoneme when pronounced in isolation are modified extensively when this same phoneme is pronounced in a connected sentence. The changes are complex and are a function of a variety of contextual circumstances. The spectrogram of the sentence shown in Figure 1 gives a number of examples of this. The syllable "comp" occurs twice in this sentence, the first time in a stressed position and the second time unstressed. The difference in the corresponding acoustic features is obviously very large. Again, the phoneme /l/ occurs in a word initial position and again in a final position and shows significant acoustic differences. Similarly for the phoneme /p/ when followed by different vowels.

Broadly speaking there are two classes of contextual influences that are significant. One is a result of linguistic factors such as stress, inflection, grammar and certain relations between speech sounds that vary from language to language. The other class is the result of articulatory context: our vocal organs are constituted in specific physical ways, their muscular control for continuous action is coordinated at some higher physiological level, their musculature allows only certain movements and the acceleration and velocity of their masses obeys the usual laws of dynamics.

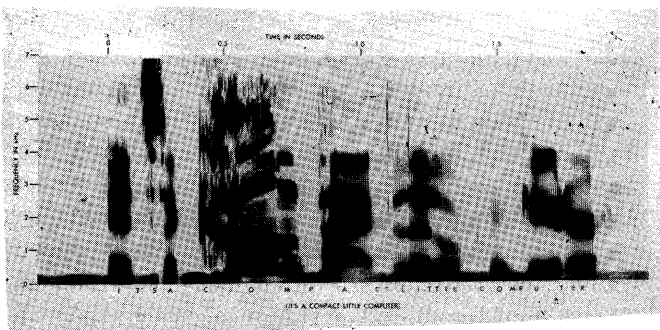


FIGURE 1—Sound spectrograms plot time along the horizontal axis and formant frequency along the vertical axis and indicate sound intensity by the darkness of lines. This spectrogram of the sentence, "It's a compact little computer" illustrates how phonemes vary according to context. Note the differences between the two "l" sounds in "little" and between the "comp" sounds in "compact" and "computer." These differences are influenced by many factors, which include sentence stress and inflection.

For many years our principal method for investigating natural speech was by examination of speech spectrograms. All effects, acoustic as well as articulatory and linguistic, were therefore evaluated mainly in terms of formant specifications.

Naturally enough, the models of speech production were also controlled in terms of formants. Such models are called "formant synthesizers." They simulate the vocal cords by pulses that repeat at the rate of the vocal cord vibrations: the frequency of these pulses determines the pitch of the synthesized speech sounds. The pulses are applied to a number of resonators, each representing one of the formants of the vocal tract. The vocal cord generator and the resonators can be simulated by computation or by conventional electronic circuits. In either case it is relatively simple to use computer programs for controlling the frequency and amplitude of the vocal cord pulses and the frequency and bandwidth of the resonators. Such computer programs would typically reset each of the above parameter values every 10 msec in order to produce the varying sound qualities appropriate for a spoken sentence. The schematic diagram of a computer-controlled electronic formant synthesizer can be seen in Figure 2 which shows the formant synthesis facilities of the Honeywell DDP 24 at Bell Laboratories.⁶ The DDP 24 controls the electronic formant synthesizer^{6a} by outputting a set of 12 numbers every 10 msec which set the 12 control parameters of the formant synthesizer (pitch, ampli-

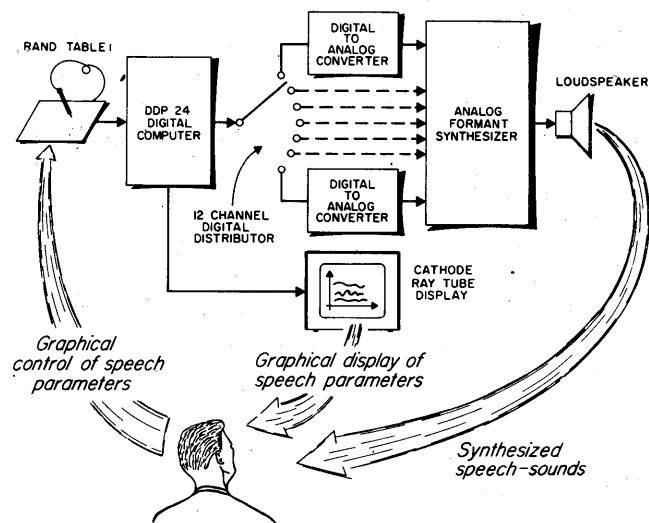


FIGURE 2—Schematic diagram of computer controlled formant speech synthesizer.

tudes, formant frequencies and bandwidths, etc). The experimenter not only hears the synthesized speech in real time but he can also see any or all control parameters displayed as a function of time on the screen of the computer controlled cathode ray tube. Programs are available for controlling each of the 12 parameters in one of two ways:

1. The experimenter can specify parameters as a function of time by drawing the appropriate curve using a Rand Tablet and he can listen to the corresponding change in the synthesized speech.

2. A sentence to be synthesized is typed into the computer, phoneme by phoneme. A computer program then calculates the variations in time of each parameter by using values stored for each phoneme and by applying programmed algorithms for determining the phoneme-to-phoneme transitions of these parameters. The computed parameter curves are then outputted to the synthesizer and to the display scope. Again, changes can be made with the Rand Tablet in real time. This second method is a typical example of synthesis by rule.

Sentences synthesized by rule usually have less than acceptable quality. At first it was thought that this was caused by inadequacies of formant synthesis. John Holmes,⁷ at the Joint Speech Research Unit in England, tested this assumption by a fundamental experiment. He synthesized the same sentence in two ways. First he controlled a synthesizer by copying, as closely as he could, the spectrogram of a human utterance of the sentence. The quality of the synthesized utterance was almost the same as that of the original human utterance. He then generated the same sentence by rule, using the same formant synthesizer, and by applying the most sophisticated rules available. The resulting speech quality left much to be desired. This experiment established that the inadequacies of synthesis by rule are due not to deficiencies in the synthesizing circuits but are caused by less than satisfactory understanding of the rules for combining successive phonemes into a continuous sentence.

Many different sentence synthesis rules have been tried during the last few years. Certain of these rules are concerned with transitions of parameter values from one phoneme to the other: some compute simple linear transitions,⁸ others compute two-part linear transitions,⁷ and others again use exponential transitions.⁹ These rules

almost never take into account the influence of more than three successive phonemes and are therefore essentially concerned with short term dependencies. Some of the synthesis by rule programs, however, also accept stress and punctuation marks as part of the typed input.^{8,9,10} These additional symbols are used by the program to control the formant values and duration of individual elements, and the pitch contours throughout the sentence. These grammatical factors effect successive elements over entire sentences rather than just two or three phonemes. Furthermore, as will be seen from the experiments described below, the factors to be considered greatly influence each other: only by computer techniques can we hope to implement these complex long term dependencies with reasonable ease.

The rest of this section will describe the synthesis by rule program written for the Bell Laboratories' DDP 24.⁸ The program computes short range dependencies by calculating parameter values as a function of adjacent phonemes; it also considers long range dependencies by computing duration and pitch values for each phoneme that are a function of where the stressed syllables are in the sentence and also takes into account whether the sentence is a statement or a question. The program also allows modification of the computed parameter values or durations either by using the Rand Tablet and the display oscilloscope or by typewritten instructions. This latter feature of the program was used for preparing synthetic speech test material for psychoacoustic tests aimed at finding quantitative relations between sound durations and pitch on the one hand and stress and inflection on the other hand.¹¹

The input to the program was obtained by typing the sentence to be synthesized in phonetic transcription. In addition to the phonetic symbols (see Table 1) a special symbol has to be typed to mark the stressed syllables and a period or question mark is to be typed at the end of the sentence to indicate if the sentence is to be a statement or a question.

Typically, each phoneme consisted of a central steady state segment, an initial transition and a final transition.

Stored values were available for the steady state segments of each of the phonemes for each of the 12 parameter values and also durations for each of the three segments and a "weight" for the transitions. Phoneme to phoneme transitions were computed as shown in the examples of

TABLE 1

Phonetic Symbol Used	Key Word	Phonetic Symbol Used	Key Word	Phonetic Symbol Used	Key Word
P	Pan	M	Moon	EE	Heat
T	Tan	N	Noon	I	Hit
K	Can	NG	Sing	E	Head
B	Ban	L	Limb	AE	Had
D	Dan	R	Rim	AH	Hot
G	Gun	Y	Yes	AW	Ball
F	Fan	W	Win	U	Put
TH	Thin	DH	Then	OO	Boot
S	Sin	Z	Zebra	UH	But
SH	Shin	ZH	Garage	ER	Sir
H	He	V	Victor		

Diphthongs were considered sequences of two vowels.

LIST OF PHONEMES ACCEPTED BY SYNTHESIS PROGRAM

Figure 3. The three examples show the influence of the relative "weights" of adjacent phonemes.

The stress marks automatically doubled the stored duration of the steady state segment of the stressed vowels and halved the stored duration of the unstressed vowels.

The pitch was originally computed on a monotone. The pitch of each stressed syllable was

made to decrease: the start of this pitch fall could be above or at the monotone level and was a variable, selectable by a Rand Tablet operated light button.

The pitch change of the final stressed syllable in each sentence decreased or increased for statement or question respectively. The intelligibility of the sentences generated by this program varied greatly for different sentences.

The same program was used to prepare test material for exploring quantitative relations between stress and the duration and the pitch of vowels.¹¹ The sentence "They are flying planes" was chosen for one of the tests. As is well-known, this is a grammatically ambiguous sentence. In one interpretation "are flying" is the verb and "planes" the object; in another way "are" is the verb and "flying planes" the noun phrase. The first version's 2413 *grammatical* stress pattern (1 is the greatest stress) becomes a 3521 if *semantic* stress is added to "planes"; the second interpretation's 2341 stress pattern becomes 3412 if semantic stress is added to "flying." The clear difference in meaning due to the stress oppositions of the last two words of the sentences with semantic stress was explained carefully to the subjects. This ensured that their responses were in terms of the differences in meaning caused by the position of stress in the sentence and avoided the need to explain the concept of "stress" to the subjects.

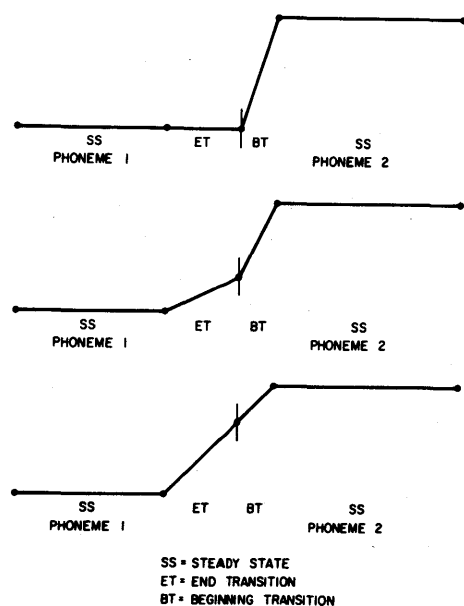


FIGURE 3—Examples of transitions between phonemes for three different "weights."

The duration and pitch of the steady state elements of the leading vowels of the diphthongs in "flying" and "planes" were varied systematically. The pitch of the sentence was kept constant at 130 Hz except as required in the test vowels. The results are shown in Figures 4 and 5. They indicate that increased duration and increased pitch of a vowel increase the likelihood of hearing the syllable that include that vowel as stressed. However, stress perception was influenced not just by the duration and pitch of the vowel concerned but also by the duration of pitch of other vowels in the sentence. The results also show a trade-off between duration and pitch: if stress is perceived on syllable A because of its higher pitch level then stress can be

shifted to syllable B either by changing the duration or the pitch of its vowel. Another important effect is that "flying" required considerably less pitch or duration increase to be heard stressed than did "planes." This means that different syllables have different requirements of relative pitch or duration in order to be heard stressed.

This may have a number of explanations. It may simply be due to the different vowels in the two syllables, or because of their different phonetic environments. It could also have been due to the sentence-final position of "planes," or due to various semantic influences. Whatever the reason, it shows an important effect that was also observed in other synthesized sentences: equal duration or equal pitch of all the vowels of a sentence do not produce equal stress. Under these conditions of equality the stress falls on the syllable which for grammatical or semantic reasons should have the least stress. Yet another factor is that by principle the definition of vowel duration must be arbitrary: the particular ad hoc definition adopted could lead to difficulties when the same vowel is used in different phonetic environments.

In summary, the strong influence of vowel duration and pitch on stress perception can be easily demonstrated. The quantitative relationship, however, is variable and affected strongly by a great variety of phonetic, grammatical and semantic conditions.

The outlook for good quality computer generated speech is not quite as black as these results might imply. Phonetic, grammatical and semantic constraints can be restricted by allowing only sentences made up from a small number of words and with a restricted grammar. The resulting ensemble of sentences could still be significant for such applications as voice read-out of computer stored information and at the same time exhibit more manageable long range effects of stress on the duration, pitch and formants of speech sounds throughout the sentence. Work on such simpler schemes are in progress now and computers are indispensable to handle the exploration of the remaining complexities that are still formidable. The lessons learned from such simpler synthesis schemes may well give useful cues for more ambitious schemes later.

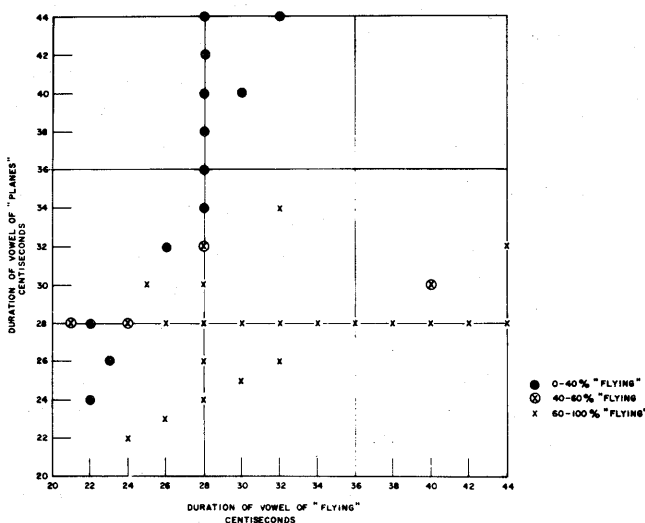


FIGURE 4—Stress judgments as a function of vowel duration.

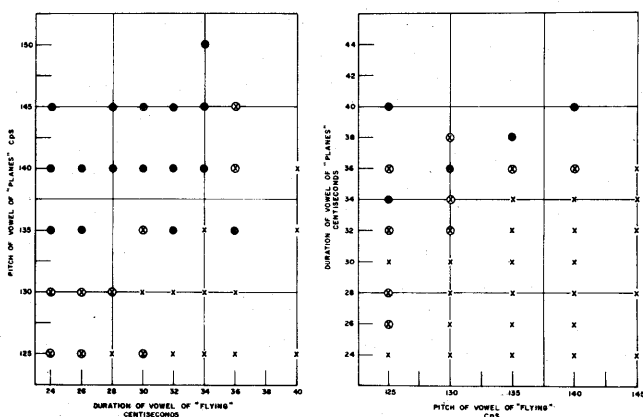


FIGURE 5—Trade-off between duration and pitch of vowels for stress judgments. The symbols used in Figure 4 also apply to this figure.

Composing algorithms

Music composed by machines has intrigued

musicians since Mozart's time, but until computers became available, only very simple processes could be attempted. Now, almost any conceivable composition algorithm can be instrumented. We will survey the methods that have been tried by describing four approaches which bracket most other efforts. In each description we will point out the kind of long range structure achieved by (or missing in) the procedure.

As a pedagogical standard of structurelessness, with which to compare these processes, we will postulate an algorithm which generates successive notes by a sequence of independent random choices. The parameters of the notes (pitches, durations, etc.) are chosen from uniform distributions. If two or more voices are involved, they are completely independent. Such a process both is and sounds structureless.

A number of the first programs^{12,13,14} were based on imposing sequential dependencies on the process we have just described. These algorithms produce note sequences in which (1) the probabilities of individual notes fit a prespecified distribution, or (2) the probabilities of two successive notes fit a prespecified second order distribution, or (3) the probabilities of three successive notes fit a prespecified third order distribution, etc. The processes have the advantage that these distributions can be estimated for existing compositions and the composing algorithm can be set to imitate an existing style.

The overwhelming conclusion from these processes is: sequential dependencies are not a well appreciated, long range musical structure. First and second order control produced subjectively significant changes from structureless sequences. Olson and Belar were able to compose "another Stephen Foster tune" in this way. Control of dependencies up to the 8th order were attempted (though accurate estimates of the higher order probabilities are not possible from the amount of existing music). Little more was achieved by the higher order control. The highest dependencies sounded repetitious and redundant. No intermediate length generated more interesting sequences than those produced with 1st or 2nd processes. Neither did the intermediate sequences closely resemble the styles on which their distributions were estimated.

These results are hardly surprising; sequential dependencies have proven to be poor describers of almost all time functions associated with intelligent human activity.

Sequential dependencies are intuitive to mathe-

maticians but have little association with music theory. A more inventive approach which incorporated some music theory was developed by Hiller and Isaacson¹⁵ and used to compose the Illiac Suite, a very respectable experiment. We shall denote the process as random generation plus rejection by rule. Hiller tried a version of the rules of first species counterpoint and obtained highly, if locally, structured results.

To describe the process in more detail, a trial "next note" is produced by an entirely random selection. The note is then checked by a set of rejection rules. If it violates any of the rules, it is rejected and another random choice is made. The process is repeated until either an acceptable choice is found or the program gives up and the entire composition is abandoned.

The great advantage of Hiller's method is that it can use very complicated rules which have strong and peculiar sequential dependencies. Some examples of the counterpoint rules which he used are:

"In proceeding from one chord to the next, at least one of the four voices must move by stepwise motion or remain stationary."

"Only consonant intervals between voices are permitted (unisons, octaves, perfect fifths, thirds, sixths.)"

"Any melodic skip (minor third or greater) must be followed by a repeat or a stepwise motion."

The complexity and number of the rules is limited only by the probability of rejecting the entire composition. Structures can be introduced by this rejection process which would be very difficult to program by any constructive process. Moreover, rejection rules are the essence of many elementary composition courses.

The Illiac Suite powerfully demonstrates style and local structure. A few bars exude the unmistakable hymn like style of the counterpoint. The style and short term structure is much stronger than that produced by sequential dependencies. However, if one continues to listen, in about 30 seconds he senses an aimless wandering. Hiller and Isaacson's rules did not include long range dependencies. The absence of structure is apparent.

In a specific effort to control long range structure, J. C. Tenney developed the third process which we will describe.¹⁶ It can be called controlled random selection. Each parameter of each successive note is selected randomly and inde-

pendently from a uniform distribution. But the mean and limits of this distribution are selected by the composer and can change over the composition. These changes constitute a long range structure.

The process can best be described with an example. The specification of two parameters, duration and loudness, for one voice is shown on Figure 6. The solid line is the mean of the range and the dotted lines the extremes of the range. The abscissa goes from 0 to 420 seconds, the duration of the composition. At some times (0 to 120 seconds) the composer can allow the computer great latitude as in choice of durations; at other times (300 seconds) he can allow no range and hence fix the computer's choice, as at 300 seconds. Average values can be likewise manipulated.

Tenney and Gerald Strang have used this method to achieve very marked long range structures. The ranges and rates of change of the distributions must be carefully selected to be perceptible to the listener, but the composer can become highly skilled at making these selections. By contrast, the short range structure is not nearly as elaborate or as perceptible as that of Hiller and Isaacson. Short sections sound like groups of random notes, which may or may not please the listener depending on his regard for John Cage's style.

One further question may be asked—who is doing the composing, the composer or the program? The answer is both, since the perceived output depends on both of their activities. The long range structure is introduced by the composer; the individual notes are selected by the

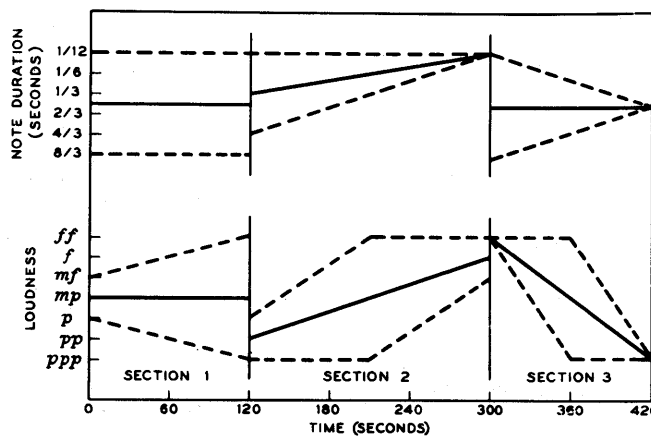


FIGURE 6—Score for Random Composition Generated by the Tenney Algorithm.

computer. Whatever is implied about authorship, it is a most effective process from the composer's viewpoint. He can quickly and easily construct the broad structure of the piece with sweeping strokes of the pen (light pen). The computer can then do the tedious details of filling in the myriad of notes. We will comment more later on the question of authorship and innovation.

The three processes we have described so far have involved random number generators. An alternate method called algorithmic development has been tried by Mathews.¹⁷ In this approach, the computer is given a theme plus an algorithm which modifies or develops the theme. The method is not well defined in the sense that many algorithms are possible and the nature of the composition as well as its success depend on the particular algorithm.

As an illustration, one example which has generated interesting results is shown on Figure 7. The theme is considered as not one sequence of notes, but as a sequence of frequencies $F_1 \dots F_6$ and a sequence of durations $T_1 \dots T_6$. Additional frequencies, F_7 and F_8 , are appended to the frequency sequence. The algorithm generates notes by proceeding cyclically through both the frequency and duration sequences, thus producing the notes $F_1 T_1, F_2 T_2, \dots, F_6 T_6, F_7 T_1, F_8 T_2, F_1 T_3$, etc. Since the frequency sequence is longer than the duration sequence, the phase between frequencies and durations is continually changing. The process is cyclic and will repeat itself after 48 notes.

The algorithm develops the theme in the sense of changing it perceptually but in a way so the developed theme can be associated with the original. The short repeated melodic and rhythmic sequences provide the perceptual clues which bridge the theme and the development. Achieving



FIGURE 7—Example of Cyclic Development of a Theme

perceptual but associable changes appear to be two criteria of a useful development algorithm. Producing an interesting sequence of notes is a third criterion.

A number of other techniques for modifying themes such as constructing "a weighted average" between two existing themes have been tried by Rosler and Mathews¹⁸ and have generally produced interesting results, particularly in rhythm. We will not describe the processes further here.

Long range structure is provided by the perceptual association between the theme and its development. Short range structure is provided by the theme itself. Hence, it has been possible to produce compositions with interesting structures over periods of several minutes. One can again ask how much is contributed by the algorithm and how much by the composer.

In a specific sense the answer to the question of authorship is clear for any algorithm. In a philosophical sense, it is far from settled. The first three composing processes involve random number generators. However, computers do not produce true random numbers but, rather, they calculate long, periodic, deterministic sequences. Given the initial number and the algorithm, the sequence can be exactly anticipated and, hence, the composer could conceivably incorporate its details into his composition. As a practical matter, of course, he does not. John Cage proposed that authorship is a subjective judgment on the part of the human composer. The notes or aspects of the composition which he has planned or anticipated are his contribution; the parts which he is surprised by when he first hears the music are the machine's contribution. A successful algorithm produces desired rather than unwanted surprises.

CONCLUSIONS

The need for better long range structure is one of the most important unsolved problems for both speech synthesis and music composition by computer. In speech, the main question is how the individual sounds are modified by the environment in which they occur, where environment means not only the adjacent sounds but also the entire sentence and the frame of mind or purposes of the speaker. In music, the main question is how to relate the sounds in one part of a composition to the sounds in another part in a perceptually meaningful and interesting way.

The differences between long range structure in speech and music arise from the fundamental ways in which we perceive and use these sounds. Speech synthesis is usually judged by its articulation score (understandability) and preference, where preference is measured by naturalness or lack of accent. Articulation cannot be measured for music since the nature of its information is so different from speech. Perhaps the equivalent question is whether the style of the composing algorithm can be learned in the sense of distinguishing examples from examples of some other algorithm. Preference can be measured, but here one asks questions about the overall piece rather than the naturalness of individual notes. Long term structure in speech is measured in seconds; in music it is measured in minutes.

Despite differences, speech and music research have often been mutually reinforcing. Many questions about voice quality and the timbre of musical sounds appear to be different aspects of the same psychoacoustic phenomena. Certainly the same computers and similar programs are useful for speech and music research. Consequently, we believe it is useful to compare the long range structure of these processes.

REFERENCES

- 1 H DUDLEY T H TARNOCZY
Speaking machine of Wolfgang von Kempelen
JASA Vol 22 pp 151-166 1950
- 2 A M LIBERMAN et al
The role of selected stimulus variables in the perception of the unvoiced stop consonants
Am J Psychol Vol 65 pp 497-516 1952
- 3 A M LIBERMAN et al
Perception of the speech code
Psychological Review
Vol 74 pp 431-461 Nov 1967
- 4 S E G OHMAN
Coarticulation of VCV utterances: spectrographic measurements
JASA Vol 39 pp 151-168 1966
- 5 B LINDBLOM
Spectrographic study of vowel reduction
JASA Vol 35 pp 1773-1781 1963
- 6 P DENES O C JENSEN
Speech synthesis with the DDP 24 on-line computer
Unpublished
P B DENES
Real-time speech research
Proc Symposium on the Human Use of Computing Machines
Bell Telephone Laboratories 1966
- 6a C H COKER P CUMMISKEY
On-line computer control of a formant synthesizer
JASA 38 1965 p 940A
- 7 J N HOLMES I G MATTINGLY J N SHEARME
Speech synthesis by rule
Language and Speech Vol 7 pp 127-143 July Sept 1964

-
- 8 P DENES R OSTER
Speech synthesis by rule on the DDP 24 computer
Unpublished
- 9 L R RABINER
Speech synthesis by rule: An acoustic domain approach
BST JVol 47 No 1 pp 17-37
Jan 1968
- 10 I G MATTINGLY
Synthesis by rule of prosodic features
Language and Speech Vol 9 pp 1-13 Jan March 1966
- 11 R OSTER
Two experiments using speech synthesized on the DDP 24
To be published.
- 12 F P BROOKS JR A L HOPKINS JR P G NEUMANN
W V WRIGHT
An experiment in musical composition
IRE Trans on Electronic Computers
EC-6 175 1957 Brooks F P Jr *Correction* ibid EC-7 60 1968
- 13 R C PINKERTON
Information theory and melody
Sci American 194 2 77 Feb 1956
- 14 H F OLSON H BELAR
Aid to music composition employing a random probability system
JASA 33 1163 1961
- 15 L A HILLER JR L M ISAACSON
Experimental music
McGraw-Hill Book Co New York 1959
- 16 J R PIERCE M V MATHEWS J C RISSET
Further experiments on the use of the computer in connection with music
Gravesaner Blätter No 27/28 Nov 1965 p 92-97
- 17 M V MATHEWS
The computer music record supplement
Gravesaner Blätter Vol 7 No 26 1965 p 117
- 18 M V MATHEWS L ROSLER
Graphical language for the scores of computer-generated sounds
Perspectives of New Music Vol 6 No 2 1968

A computer with hands, eyes, and ears*

by J. MCCARTHY, L. D. EARNEST,
D. R. REDDY and P. J. VICENS

Stanford University
Stanford, California

INTRODUCTION

The anthropomorphic terms of the title may suggest an interest in machines that look or act like men. To this extent it is misleading. Our interest is in extending the range of tasks to which machines can be applied to include those that, when performed by a human, require coordination between perceptual and motor processes. We attempt to suppress the egocentric idea that man performs these tasks in the best of all possible ways.

In place of "eyes, ears, and hands" we could refer to "cameras, microphones, and manipulators," but find latter terms less suggestive of the functions that we wish to emulate. We leave the term "robot" and the ideas that go with it to the science fiction writers who have made them so entertaining.

Shannon, Minsky, McCarthy, and others had considered the possibility of a computer with hands, eyes and ears at one period or another during the latter part of the last decade. The main obstacles to the realization of the idea were the unavailability of suitable computers and I-O devices, and the prohibitive cost of such a system. Ernst¹ and Roberts² were among the first few who used a computer to realize these objectives. Glaser, McCarthy, and Minsky³ proposed that the first major attempt at the biological exploration of Mars should be made by a computer controlled automatic laboratory, containing a wide variety of visual input devices and mechanical manipulators which can under computer control perform many of the tasks of bio-chemical laboratory, requiring

only a limited supervision by the experimenter on earth.

The work reported here and a related project at M.I.T. were undertaken several years ago to combine and improve techniques for machine perception and manipulation. Progress has been slower than we hoped because there have been many previously unrecognized problems and few simple solutions. Nevertheless, we have a system that does such things as recognize spoken messages that are combinations of terms previously "learned," "see" blocks scattered on a table, and manipulate them in accordance with instructions.

The work is just beginning. Our existing system exhibits more problems than solutions, but that was its purpose. The following sections discuss considerations leading to the choice of equipment, techniques used to convert the huge masses of television camera and microphone data into useful information, and the control of arms.

System configuration

Major considerations in the choice of system components have been off-the-shelf availability and ease of interfacing. This approach has both advantages and disadvantages. The main advantage, of course, is a relatively quick start. Figure 1 shows major components of the existing system. Figure 2 is a photograph of the hand-eye system.

At the center of the system is a time-shared PDP-6 computer with 131K words of core memory of 36 bits each and an 11 million word fixed-head disk file. The PDP-6 was chosen for having a working time-sharing system, ease of adding special I-O devices, and unrestricted data transfer rates of up to 30 million bits per second between memory and external devices. The Librascope

*This research was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contract No. SD-183.

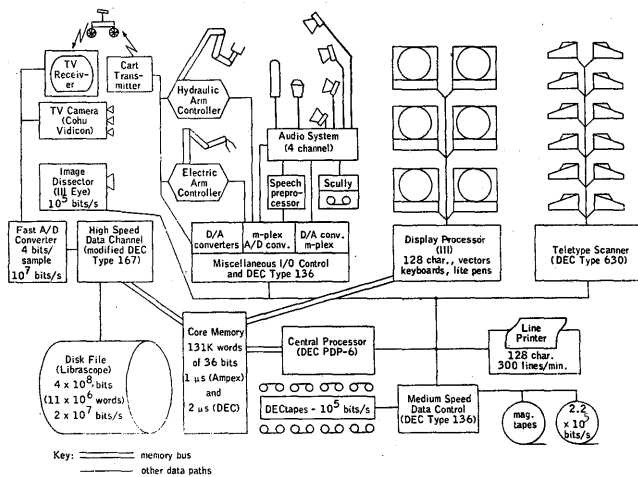


FIGURE 1—Stanford A.I. computer system

disk file has a 24 million bit per second transfer rate and provides both swapping and permanent file storage.

There are a number of local and remote teletypes, CRT displays, a line printer, plotter, and tape units for general user services.

Our research objectives imposed several special requirements on the time-sharing system. When a person begins to speak, the system must ensure that the audio system is listening to him and must not swap out the program just because its time is up. Similar considerations apply to arms that are in motion. The system must also provide for communication between programs. The DEC time-sharing system was modified locally to meet these requirements.

Visual input to the system is provided by a vidicon television camera operating in accordance with EIA standards. The video signal is digitized to four bits (16 levels of light intensity) and sampled at an instantaneous rate of 6.5 million samples per second. Making use of interleaving, any rectangular portion of the image, up to 666×500 points for the full field of view, may be read into memory under program control in two frame times (1/15 second). For static scenes, finer gray-scale resolution can be obtained by averaging measurements over multiple frames.

The laboratory also has an image dissector camera which is capable of measuring the brightness of image points in arbitrary order. It is capable of directly resolving better than 1000×1000 points with a gray-scale resolution of 6 bits or more. It is relatively slow if a large number of

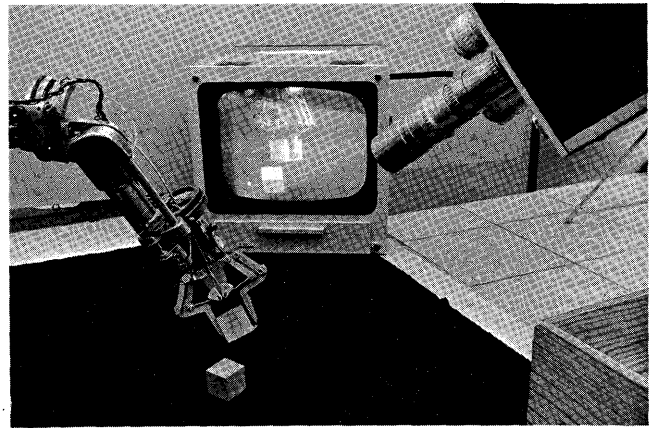


FIGURE 2—A picture of the hand-eye equipment

points are to be read and suffers from settling-time problems when large deflections are used.

A comparative study of optical sensors for computers, including the possibility of a laser eye with direct depth determination, has been made by Earnest.⁴

Audio input to the system is provided through an A-D converter connected to the PDP-6. Two different audio devices are currently attached. In one, composed of a condenser microphone and a high quality amplifier, the speech signal is sampled at a rate of 20,000 samples per second and digitized to 9 bits. In the other, shown on Figure 3, the output of a crystal microphone is amplified and filtered into three frequency bands. In each band, the maximum amplitude of the signal and the number of zero crossings are measured by analog circuitry. Every 10 milliseconds, each hold circuit is read by the A-D converter to 16 bits and then reset for the next 10 milliseconds.

Input of the raw speech waveform without any preprocessing hardware, such as a filter bank, has the disadvantage of requiring more processing by the computer and more storage. But on the other hand, it provides the user with a very flexible means of analysis and permits all kinds of processes to be simulated. In fact, we believe that no solution should be implemented in hardware until it has been proven to be one of the best possible solutions by computer simulation. Reddy⁵ states that prosodic parameters of speech, requiring the use of segmentation and pitch detection are more easily determined from the direct speech signal than from the output of a bank of filters.

The second audio-device arises directly from the preprocessing program of Reddy and Vicens.⁶

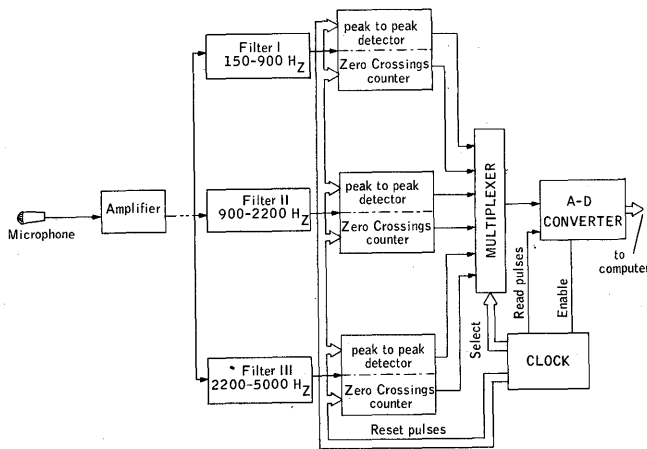


FIGURE 3—A schematic of the speech preprocessing hardware

They use two smoothing functions which produce approximations of the same parameters (maximum amplitude and zero-crossings). This kind of system is required when one wishes to analyze long utterances, because a direct analysis of the speech wave would consume large amounts of core storage and preprocessing time.

The example of speech recognition discussed below uses the preprocessing hardware. The sampling rate of this device, being very slow (3600 bps), allows the central processor to process the input as it comes, which is an important consideration if real time speech recognition is desired.

The electric arm was originally designed as a device to be strapped to a paralyzed human arm. It has six degrees of freedom which permits it to place the hand in arbitrary positions and orientations within reach, plus a finger-closing motion. It is powered by small permanent magnet gear-head motors mounted on the arm, giving joint velocities of 4 to 6 r.p.m. with small loads. Position feedback is provided by potentiometers mounted at each of the six joints. The hand is a two finger parallel grip device approximately the size of a human hand and has a maximum finger opening of 6.4 centimeters (3.5 inches). The maximum reach is about 68 centimeters (27 inches), and its weight is about 7 kilograms (15 pounds). Power to the motors takes the form of 16 volt pulses whose width and repetition rate are controlled by the computer program.

In its original form, this arm had a number of maladies such as severe mechanical play in the joints and imprecision in the potentiometer read-

ings due to unstable mountings. Despite several improvements, the arm is still rather slow, shakey, and inaccurate. The position of the hand may differ from computed values by as much as a centimeter.

A hydraulic arm, recently completed, is faster, smoother, and more accurate in its motions. It is also capable of dealing a fatal blow to the experimenter; exhibits other antisocial properties such as leaking hydraulic fluid, and tends to destroy itself periodically.

Scene analysis and description

If we digitize the light intensity at every point in the whole field of view of the television camera, the computer will receive 666×500 or 333,000 samples, or 1,332,000 bits of data per frame. The problem of scene description is the formulation of a program which will abstract meaningful descriptions of objects of interest in the scene and their positions.

The problem of scene description must be distinguished from the problem of classifying pictures into categories with which much of the published theory deals. The first working description program was reported by Roberts.² Narasimhan⁷ suggests that richly structured pictures such as bubble chamber pictures, line drawings, and others are best studied in the form of picture analysis and description and proposes the use of a linguistic model. Recent work by Miller and Shaw,⁸ and Shaw⁹ illustrates the power of the linguistic models in the analysis and generation of pictures.

The linguistic models have the advantage that they can be used for both analysis and generation of pictures, and that many of the powerful tools developed for syntax-directed analysis of languages can be directly utilized for the analysis of pictures. The weaknesses of the present linguistic models, at least as far as the analysis of images of 3-D scenes is concerned, are the following:

- . Attempts at describing the connectivity of 3-dimensional objects, using a data structure primarily developed for the description of strings often results in unwieldy and awkward descriptions leading one to doubt whether such descriptions really facilitate analysis. Extension of the models to use list structures instead of strings should remedy this weakness.

- . The present linguistic models also suffer from many of the problems of error recovery of

syntax directed compilers. This is especially critical when dealing with analysis of pictures; as a result of noise and jitter in the input device spurious lines and edges may appear all over the picture, and often some of the expected edges may be missing.

One of the main problems with images of 3-D scenes is not so much how to describe what is visible but rather how to describe what is only partially visible. Heuristics for handling degenerate views of objects cannot be conveniently incorporated into presently proposed linguistic models.

In view of the above consideration it appears that generalizations of linguistic models will be needed before they can be used effectively for the analysis of images of 3-D scenes.

Our existing scene analysis program is related to the one used by Roberts² and has recently been described by Pingle, Singer, and Wichman.¹⁰ Instead of repeating that description, we shall note some shortcomings of the existing system and some possible solutions.

The existing eye program locates cubical blocks of various sizes scattered at random on a contrasting background. Its "world model" has room for just one block at a time and those that are partly obscured by others may be perceived only after the intervening blocks have been removed by the hand. Depth determination depends on the assumption that all objects rest on a known planar surface.

The edge tracing program in use does not reliably detect subtle differences in brightness between adjacent surfaces of the same or similar objects. The block stacking tasks that have been undertaken to date do not require this information.

A more general world model, in the form of a multiply-linked data structure, is being devised that will accommodate at least multiple objects bounded by combinations of planar surfaces in arbitrary positions. More powerful edge tracing procedures are being tested, and we plan to employ some of the contiguity recognition techniques of Guzman¹¹ together with three-dimensional plausibility tests of postulated objects.

In related work, the problems of combining information from several views and viewpoints into a single model is being attacked. We expect these combined efforts to produce a much more complete description of the work environment.

Speech analysis and description

If we plot the changes in air pressure produced by a speech utterance as a function of time we will see a waveform such as the one given in Figure 4. This signal, as reflected by the changes in voltage generated by the microphone, is digitized in our system to 9 bit accuracy every 50 μ s, resulting in a data rate of about 180,000 bits per second of speech. In normal speech, every second of speech contains about 5 to 10 different sounds which usually require less than 50 bits to represent in the written form. The problem of speech description, then, is the development of a set of procedures which will reduce the 180,000 bits of information to about 50 bits. Of course, human speech carries other information such as speaker identity, emotional state, his location, age, sex, health and other such features. But what is of primary interest to us here is the message uttered by the speaker.

First attempts at speech recognition by a computer were restricted to the recognition of simple sounds like vowels and digits, just as preliminary attempts at picture recognition were restricted to the recognition of characters. The approaches developed for the recognition of simple sounds, such as the use of a metric in a multidimensional space partitioned by hyperplanes, are not easily extendable for the analysis of a complex sequence of sounds which may be part of a spoken message. The structure of the message and the interrelationships among the sounds of the message are the important factors.

Speech is, perhaps, the most extensively investigated of all the human perceptual and motor processes. And yet a large body of this research is not directly relevant for machine recognition of speech. Even the relevant literature on the acoustic characteristics of speech is more qualitative than quantitative and is meant for use by men rather than by machines. Stevens¹² has recently summarized much of the known data on acoustic properties of speech sounds in a form amendable for machine processing. Recent attempts at computer speech recognition by Bobrow and Klatt¹³ and Reddy⁵ provide models which can be used for the recognition of phrases, sentences, and connected speech. The latter forms the basis of present work. The model currently being used consists of four stages: segmentation, sound description, phrase boundary determination, and phrase recognition.

Segmentation

If you consider the sound in Figure 4 you will see that it is not clear where one word ends and another begins or where a particular sound within a word ends and the next begins. This is because the shape of the vocal tract is continuously changing and there is no clear cut point in time where we stop saying one sound and start another. To be able to associate discrete symbols with the continuous speech wave, a machine must be able to segment a connected speech utterance into discrete parts.

To be able to segment speech we need to answer questions such as "What is a sound?" and "How do you distinguish one sound from another?" One can define a *sound* on purely acoustical basis: a sound is a part of the speech wave in which the acoustic characteristics are similar (a sustained segment) or one in which the characteristics vary with time (a transitional segment). To distinguish one sound from another according to the above definition we need a measure of similarity or closeness between two adjacent units of speech.

Conventional metrics such as the Euclidean distance fail to be satisfactory. To be usable the closeness function should be based on the following heuristics:

- . Since some parameters are more variable than others the closeness function should provide for appropriate weighting of parameters.
- . Although most of the parameters may be similar a drastic change in one parameter should result in a 'not-similar' indication.
- . If the difference between two corresponding parameters is less than a minimum, then the two parameters should be considered as identical.
- . The greater the parameter value, the greater should be the difference we are willing to accept, suggesting the use of a relative error function such as dy/y .
- . When the parameters are close to zero the relative error function dy/y can take abnormally large values, suggesting the use of a function such as $k \cdot dy/\sqrt{y}$

A closeness function which satisfies the above heuristics and a detailed description of segmentation are given by Reddy and Vicens.⁶ The segmentation process can be summarized as follows. A preprocessing procedure divides the speech wave into 10 ms minimal segments and calculates estimators of four characteristic parameters: the

amplitude and zero crossings of the dominant frequency under 1000 cps and the amplitude and zero crossing of the dominant frequency under 1000 cps. An alternative for this preprocessing task is to use special hardware;⁷ it divides the speech wave into 10 ms minimal segments and accumulates six parameters: amplitude and zero crossings of the signal in three frequency bands: 150-900 Hz, 900-2200 Hz, 2200-5000 Hz.

A primary segmentation procedure calculates closeness values and groups together adjacent minimal segments that may be regarded as similar. A secondary segmentation procedure divides these primary segments into smaller segments if the within-the-segment variation of parameters is too high. The closeness values are recomputed between the secondary segments, giving greater weight to the frequency components than the amplitude components. If two secondary segments are sufficiently close and are not local maxima or minima, they are combined to form larger segments.

Sound description

The purpose of generating a sound description is to abstract, from the wide range of possible values of parameters, a label which would adequately describe the nature of a sound segment. The higher the level at which a sound is described, the easier it is for the pattern matching and recognition routines to determine what was said. At one extreme the description might consist of the average parameters of the segments and the other a single label for the whole utterance. In between, descriptions can be attempted at the level of phoneme groups, phonemes, diphones, or syllables. The nature of our segmentation is such that it is more appropriate for us to attempt description in terms of phonemes or phoneme groups.

Phoneticians have classified the sounds we produce according to the shape of our vocal tract. There are about 40 such different sounds (phonemes) in English. One natural description of a speech utterance is in terms of its phonemic transcription. For example the word *picks* could be described as consisting of four sounds, P, I, K, and S in that sequence. However, various allophones of the same phoneme exhibit widely varying acoustic characteristics depending on the context in which they occur. This often results in a substantial overlap of characteristics among similar

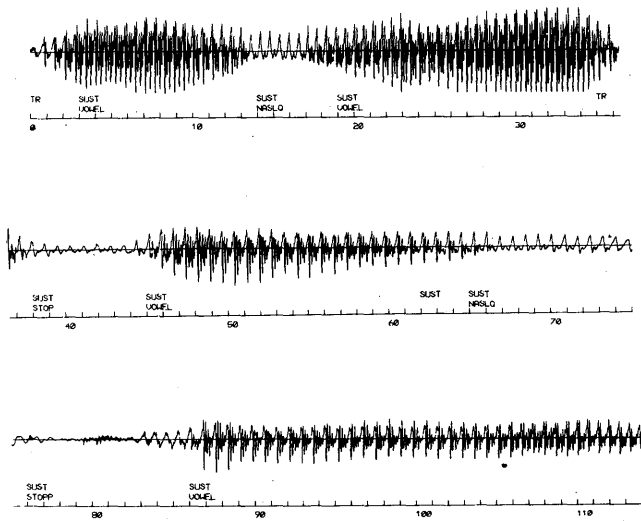


FIGURE 4—Analysis of the waveform of "How now brown cow"

phonemes. Thus it is not uncommon for the word *bits* to have similar acoustic parameters to *picks*. This possibility of error precludes the use of simple pattern matching routines. At present, we generate descriptions in terms of phoneme groups: vowel, nasal, fricative, stop, burst and so on. We will see later how such a description is useful in reducing the search space.

The procedure used for the classification of segments into phoneme groups is an extension of the one given by Reddy.⁵ If a segment is noiselike, then it is labelled FRICS. Otherwise if the segment-amplitude is a local maximum, then the segment is labeled VOWEL. Otherwise the segment is labeled STOP, NASAL, CONSONANT depending on segment parameters. For example, the description generated for the word *picks* might be as follows: "The sound consists of a stop, followed by a transition, followed by a vowel, followed by a stop, followed by a fricative each with the following parameters. . . ."

Word and phrase boundary determination

Like many other aspects of English, the problem of determination of word boundaries in connected speech is ambiguous. For example the sound description /AISCREEM/ could have resulted from the words "I scream" or "ice cream." The problem of word or phrase boundary determination can be completely by-passed if the problem at hand requires only the recognition of a limited set of words, phrases or sentences. Then

the sound description of the whole utterance can be stored in the lexicon for future matching with a similar descriptive. However, as the lexicon gets larger and larger it becomes necessary to consider breaking up a connected speech utterance into words and phrases which can then be recognized by a phrase recognition program.

One obvious solution to this problem is to require the speaker to pause for a few milliseconds between words or phrases. But this gets to be annoying after a while. At present we are considering connected speech utterances of the form

```

<command> ::= <function name> <argument list>
<argument list> ::= <argument> | <argument list> <preposition> <argument>
<function name> ::= PICK UP | STACK | ASSIGN | ADD | SUBTRACT | ...
<argument> ::= THE LARGE BLOCK | FAR LEFT | ALPHA | BRAVO | ...
<preposition> ::= AT | OF | TO | FROM | ...

```

By carefully choosing the function names, the possible arguments, and the associated prepositions it is possible to determine the word or phrase boundaries. Certain keywords play an important part in this determination. A good example of such a word is BLOCK, starting with a silence (B) and ending with a silence (K). As a result such heuristics as 'scan until you find BLOCK' may be done with a low percentage of error. Use of restricted special purpose command languages for communication to the computer such as the one above is not unreasonable in view of the fact that we have had to make a similar compromise for programming languages. How interesting the spoken languages can get will depend on how reliably and precisely we can generate sound descriptions.

Word and phrase recognition

Given a sound description of a word or phrase, we need a description matching algorithm to determine what was said. If we can guarantee that the description is an error-free phonemic tran-

scription then all that is needed is a simple classification net which grows as new sounds are uttered. Since such a guarantee will not be forthcoming in the near future, if ever we need a recognition procedure which will cater to the posances of the same phrase by a single speaker can exhibit different descriptions even under the same environmental conditions. Due to minor changes in the emotional state he may say it faster, slower or with slightly different stress and intonation resulting in a loss of segment, insertion of a segment, or assignment of a wrong label, e.g., NASAL instead of STOP. This possibility of error in the description poses the well known lack of synchronization problem, i.e., if two descriptions of the same phrase differ by one or more segments, the problem of determining which one it is that is missing.

Given two descriptions which are to be compared, a mapping procedure determines the possible correspondences between segmental descriptions. It uses the heuristic that VOWEL and FRICS segments can be reliably detected and first maps VOWEL for VOWEL and FRICS for FRICS. The remaining unmapped segments are then mapped on the basis of similarity of parameters.

Given the correspondences between segment descriptions, an evaluation procedure compares the parameters of the mapped segments to determine if they could possibly be two different utterances of the same phrase. Similarity of the parameters is given based on the heuristics given for the closeness function in the section on segmentation. Of course, any unmapped segments have a detrimental effect on the similarity measure. If the similarity value is over a certain threshold then the two descriptions are considered the same.

If no candidate was found during the first try, the program, assuming that it knows what was said, supposes that the FRICS were not well determined and were classified as a high frequency stop (burst) or vice versa. If this is the case, a second search is attempted mapping only VOWEL for VOWEL. All the remaining unmapped segments are then mapped on the basis of similarity of parameters as in the first try (but with FRICS included).

If after this second try no satisfactory candidate was found, two different actions may take place: in learning mode, the new description is entered

into the lexicon along with the print name; in recognition mode, it is rejected.

Unless the candidates for pattern matching with an incoming description are chosen carefully from the lexicon, it could take a long time to determine what was said. To minimize the search space, the lexicon is ordered on the basis of the number of vowel segments and the number of FRICS segments, furthermore the vowels are classified in nine subclasses according to the values of their zero crossing parameters. The direct matching of these subclasses easily eliminates some entries in the initial list of the possible candidates. Each entry in the lexicon consists of a packed version of the description and parameters generated by the sound description procedure. A detailed description and evaluation of the phrase recognition procedure is given by Reddy and Vicens.¹⁵

Remarks

The preceding subsections attempt to give an overview of the state of accomplishment in speech recognition at our project. Segmentation and phrase recognition procedures give correct results about 95 percent of the time. This can be improved slightly by using more sophisticated preprocessing routines. Work has barely begun on the determination of classes of words and/or phrases for which boundaries can be determined unambiguously in connected speech. A great deal of work remains to be done in the generation of reliable sound description.

If a reliable description can be obtained in the form of a phonemic transcription (or some such unit) we can reduce the search space of the word and phrase recognition routines considerably, and we will be able to unambiguously determine word boundaries for a larger class of words. Only then can we hope to recognize words from a lexicon of, say, 20,000 words in close to real time. We have already mentioned that the main difficulty in obtaining a reliable phonemic transcription is the wide variability of acoustic characteristics of a phoneme depending on context. Theoretically every phoneme can occur in 1600 or so different contexts. Many of them do not occur in natural speech and the remaining can perhaps be grouped together into 10-20 contextual categories for each phoneme. The huge task that remains to be done is the investigation and methodical cataloguing of the modifications to the features of a phoneme, and the development of rules for transformations

on phonemic features based on context. It will perhaps be many years before such a study is complete but a great deal can be done in computer speech recognition even with incomplete results using the model proposed here.

Control of a mechanical arm

In order to carry out complex manipulation tasks, it is necessary to do planning for and control of the arm at several levels. At the top level there is a goal-seeking process which integrates the activities of the various sensory, perceptual, model-building, and manipulation processes. Next there must be planning of subtasks. For example, if we are given a description of an object to be assembled and a description of available components, we must plan which components will go in which locations and the order in which they are to be placed.

Each subtask generates a sequence of motions (e.g., move hand H to point P and open fingers). At this point, the model of the environment should be checked for space occupancy conflicts (i.e., the arm shouldn't bump into things accidentally). In case of conflicts, we must replan the arm motions and, possibly, the order in which components are put in place.

Given that the hand is to reach a certain position, we must calculate how each of the arm joints is to be positioned. For arms with certain geometric properties, this can be a very quick and reliable calculation. For others, it may involve a slow and uncertain iterative process.

Finally, there is a process that servos the arm from place to place, possibly with constraints on the velocity or force to be employed.

Our existing system exhibits each of these levels of planning and control in some form, but without much generality. In most cases, an ad hoc sequence of subroutine calls takes the place of a flexible planning function. As one consequence, the arm readily runs into objects in its vicinity. The calculation of joint positions required to reach a given point is relatively straightforward for the arms we have and has been described by Pingle, Singer and Wichman.¹⁰

An obstacle avoidance technique has been devised by Pieper.¹⁶ It attempts to make the point of closest approach greater than a specified value between all parts of the arm, modelled as a series of cylinders, and an environment containing planes, spheres, and cylinders.

There is much to be done in the area of planning assembly tasks. Many of the things that we do instinctively, such as building things from the bottom up or from the inside out, need to be formalized and translated into programs.

An example

As an illustration of existing capabilities, we describe a system that obeys the experimenter's voice commands to find blocks visually and stack them as ordered. The grammar chosen for this example is as follows.

Syntax

```

<command> ::= <command1> |
            <command2>

<command1> ::= <order1> | <order1>
              EMPTY

<order1> ::= RESCAN | STOP

<command2> ::= PICK UP <argument list>

<argument list> ::= <size indicator> EMPTY
                  <position indicator>

<size indicator> ::= EMPTY | <size> BLOCK

<size> ::= SMALL | MEDIUM |
         LARGE | EMPTY

<position
  indicator> ::= EMPTY | <position1> |
              <position2>

<position1> ::= <position> SIDE

<position2> ::= <position'> <position">
              <position word> |
              <position"> <position'>
              <position word>

<position word> ::= ANGLE | CORNER

<position> ::= <position'> | <position">

<position'> ::= EMPTY | LEFT | RIGHT

<position"> ::= EMPTY | UPPER | LOWER

```

Semantics

The meaning of some of the terminal symbols is obvious, but some others, like RESCAN and EMPTY need explanation.

The command "rescan" is used to indicate that the scene might be disturbed and that the vision program should generate a new scene description.

The terminal symbol EMPTY means no speech utterance at all or sounds not recognized by the word recognizer. If any of the non-terminal symbols is finally reduced to EMPTY the middle value is assumed. For example if $\langle \text{size indicator} \rangle = \text{EMPTY}$, a block of medium size will be assumed.

Sentences like "pick up the small block standing on the upper right corner," "rescan the scene," "pick up any block" are correct according to the grammar.

After the preliminaries such as training the phrase recognition system and calibration of the arm and eye coordinate systems, the picture recognition program looks at the image and generates a scene description of all the cubes present in the field of view. The description for each block consists of the location, size, and orientation of the block.

Given a command, the speech analysis program segments the speech and generates a sound description. This description is then used by the scanner-recognizer which decodes it and passes the result of its analysis to the main program.

The scanner-recognizer requires a good word recognizer utility program. The recognition is done by scanning the speech utterance description forward and backward using feedback from the grammar. The decoding of a sentence like "pick up the small block standing on the right side" will be done as follows:

```
Recognize PICK UP.
Then scan until BLOCK.
If BLOCK backtrace to find size attribute.
Backtrace from the end to find SIDE.
Backtrace to find RIGHT.
```

At any step, feedback is used so that the only candidates considered are those that are syntactically correct. For example, when the program is trying to reduce the non terminal symbol $\langle \text{size} \rangle$ the only available candidates for the matching process are the descriptions of LARGE, SMALL and MEDIUM.

Based on the command, the arm is directed to pick up or stack a block. If it is to pick up, the location and orientation of the block are given. If it is to stack, the location of the stack is given. The movie, to be shown, illustrates the response of the arm to various commands, and presents the details of various analysis and description generation processes displayed on a CRT.

CONCLUSIONS

Many of the problems discussed at the end of the preceding sections can and will be solved in the next few years. However, it will probably be a long time before a computer can equal the perception and dexterity of a human being. This will require not only advances in the areas of computer architecture and the quality of the external devices, but also a better understanding of perceptual and motor processes.

Even the limited progress achieved so far can result in computer hand-eye-ear systems that are better suited for some purposes than human beings. For example, they may see things and hear sounds that a person cannot, and they may be faster, stronger, more economical, or more expendable.

The fact that a computer may not be able to see all the things we can see or carry on fluent conversation should not be a cause for extra concern. Consider the case of programming languages. Although we have not been able to communicate with computers in our natural language, we have managed to achieve a great deal using contrived and ad hoc languages like Fortran. There is no reason to suspect that the same will not be the case with visual and voice input to the computers or with computer control of manipulators.

We foresee several practical applications that can profitably use the techniques described in this paper. One that is most often mentioned is the possible bandwidth reduction in picture and speech transmission systems. We believe that computer controlled carts which can navigate themselves, and automated factories, where computer controlled manipulators with visual feedback can handle many situations which cannot be presently handled by fixed sequence manipulators, are also within the range of the present state of the art.

ACKNOWLEDGMENTS

The success of any system of this magnitude depends on the team effort of many people. It is a pleasure to acknowledge the contributions of Karl Pingle, who did most of the eye programming, and Jeff Singer, who developed most of the arm and hand programs. Excellent hardware and software system support were provided by Stephen Russell, Gerald Gleason, David Poole, John Sauter and William Weiher.

REFERENCES

- 1 H A ERNST
MH-1 a computer operated mechanical hand
Doctoral Thesis MIT Cambridge Massachusetts 1961
- 2 L G ROBERTS
Machine perception of three-dimensional solids
Optical and Electro-Optical Processing of Information MIT Press Cambridge Massachusetts 1963
- 3 D GLASER J MCCARTHY M MINSKY
The automated biological laboratory
Report of the subgroup on ABL summer study in exobiology sponsored by the Space Science Board of the National Academy of Sciences 1964
- 4 L D EARNEST
On choosing an eye for a computer
AI memo No 51 Computer Science Department Stanford University 1967
- 5 D R REDDY
Computer recognition of connected speech
J Acoust Soc Am 42 329-347 1967
- 6 D R REDDY P J VICENS
A procedure for segmentation of connected speech
To be published in J Audio Engr Soc 16 4 1968
- 7 R NARASIMHAN
Syntax-directed interpretation of classes of pictures
CACM 9 3 166-173 1966
- 8 W MILLER A SHAW
A picture calculus
GSG Memo 40 Computation Group Stanford Linear Accelerator Center Stanford California 1967
- 9 A SHAW
The formal description and parsing of pictures
PhD Thesis CS Report No 94 Computer Science Department Stanford University Stanford 1968
- 10 K K PINGLE J A SINGER W M WICHMAN
Computer control of a mechanical arm through visual input
To be published in the proceedings of the IFIP 68 1968
- 11 A GUZMAN
Some aspects of pattern recognition by computer
MAC-TR-37 Project MAC MIT Cambridge Massachusetts 1967
- 12 K N STEVENS
Unpublished paper 1968
- 13 D G BOBROW D KLATT
A limited speech recognition system
To be published in the proceedings of FJCC 68 1968
- 14 P J VICENS
A speech preprocessing device
Stanford Artificial Intelligence Memo to be published 1968
- 15 D R REDDY P J VICENS
Spoken message recognition by a computer
To be published 1968
- 16 D PIPER
Kinematics of manipulators under computer control
PhD Thesis Stanford University to be published 1968

Digital simulation of continuous dynamic systems: An overview

by JON C. STRAUSS

Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION

There has been a growing interest over the last decade, both in this country and abroad, in the use of the digital computer to model continuous dynamic systems. Prior to 1960, and still true in large measure today this application area had been the exclusive province of the analog computer. The initial motivation for integrating the ordinary differential equations characteristic of continuous system simulation models on the digital computer was one of accuracy; given enough time, solutions of essentially unlimited accuracy could be obtained from the digital computer. These accurate, but costly, solutions were then used as consistency checks on the many hundreds of solutions characteristic of a simulation study on a high speed electronic analog computer.

This emphasis has changed somewhat over the years but not strictly on a speed ratio alteration basis. While digital computers have become appreciably faster, analog computers have more than kept pace. In addition, hybrid computers (combined systems of analog and digital computers) have been developed and are being employed in an attempt to exploit the best features of both analog and digital computation. It is still true that for many applications on a per solution basis, all other things equal, the analog and/or hybrid computer is several orders of magnitude faster than a digital computer of equivalent cost.

Among the main items which have increased the relative usage of digital computers in continuous system simulation are:

- 1) guaranteed accuracy of solution,
- 2) accessibility, and
- 3) the existence of problem oriented simulation languages that smooth the interface between the analyst and the computer.

New developments in graphical hardware and conversational software promise to increase the usage of digital computers in this application area even more. Thus while analog and hybrid computers continue to retain a marked speed and cost advantage over digital in certain problem areas, notably optimization and real time simulation, it appears that usage of digital computers in general continuous dynamic system simulation will continue to increase.

The first attempts at digital computer simulation were characterized by the extremely long set up and checkout time associated with the coding and debugging of individual programs in machine language. With the advent of effective problem oriented algebraic compiler languages (e.g., Fortran, Algol, etc.) in the late '50s and early '60s, this delay time was somewhat reduced. Efficient numerical integration programs were, however, still very complicated and hence costly and time consuming to write and debug on a single problem justification basis. Even the "packaged" numerical integration programs tended to require programming sophistication for their efficient utilization. Moreover at this level of programming complication, it was still difficult to get the engineer who was, and is, concerned with the difficulties inherent in his problem representation to become concerned with the difficulties inherent in the problem solution. He would obviously have been more willing to become involved if simulation programming involved mainly model representation, as on the analog computer, and he was somehow buffered from the solution difficulties.

It was this concern combined with the growing realization that a digital computer was more than just a large "number cruncher" that led to the design of the first problem oriented simulation languages. In format, the early languages resembled a combination of a three address absolute machine code and an operational block

description of an analog computer wiring diagram. In solution techniques, they generally weren't much better; most offered rectangular or some other low accuracy integration scheme. They did, however, make an important contribution; i.e., they introduced the engineer interested in dynamics to the digital computer.

More recently, simulation languages have been developed that permit model description in an equation based representation that is independent of analog computer notation. Through provision of a varied set of functional operators, equation based notation can include block diagram based notation as a subset.

A review of the literature gives the impression that there has been almost too much work reported on digital simulation languages (in the sense of lots of work and too little thought). In addition, much of this work has ignored the intimate relationship between the design and use of simulation languages, the structure of the simulation process, and the natural environment of numerical integration. A history of simulation language development is presented in several survey articles.^{1,2} There is, therefore, little need for repetition here.

The riot of invention (and duplicated effort in all too many cases) together with the apparent neglect of the numerical integration process indicated the need for control and synthesis. In response, the Simulation Software Committee was organized under the auspices of SCI to develop standardization guidelines for continuous dynamic system simulation languages. In an effort to dispel the vague generalities associated with many standardization efforts, it was decided to present the recommendations in the form of a communication language for dynamic system simulation. This language, the Continuous System Simulation Language (CSSL), was specified in a recent report.³ The CSSL report does not address the problem of optimal numerical integration system design, but it does attempt to relate the structuring capabilities of CSSL to the inherent structure of numerical integration and to a general structure for the simulation activity. It is these interrelationships that form the basis for the current discussion.

Details of numerical integration system design are presented in Reference 4. Reference 5 describes a simulation language system that augments the recommended CSSL solution options by providing for steady state and frequency response in addition to the standard transient response calculation. Reference 6 introduces an area of dynamic system simulation that to date has received very little attention in terms of problem oriented simulation languages, namely, the simulation of spatially continuous systems.

The character of simulation

In a recent book,⁷ Evans, Wallace and Sutherland present an interesting discussion on the "nature of simulation." Their definitions are paraphrased and particularized here for the restricted environment of continuous dynamic system simulation.

Continuous Dynamic System: System in which the response phenomena occur continuously in one or more independent variables; i.e., the response can be modeled by systems of algebraic, ordinary differential, partial differential, and difference equations.

Of primary concern here are difference equation models and/or ordinary differential equation models (which ultimately are reduced to iterated difference equation models as part of the numerical integration process). Reference 6 introduces the topic of simulation language design for partial differential equation models.

Mathematical Model: A set of equations, the solutions of which represent some corresponding response phenomena in the system. The question of adequacy of the representation, while certainly important, is not an issue here; suffice it to say that the solution of the model represents the behavior of the system against some criteria.

For example, the solution, $x(t)$, of the differential equation:

$$m \ddot{x}(t) + d \dot{x}(t) + kx(t) = gm \quad (1)$$

represents the position of the mass (parameter m) in a simple linear spring, mass, and dashpot mechanical system. The model of Equation 1 assumes an ideal linear spring (spring parameter k), an ideal linear dashpot (parameter d), negligible air resistance (unless accounted for in d and k), and a known acceleration of gravity (g).

Simulation: The process of solving the mathematical model for a particular set of parameters and conditions. The solution is said to represent the behavior of the system for the same set of conditions; i.e., it represents the true system behavior subject to the adequacy of the model.

For example, the solution of the mathematical model of Equation 1 over the interval $0 < t < t_f$ with parameters:

$$d = 1, k = 1, m = 1$$

and conditions:

$$x(0) = 0, \dot{x}(0) = 0, t_f = 10$$

constitutes a simulation.

Simulation Study: One or more applications of simulation to a system.

For example, the process of repeating the simulation of the last example five times increasing m by $+ .1$ each time would be a simulation study of the effects of increasing mass on the dynamic response of the system.

The mathematical model is the representation of the system. The simulation involves a command over the mathematical model (e.g., *FOR* "d = 1, k = 1, . . . etc." *SOLVE* "Math Model" *OVER* "Range.") The simulation study involves an ordered sequence of simulations (or a procedural sequence of commands).

These definitions provide a framework within which the relationship of simulation languages and the numerical integration process is easily presented.

As mentioned in the introduction, a simulation language must obviously be concerned with the representation of the system being simulated; i.e., with the description of the mathematical model. It must also provide for the presentation of the parameters and conditions that specify a single simulation. To meet the requirements of the simulation study, it must, in addition, provide for the specification and monitoring of the experimental procedure that is a simulation study.

It is of some interest to note the disparity of these requirements. The model of a continuous dynamic system is parallel in character; i.e., to solve the first order vector differential equation in Equation 2,

$$\dot{z}(t) = f(z(t), t) \quad (2)$$

it is necessary to know $z(t)$ to compute $f(z(t), t)$ to employ in the process of computing $z(t)$, etc. Description of the mathematical model involves the specification in a problem oriented notation of $f(z(t), t)$ in Equation 1. Because this is a parallel model, it is clear that this description should be, in general, nonprocedural; i.e., it should be independent of the order of description. Moreover, the description should be in a notation convenient to the discipline in which the model originates; i.e., the operators of the language, whether blocks or general functions, should be appropriate for the original level of description of the model. The specification of the simulation study procedure is, on the other hand, a standard procedural task. For the most part it is entirely satisfactory to describe such a task with a standard procedural language such as Fortran or Algol. In fact, as is discussed in Reference 3, one of the major design criteria for the CSSL effort was that it resolve these conflicting requirements of non-procedural model representation and procedural task specification.

It is the function of the numerical integration system to solve the ordinary differential equations of the model and in so doing perform a single simulation in the fashion dictated by the conditions and parameters of the simulation. The numerical integration system has the task of operating on the differential equations that describe the parallel model so as to give a reasonable approximation to the solution on the serial (sequential) digital computer.

It thus becomes clear from the definitions that the simulation language must be concerned with the model representation and the simulation study activity while the numerical integration system has responsibility for the actual mechanics of the simulation (i.e., the solution of the model).

Structure of a simulation

A discussion of the general structure of a simulation of a dynamic system is best approached through a specific example of a numerical integration scheme. The scheme chosen for presentation here is the Euler Method. This scheme has the virtue of illustrating the idea without requiring burdensome detail. The working hypothesis is that since a simulation is, by definition, a solution procedure, the general structure of the continuous system simulation process should be intimately related to the general structure of numerical integration.

Euler integration method

The object is to integrate Equation 2 from point t_n to point t_{n+1} . Certainly the simplest, non-trivial result is to expand $z(t_{n+1})$ in a Taylor Series about $z(t_n)$, as in Equation 3, and save only the first two terms, as in Equation 4.

$$z(t_{n+1}) = z(t_n) + \dot{z}(t_n)(t_{n+1} - t_n) + \ddot{z}(t_n) \frac{(t_{n+1} - t_n)^2}{2} + \dots \quad (3)$$

or

$$z_{n+1} = z_n + hf_n \quad (4)$$

where: $z_n = z(t_n)$ $h = t_{n+1} - t_n$

Equation 4 is a satisfactory, but not very accurate, integration formula known as the Euler Method. Note that Equation 4 is an approximation for $z(t_{n+1})$ which is represented exactly in Equation 3, but the terms involving the second and higher derivatives are, in general, too unwieldy to calculate. The Euler Method is of the prediction (extrapolation) type since it does not need any information concerning the derivative function (f) at t_{n+1} to integrate (extrapolate) to t_{n+1} .

A flow chart of a digital computer program to in-

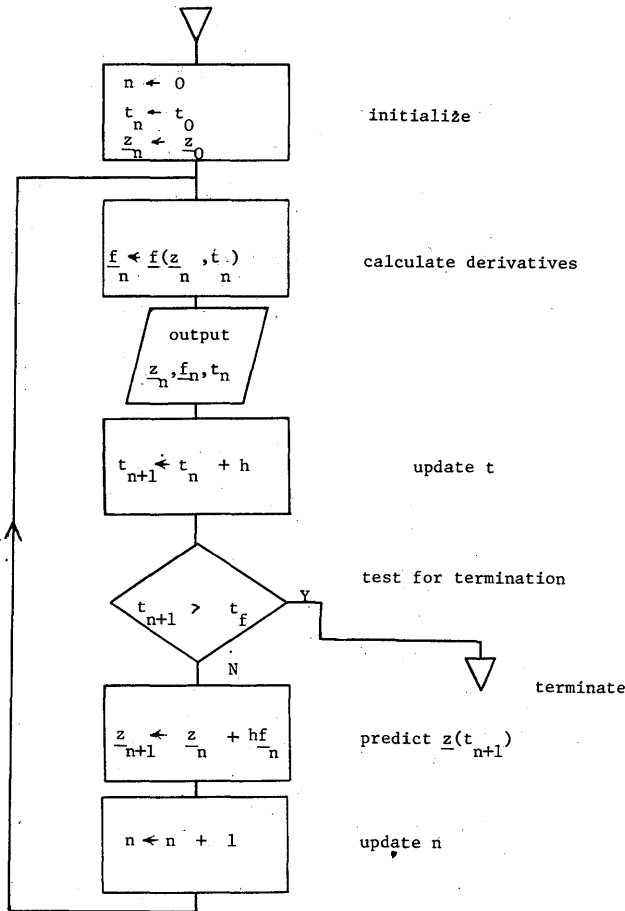


FIGURE 1—Flow chart of Euler method use

tegrate Equation 2 employing Equation 4 from t_0 to t_f in steps h is presented in Figure 1.

General structure

Inspection of the flow chart for the use of the Euler Method in Figure 1 reveals that certain of the operations are strictly a function of the particular prediction formula employed while others are strictly dependent on the model (namely the derivative calculation) and still others are strictly a function of the numerical integration task. These latter operations (i.e., updating of the independent variable t , testing for completion, and routing of control between tasks) are referred to as the numerical integration environment.

Figure 2 presents a general flow chart for a simulation of a system with the general mathematical model of Equations 5 and 6.

$$\dot{z}(t) = f(z(t), p_s(t), x(t)) \tag{5}$$

$$y(t) = h(z(t), p_m(t), x(t)) \tag{6}$$

$$z(t_0) = z_0$$

where (5) is the state equation,
 (6) is the measurement equation,
 $z(t)$ is the state vector,
 $p_s(t)$ is the vector of parameters in the state variable model,
 $x(t)$ is the vector of system inputs,
 $y(t)$ is the vector of measurable system outputs.
 $p_m(t)$ is the vector of parameters in the measurements model,
 z_0 is the initial condition vector for $z(t)$.

The flow chart for the Euler Method in Figure 1 can, with only slight modification, be presented in the more general outline of Figure 2. There are several other salient features to this organization:

1. The operations peculiar to the model are isolated in a routine that calculates the state variable derivatives, f , and in the calculations for the measured variables in the system, y .
2. The initialization operations are isolated from the integration operations.
3. The input/output operations are assumed to take place at a fixed frequency ($1/\Delta$ in simulated time). This is reasonable in light of the interpretation demands of digital simulation; i.e., printouts and

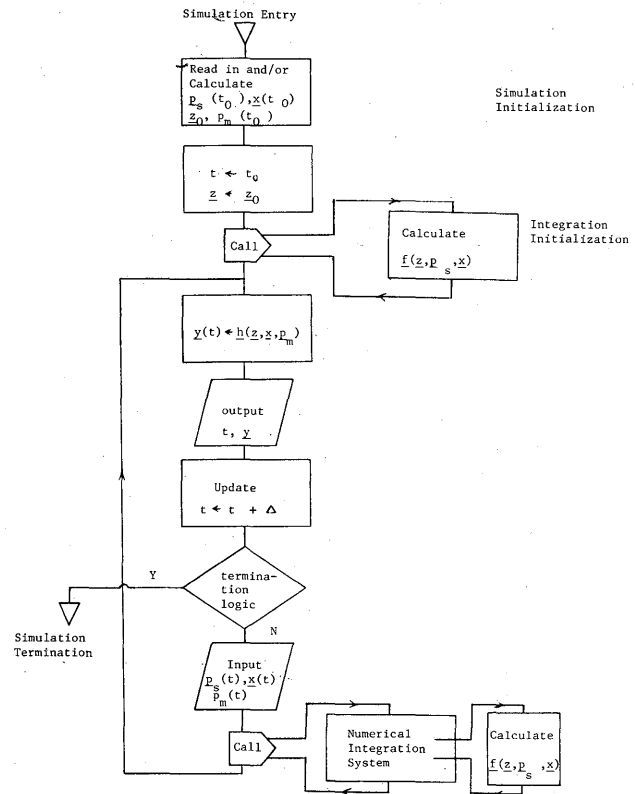


FIGURE 2—General simulation flow chart

plots are more easily read and understood if information is presented at a fixed interval in the independent variable. This interval, Δ , is known as the *communication interval*.

4. The communication interval is not necessarily the interval employed in the numerical integration being performed under control of the integration system. This other interval, δ , is termed the *calculation interval*; it is not necessarily constant, but generally is an integral factor of Δ . The detailed design of numerical integration systems is not at issue here. (Reference 6 relates some of the design considerations to the general structure presented here.)
5. The termination logic terminates the simulation. The control is then passed to a control program concerned with attaining the objectives of the simulation study. This program might do nothing more than return control to the simulation entry to read in new data (z_0, p_m, p_s, x) for another simulation of the model, or it might modify parameters in the model based on the results and return control for another simulation as part of an optimization algorithm.

Simulation study structure

Figure 3 presents a general control flow structure for a simulation study program. As might be expected from the definition of simulation study, this structure consists of a controlled iteration of the flow chart of a single simulation presented in Figure 2. It is interesting to note that the general structure of Figure 3 is not peculiar to the digital simulation activity; rather any continuous system simulation study (digital, analog, and/or hybrid) can be couched in this structure.

The similarity between Figures 2 and 3 clearly emphasizes the relationship between the general structure of a simulation study and that of the numerical integration process. The relationship of the general structure to a continuous system simulation language is provided by the regions denoted on Figure 3 as Initial, Dynamic and Terminal. As is discussed in Reference 3, CSSL provides a programmable structure capability that encourages, at the source program level, the description of actions to be taken by the simulation study program at the appropriate point in the control flow of Figure 3.

CONCLUSIONS

The structure of the numerical integration process determines the structure of a run time digital computer program for the simulation of continuous dynamic systems. If simulation languages are to facilitate the

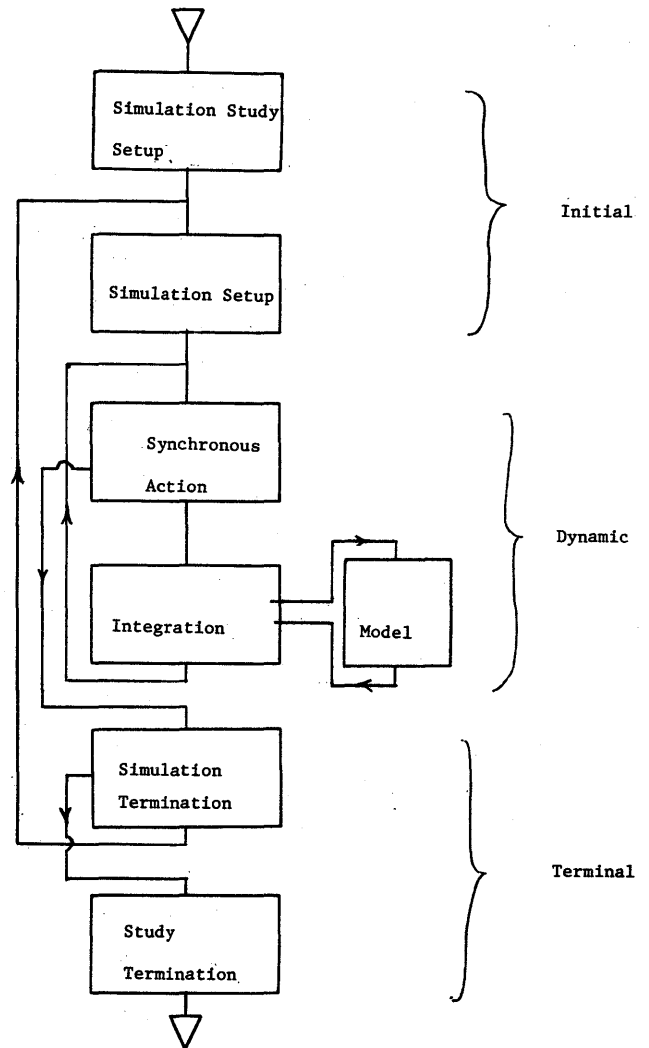


FIGURE 3—Simulation study control structure

complete simulation study activity on the digital computer, they must provide appropriate interfaces to the structure determined by numerical integration and the simulation study requirements.

REFERENCES

- R D BRENNAN R N LINEBARGER
- 1 *A survey of digital simulation; digital analog simulator programs* Simulation Vol 3 No 6 Dec 1964
 - 2 J J CLANCEY M F FINEBERG
Digital simulation languages, a critique and a guide AFIPS Conference Proceedings Vol 27 1965 FJCC December 1965
 - 3 *The SCI continuous system simulation language (CSSL)* Report of the SCI Simulation Software Committee J C Strauss, Editor Simulation Vol 9 No 6 December 1967
 - 4 D H BRANDIN
The mathematics of continuous system simulation AFIPS Conference Proceedings

Vol 33 1968 FJCC December 1968
5 T E SPRINGER O A FARMER
*TAF—a steady state, frequency response, and time response
simulation program*
AFIPS Conference Proceedings
Vol 33 1968 FJCC December 1968
6 S M MORRIS W E SCHIESSER

*SALEM—a programming system for the simulation of systems
described by partial differential equations*
AFIPS Conference Proceedings Vol 33 1968 FJCC December
1968
7 G W EVANS G F WALLACE G L SUTHERLAND
Simulation using digital computers
Prentice Hall Inc N J 1967

Mathematics of continuous system simulations

by DAVID H. BRANDIN

Informatics Inc.
Sherman Oaks, California

INTRODUCTION

The purpose of this paper is to summarize briefly the state-of-the-art in numerical methods applied to simulation of continuous dynamic systems.* The principal concern is with numerical techniques encountered in general simulation problems. Because special purpose integration techniques and approximation methods which reduce systems of differential equations to systems of algebraic equations are not commonly employed, they are discussed only briefly.^{2,3,4}

This paper views the state-of-the-art in simulation mathematics from the engineer's point of view: What are the commonly used methods? Why are they used? How do they relate to accuracy, speed, and stability? What is their relation to digital simulation languages?

The problem

The purpose of a digital computer simulation of continuous dynamic systems is to generate the point-by-point solution along the time axis to a set of simultaneous differential equations which represent the mathematical model. The differential equations may be linear, nonlinear, partial, of any order, and have initial conditions, boundary values, and other constraints and complexities. However, simple algebraic techniques and approximations reduce virtually all systems of this type to a set of simultaneous ordinary differential equations of the first order.^{5,6} The numerical methods employed in the simulation generate the solution of these first order systems.

The numerical procedures to generate the solutions are well known—Runge-Kutta and predictor-corrector algorithms.

The application of these procedures on a digital computer presents practical difficulties. The constraints of the problem usually necessitate computationally

efficient and accurate solutions—computationally efficient because digital computers are expensive and thousands of runs may have to be made to validate a solution or perform a parameter variation study; accurate because refined models may be very sensitive to small variations in parameter values. Accuracy may also be needed to verify an analog computer solution or maintain a stable solution.

Needless to say, these two constraints have the unfortunate property that they are mutually contradictory. To improve speed (and reduce cost), practitioners suggest a minimum of extended precision arithmetic, low order integration algorithms, large integration step sizes, and “tight” programming. To improve accuracy practitioners suggest maximum use of extended precision arithmetic, stable, high order integration algorithms, and small integration step sizes. Numerical techniques for computer simulation of continuous systems are therefore bound to compromise either economy, accuracy, or both.

The problem, then is not trivial. Intelligent tradeoffs must be made when selecting the mathematical techniques. The programmer (or simulation analyst) must understand the physical system, the mathematical model, and the numerical techniques and their constraints.

Mathematical techniques

Numerical integration

The numerical integration techniques commonly encountered are extracted from the Runge-Kutta and predictor-corrector families. Most generalized simulation programs for a specific range of applications will have only one algorithm or one combination (e.g., predictor-corrector with a Runge-Kutta starter and variable step size) while many simulation languages will incorporate a variety of methods.^{7,8,9,10}

The Runge-Kutta family is characterized as follows:

*The definition of a Continuous Dynamic System is presented in an earlier paper in this session—Reference 1.

- The methods are self starting and do not require previous information,
- They require N evaluations of the derivative for an N th order method,
- The truncation error at each step is proportional to h^{N+1} where h is the integration step size and N is the order.

In contrast, the predictor-corrector family is characterized as follows:

- The methods are not self starting; they do require previous information,
- They may iterate the "corrector" until desired convergence is reached,**
- They provide an excellent approximation to truncation error if the predictor and corrector equations are of the same order,
- Previous information must be retained.

An argument can be made justifying the use of either of the families of integration algorithms. The choice is dependent on the nature of the systems of equations, required accuracy, costs, programming competence, etc. Defending the choice of any particular family for the general case is questionable since, for any given selection, a system of equations can be defined which are unsuitable for that method. The selection of a given algorithm within a given family is more controversial. Frequent selections are fourth order Runge-Kutta or Adams (Moulton) predictor-corrector. The Adams Family (a 2nd order method is displayed in Figure 1) has some desirable characteristics with respect to efficiency.¹¹

If one assumes fourth order algorithms and a fixed step size, then predictor-correctors (without repeated corrector iterations) require half the derivative evaluations as the Runge-Kutta method. In this case, predictor-correctors are presumed to be two times faster than Runge-Kutta methods. Unfortunately, nothing in simulation is this simple, as the next section illustrates.

It should be noted that the predictor-corrector algorithms require starting values which are usually provided by Runge-Kutta formulae. The incorporation of a variable step size and Runge-Kutta equations inevitably compounds the programming problems and makes it more difficult to demonstrate explicitly the superiority of one technique over the other.

It is interesting to note that a large class of second order differential equations commonly encountered in physical problems can be solved explicitly with

**Numerical integration subroutines are rarely programmed to perform repeated iterations of the corrector. See later section.

$$\text{GIVEN } \dot{Y} = \frac{dY}{dt} = f(t, Y)$$

$$Y_{N+1}^{(p)} = Y_N + \frac{h}{2} (3 \dot{Y}_N - \dot{Y}_{N-1})$$

$$\dot{Y}_{N+1}^{(p)} = f(t_N + h, Y_{N+1}^{(p)})$$

$$Y_{N+1}^{(c)} = Y_N + \frac{h}{2} (\dot{Y}_{N+1}^{(p)} + \dot{Y}_N)$$

Re-iterate Corrector or $Y_{N+1} = Y_{N+1}^{(c)}$

FIGURE 1—Second order Adams predictor-corrector formulae

Runge-Kutta formulae. These equations are of the form:

$$\ddot{y} = f(t, y).$$

Curiously, the techniques are described in the common literature¹² however they are rarely used. A third order method is displayed in Figure 2.

Variable step size

The predictor-corrector algorithms provide an estimate of truncation error which is superior to estimates of the error in Runge-Kutta algorithms. If $Y_i^{(p)}$ denotes a predicted point in the solution of the i -th equation and $Y_i^{(c)}$ denotes the corrected point in the i -th solution, the truncation error ϵ_i is given by

$$\epsilon_i = K |Y_i^{(p)} - Y_i^{(c)}|$$

where K depends on the specific predictor-corrector algorithm. With information of this type, it is common to vary the integration step size to take advantage of small truncation errors. One usually computes

$$\epsilon = \max_i \left\{ |Y_i^{(p)} - Y_i^{(c)}|, \left| \frac{Y_i^{(p)} - Y_i^{(c)}}{Y_i^{(c)}} \right| \right\}$$

where either the absolute or relative error for each equation is computed based upon certain criteria, e.g., the magnitude $|Y_i^{(c)}|$. Variable weight factors may be assigned to each error in order to vary its influence on the step size. Unfortunately, knowledge concerning the selection of the weight factors is usually limited and

Given $\ddot{y} = \frac{d^2y}{dt^2} = f(t, y)$

$$K_1 = f(t_n, y_n)$$

$$K_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2} \dot{y}_n + \frac{h}{8} K_1)$$

$$K_3 = f(t_n + h, y_n + h\dot{y}_n + \frac{h}{2} K_2)$$

$$y_{n+1} = y_n + h[\dot{y}_n + \frac{1}{6}(K_1 + 2K_2)]$$

$$\dot{y}_{n+1} = \dot{y}_n + \frac{h}{6} K_1 + \frac{2h}{3} K_2 + \frac{h}{6} K_3$$

FIGURE 2—Runge-Kutta formulae for a class of second order equations

they are frequently all set to the same value. The step size is commonly varied as follows:

- If $\epsilon < E_s$, double the step size;
- If $\epsilon > E_t$, halve the step size and restart the integrations;
- If $E_s \leq \epsilon \leq E_t$ the step size remains unchanged.*

The problems encountered with this technique are due to the substantial variation in ϵ from step to step, resulting in subsequent halving and doubling of the interval. This generates a large number of integration restarts, many iterations (frequently Runge-Kutta passes or backwards interpolations) to generate the required starting conditions, and a consequential loss in speed which the variable step size was intended to improve. Meaningful values of E_s and E_t are therefore necessary if the benefits of a variable step predictor-corrector algorithm are to be reaped. Although it is not possible to define general values for E_t and E_s , it is generally agreed that they should satisfy the relationship:

$$20 \leq \frac{E_t}{E_s} \leq 100$$

The reader can conclude, therefore, that predictor-

*Some programs always halve or double the step size at each step.⁷

corrector algorithms with variable steps are not unconditionally faster than Runge-Kutta techniques of the same order. Based upon this conclusion, many people select a Runge-Kutta algorithm with fixed step size (usually fourth order) and assume that no improvements in running time can be obtained.

Runge-Kutta algorithms are also used with variable steps. These techniques may employ two methods of different order and compare results at the completion of the full integration step. Another method is to integrate with the same algorithm but employing two different step sizes. Intermediate points in the solution common to both methods can be used by both algorithms to conserve running time.⁷ Other methods integrate in blocks of N steps utilizing information within each block. Also, by analyzing the behavior of the solution over N steps, estimates of the accuracy can be derived which compare in reliability with those for predictor-corrector methods.¹³

A commonly made suggestion encountered is to integrate different differential equations in the mathematical model with different integration step sizes. This technique is most commonly employed in hybrid calculations. Multirate integration techniques require good knowledge of the frequencies present in the mathematical model. If such a technique is implemented, the integration routines do become quite complicated. Canned simulation programs and simulation languages rarely incorporate multirate integration algorithms although their use is becoming more popular.¹⁴

Truncation and roundoff error

We all realize that it is theoretically possible to reduce the integration truncation error to zero at each step by driving the integration step size to zero. On a digital computer, however, this is impractical since the roundoff error will grow rapidly as the step size is decreased. The general relationship between roundoff and truncation error is illustrated in Figure 3. More exact curves can be drawn for specific algorithms. The error graphed in Figure 3 represents the error at each step. Clearly, the most desirable step size is one which will reduce the total error; that step size is somewhere in the vicinity of h_M . The exact value of h_M will depend on the specific shapes of the curves.¹⁵

Most analyses are directed at studies of truncation errors since these are far easier to investigate than roundoff errors. However, for problems with wide dynamic ranges of the independent and dependent variable, roundoff error may actually dominate the solution. In this event, arbitrarily decreasing the step size only compounds the error. In addition, the roundoff error may confuse the step size variation criteria and results may then become meaningless.

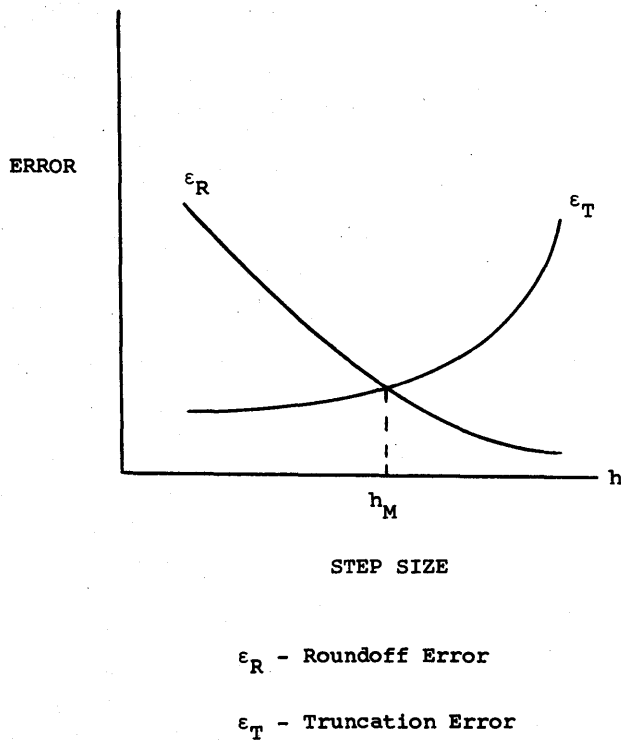


FIGURE 3—Roundoff and truncation error

A decrease in roundoff error can always be achieved by implementing extended precision arithmetic operations (e.g., double precision floating point arithmetic). If a variable step size is implemented it is not necessarily true that double precision computations will be slower. If the step size does not increase in a double precision solution when compared to single precision it may indicate that previous results were incorrect.¹⁶ Double precision arithmetic does utilize twice as much storage for data as single precision, and "half double precision" techniques are frequently implemented to conserve storage.

"Half double precision" techniques accumulate the integral as a double precision sum but convert the results to single precision before transmitting them to the simulation equations. This eliminates much of the roundoff error in the integration algorithms; however, roundoff error in the equations of the mathematical model may be severe. An interesting modification to this technique is illustrated in Figure 4. This technique restarts the integrator, along with a suitable readjustment of the initial conditions, whenever the increment added to the integral becomes small enough to border on the arithmetic limitations of the machine.

Sampling considerations

The selection of the integration step size, error

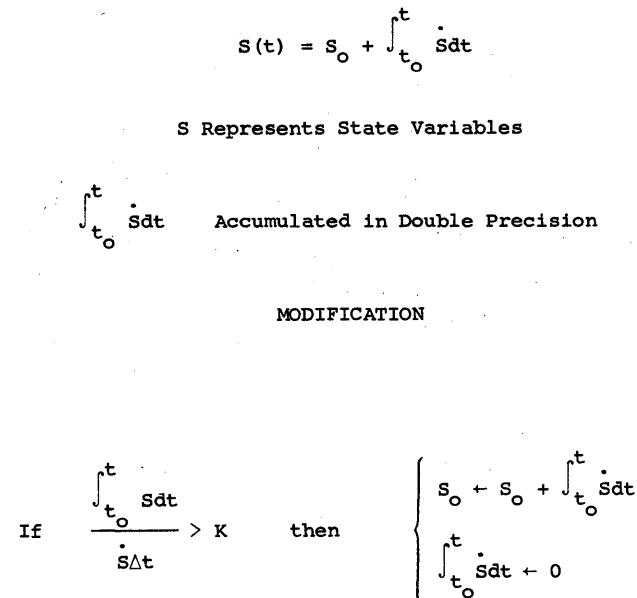


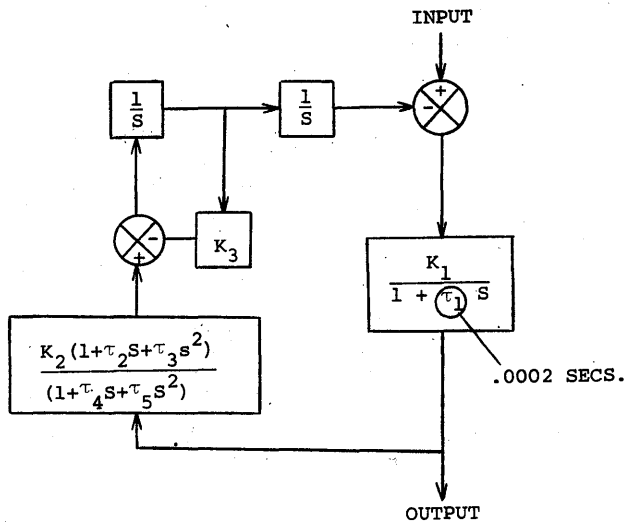
FIGURE 4—Half-double precision integration and modification

limits, and other integration parameters is an art bordering on black magic. Great care must be taken to account properly for high frequency effects in the mathematical model. The Shannon¹⁷ sampling theorem does not automatically apply to digital simulations' Shannon's theorem states that a function can be accurately reproduced if it is sampled from negative infinity to positive infinity with a frequency twice the highest frequency present in the function. The infinite range of the independent variable is impractical on a digital computer and rates of ten to one hundred times the highest frequency present are often necessary in closed loop simulations. Sampling rates of one hundred times the highest frequency may not seem severe in many applications, but this can be misleading.

Consider a digital simulation of a space vehicle in which the entire control system is simulated at the transfer function level. The highest frequency present in the vehicle may only be 5 cycles per second, however time constants in the control system components may be very small.⁹ This is illustrated in Figure 5. In this event, the step size must conform to the requirements for the smallest time constant present in the model—even though its effect on the vehicle may be negligible. This is one example in which weight factors applied to the integration error control are invaluable.

Stability

The stability of the integration algorithm must also be considered when selecting the algorithm and parameters. Generally speaking, an integration method is



ELEVATION CHANNEL OF TRACKER SERVO LOOP WITH SMALL TIME CONSTANT

ROLL RATE = 5 CPS ABOUT INTERCEPTOR ROLL AXIS

FIGURE 5—Relation of Gross system behavior to high frequency components in mathematical model

said to be stable if the error introduced at each step decreases with increasing number of steps. It is said to be relatively stable if the error introduced at each step grows more slowly than the solution.¹⁸ The stability of the solution is a function of the integration step size, the algorithms, and the time constants of the mathematical model.

In general, relative stability is the significant criteria in the numerical solution of differential equations.¹⁹ Although predictor equations may often be unstable, one usually chooses correctors which are stable. In this event, the instability of the predictor has little effect, and, if the corrector is iterated, the predictor's instability is irrelevant. Curiously it is generally more desirable to decrease the step size than to perform repeated iterations of the corrector.¹⁹

Real time?

The speed of the simulation is a critical factor. One often hears demands for (or claims of) "real time" speeds or "faster than real time" speeds. These speeds are certainly feasible in many applications, and, in fact, many hybrid computer simulations actually depend upon these speeds. However, it appears that many people assume categorically that they can obtain real time speeds in virtually every digital simulation they perform. Examples are given, results cited, and general-

ized programs are frequently demonstrated with real time speeds.

The speed of the simulation cannot be estimated until the natural frequency of the models are determined, integration algorithms selected, stability properties are studied, and intelligent decisions concerning the step size are made.

The detail of the mathematical model is also significant. Detailed models are certainly more difficult to simulate in real time than less detailed configurations.

The trap that many people fall into is characterized by "The Real Time Syndrome." Briefly, the Syndrome states: "It can be shown that any system can be simulated in real time with a suitable number of significant simplifications." The truth of this statement is self-evident and should be considered when making requests for, or claims of, real time speeds in a simulation.

Mathematics in simulation languages

Numerical techniques embedded in most simulation languages generally conform to the techniques described in earlier sections, i.e., algorithms drawn from the Runge-Kutta and predictor-corrector families. Earlier languages used simple algorithms;^{20,21,22} however the current languages generally use algorithms of at least fourth order.^{7,8,14} See, for example, Figure 6.

Simulation languages are usually enhanced by the inclusion of "canned" mathematical functions in addition to the integration routines. They include such things as transfer function blocks representing a variety of circuits, blocks representing nonlinearities such as dead zones, transport delays, etc.^{14,23} See Figure 7. Implicit algebraic loops present in the model equations are not allowed in the use of simulation languages and the languages usually incorporate iterative and binary search techniques which enable the user to obtain the necessary algebraic solutions.

LANGUAGE	ALGORITHM	VARIABLE STEP
DAS	RECTANGULAR	NO
MIDAS	5TH ORDER MILNE PREDICTOR CORRECTOR	YES
MIMIC	FOURTH ORDER RUNGE KUTTA	YES
HSL	VARIETY AVAILABLE ADAMS AND 4TH ORDER RUNGE KUTTA RECOMMENDED	YES
CSMP	7 AVAILABLE	YES
CSSL (RECOMMENDED)	VARIETY RECOMMENDED	YES

FIGURE 6—Some numerical integration algorithms

LANGUAGE	FUNCTION	EQUATION
DAS	BANG-BANG	$0 = i_2 \text{sign } i_1$
MIMIC	DERIVATIVE	$0 = \frac{di_2}{di_1}, 0(0)=i_3$
DSL/90	DELAY	$e^{-1, S}$
HSL	FIRST ORDER TRANSFER FTN.	$\frac{i_1 + i_2 S}{i_3 + i_4 S}$
CSMP/360	2ND-ORDER LAG-COMPLEX	$\frac{1}{S^2 + 2i_1 i_2 S + i_2 S^2}$
CSSL	IMPLICIT	BREAKS SORT LOOPS $0 = \text{IMPL}(i_1, i_2)$

INPUTS i_1, i_2, \dots, i_n
 OUTPUTS o_1, o_2, \dots, o_n

FIGURE 7—Some mathematical functions of simulation languages

The simultaneous nature of the differential equations describing continuous dynamic systems creates a burden on the integration systems design in simulation languages. Because the digital computer is a serial processor, it is necessary to construct an integration package to integrate all of the differential equations in a parallel manner. This is accomplished by using a centralized routine which integrates each derivative serially but holds the value of each integral until all integrals at a given time step are evaluated. Once all the integrals are evaluated, they are "simultaneously" updated, the independent variable is advanced, and the solution continues.

The problem of parallel integration can best be explained by an example. Consider the system of two simultaneous differential equations:

$$\dot{y} = f(t, y, z)$$

$$\dot{z} = g(t, z, y)$$

The numerical solution at $t + \Delta t$ is obtained by using some integration algorithm to evaluate

$$\dot{y}(t) = f(t, y(t), z(t))$$

$$y(t + \Delta t) = y(t) + \int_t^{t+\Delta t} \dot{y}(t) dt$$

$$\dot{z}(t) = g(t, z(t), y(t))$$

$$z(t + \Delta t) = z(t) + \int_t^{t+\Delta t} \dot{z}(t) dt.$$

The problem appears in the evaluation of

$$\dot{z}(t) = g(t, z(t), y(t))$$

If $y(t)$ is a specific storage location (usually the first location of an array) the value used in the calculation of the derivative $\dot{z}(t)$ cannot be the result of the integral since this represents the value $y(t + \Delta t)$, not $y(t)$. The centralized integration subroutine, referenced in the DERIVATIVE section of a simulation language source program, will update all of the solutions in parallel.²⁴ See Figure 8.

An alternative method employed in some programs²³ is to start each computational pass by evaluating all of the integrals. This advances the solution of the state variables to the next point in time. It is then necessary to compute the derivatives of the state variables for the given point in time.

Using either technique, the parallel nature of the problem creates programming complications. Arrays

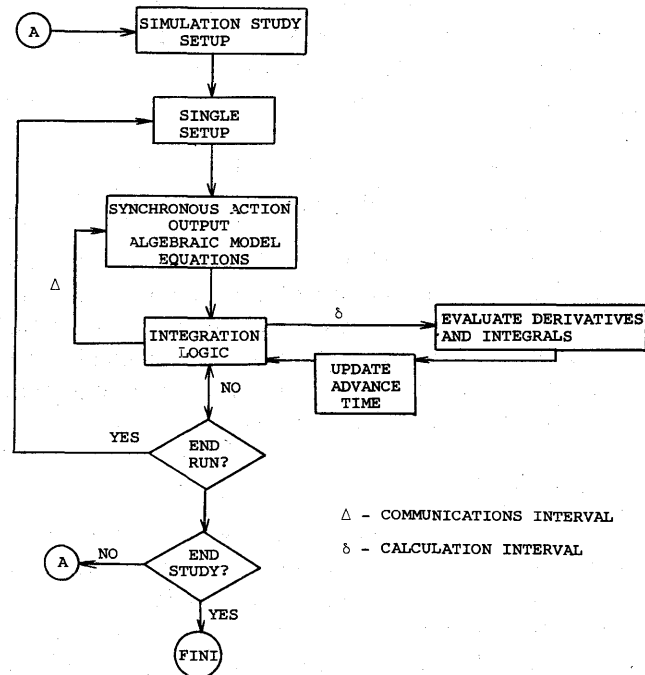
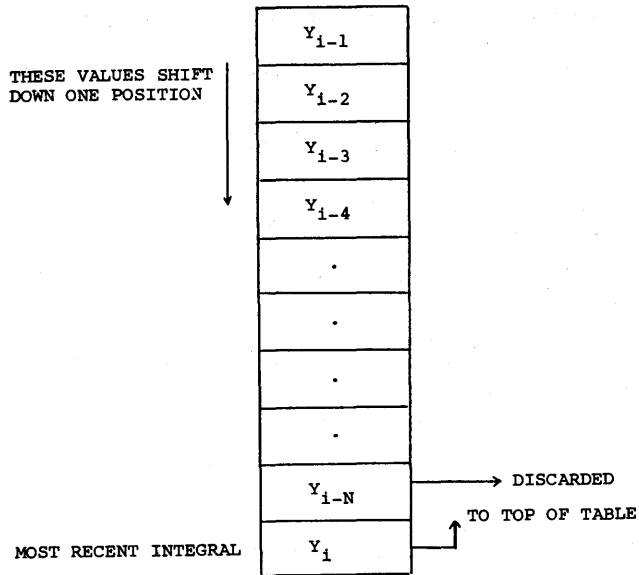


FIGURE 8—Integration systems design



Y_i indicates value of the solution calculated on the i -th pass. The integration routine places Y_i at the bottom of the table until time is advanced. During the UPDATE phase the value is moved to the top of the table.

FIGURE 9—Update table

used to store current and past solutions are in a constant state of fluctuation. The buffers used to store the integrals temporarily also are used for retaining previous information describing the solution for the predictor-corrector algorithms, and for storing intermediate points in the Runge-Kutta techniques. The number of locations occupied by each of the arrays is a function of the order of the integration algorithm. Buffers are required for both the derivative and solution values.

Figure 9 illustrates a typical "UPDATE TABLE."²⁵ The table is originally filled with Runge-Kutta values and then maintained by a predictor-corrector algorithm. Sufficient points are usually maintained to continue integrating if the step size is increased without requiring a Runge-Kutta restart.

SUMMARY

Mathematical techniques embedded in simulation applications cover virtually the entire mathematical spectrum from numerical analysis through classical analysis to algebra and the theory of numbers. It is clearly impossible to address all of these topics in a single paper, however this paper summarizes the common mathematical questions encountered in most general simulations. Special techniques, such as those dis-

cussed in the introduction, and others, for example discretization of continuous systems into sampled data systems using z transforms, adaptive filters, etc.,²⁶ are not commonly employed in simulation applications and almost never in simulation languages.

Selection of languages, integration algorithms, single or double precision, sampling rates and integration parameters are all perplexing problems at present, but these are the questions the engineer must address prior to simulation of a continuous dynamic system on a digital computer. Although a substantial amount of work is being done to attack these problems, it is the author's prediction that effective elimination of these numerical problems will come about sooner by the development of improved hardware than by the development of improved analytical and computational techniques.

REFERENCES

- 1 J C STRAUSS
Digital simulation of continuous dynamic systems: An overview
AFIPS Conference Proceedings Vol 33 1968 FJCC 1968
- 2 H H TRAUBOTH
Digital simulation of general control systems
SIMULATION 9 5 1987
- 3 W H ANDERSON R B BALL
A numerical method for solving control differential equations on digital computers
J ACM 7 1 1960
- 4 L A NICHOLLS
Some aspects of the digital simulation of a surface-to-air guided missile
Technical Note SAD 126 Weapons Research Establishment Salisbury So Australia 1963
- 5 J C STRAUSS
Simulation languages and HSL
EAI Research and Advanced Development Report 67-2 1967
- 6 D D McCRACKEN W S DORN
Numerical methods and Fortran programming
John Wiley and Sons Inc New York 1964 chapter 11 p 365
- 7 F J SANSOM H E PETERSEN
MIMIC Programming Manual
USAF AFSC SEG-TR-67-31 1967
- 8 HYTRAN simulation language programming manual
EAI Publication No 0780000060 1967
- 9 D H BRANDIN K JAKSTAS P STRAMESE
Digital simulation of guidance and control for exoatmospheric intercept
Supplement to the IEEE Transactions on Aerospace and Electronic Systems AES-2 4 1966
- 10 R C BROWN R V BRULLE G D GRIFFEN
Six-degree-of-freedom flight-path study generalized computer program
USAF AFSC WADD-TR 60-781 1961
- 11 T E HULL A L CREEMER
Efficiency of predictor-corrector procedures
J ACM 10 3 1963
- 12 *Handbook of mathematical functions*
U.S. Department of Commerce NBS Applied Mathematics Series 55 1964 p 896

- 13 J B ROSSER
A Runge-Kutta for all seasons
 SIAM Review 9 3 1967
- 14 J C STRAUSS
The SCi continuous systems simulation language
 SIMULATION 9 6 1967
- 15 R B AGNEW
Differential equations
 McGraw-Hill Book Co Inc New York 1960 2nd Edition
- 16 D H BRANDIN
Numerical integration and digital simulation of continuous systems
 SIMULATION 9 4 1967 Technical Note
- 17 C E SHANNON
 Proc IRE Vol 37 1949 pp 10-21
- 18 A RALSTON H S WILF
Mathematical methods for digital computers
 John Wiley and Sons Inc New York 1967 Volume 2
- 19 R W HAMMING
Numerical methods for scientists and engineers
 McGraw-Hill Book Company Inc New York 1962
- 20 R A GASKILL J W HARRIS A L McKNIGHT
DAS-a digital analog simulator
 AFIPS Conference Proceedings Vol 23 1963 SJCC p 69
- 21 J R HURLEY
Digital simulation I: DYSAC A digitally simulated analog computer
 AIEE Summer General Meeting Denver Colorado 1962
- 22 R D BRENNAN R N LINEBARGER
A survey of digital simulation: digital analog simulator programs
 SIMULATION 7 12 1964
- 23 R D BRENNAN M Y SILBERG
Two continuous system modeling programs
 J IBM Systems 6 4 1967
- 24 F J SANSOM
Continuous system simulation languages
 AFIPS Conference Proceedings Vol 33 1968 FJCC 1968
- 25 D H BRANDIN
Simulation of interceptor missiles on an IBM 360/50 digital computer
 N00123-67-C-0547 Naval Ordnance Laboratory Corona Calif 1967
- 26 J S ROSKO
Improved techniques for digital modelling and simulation of nonlinear systems
 AFIPS Conference Proceedings Volume 32 1968 SJCC 1968

SALEM-A programming system for the simulation of systems described by partial differential equations

by STANLEY M. MORRIS and WILLIAM E. SCHIESSER

Lehigh University
Bethlehem, Pennsylvania

INTRODUCTION

Digital simulation has attained a broad and enthusiastic usage in recent years, due largely to the increased speed and flexibility of digital computer hardware and the notable efforts of the authors of simulation programs such as MIMIC,¹ DSL-90,² PACTOLUS,³ and many others. All of these programs perform basically a single task: they solve a system of ordinary differential equations of the initial value type. Indeed, the expression "digital simulation of continuous systems" has become synonymous with this definition.

Users of these programs can study complex physical processes with little knowledge of computer programming or numerical analysis. Logically, they are never far removed from the differential equations, but they have all of the capability of the digital computer at their disposal.

SALEM extends these concepts to partial differential equations, which always arise in the mathematical modeling of distributed parameter systems. Most physical systems are of the distributed type; that is, the dependent variables are functions of more than one independent variable. The ordinary differential equations commonly encountered in mathematical models are usually simplifications of the partial differential equations which actually describe the system.

A partial differential equation (PDE) simulator would enable a user to investigate distributed systems with the ease that he may presently investigate a discrete or ordinary differential equation (ODE) system. The problems involved in actually writing a PDE simulator, however, are far more complex and varied than those involved in an ODE simulator. There is no one universal method for numerically solving all the PDE's commonly found in engineering problems. Each different problem may require a unique method, a possibility commonly attested to by authors of standard texts in the field.

Therefore, SALEM is organized into two parts: an executive program capable of translating user-generated descriptions of partial differential equations into data usable by a computer, and a library of calculation routines which produce the solution.

SALEM input format

Every attempt has been made to allow a free-form algebraic input format. If we are interested in the solution to the equation:

$$2\partial^2 C / \partial x^2 - \partial C / \partial t + 1/2C = 1$$

we would program it as:

$$(2.0)*D2(C, X) - D1(C, T) + (0.5)*C = 1.0$$

Note that D2(C,X) refers to the second derivative of C with respect to X. The coefficients may also be algebraic expressions or functions, expressed in Fortran notation.

$$\partial^2 C / \partial R^2 + (2.0/R)\partial C / \partial R - \partial C / \partial t = 0$$

$$D2(C, R) + (2.0/R)*D1(C, R) - D1(C, T) = 0$$

Initial and/or boundary conditions are expressed in an analogous fashion.

$$\partial C / \partial x + C = 1, x = 0$$

$$B\text{OUNDARY C\text{O}NDITI\text{O}N, D1(C, X) + C = 1.0, X = 0.0$$

$$C = 1.5, t = 0 \text{ (t is time)}$$

$$I\text{NITIAL C\text{O}NDITI\text{O}N, C = 1.5, T = 0.0$$

There are other statements which define (1) the

values of the independent variables for which the solution is calculated and printed out, (2) the desired stopping point for open-ended equations, and (3) special statements which allow the calculation of complex equation coefficients.

Table 1 lists the types of equations which can presently be solved by SALEM, along with allowable boundary and initial conditions.

Sample problem

A simple sample problem will serve to illustrate the procedure involved in obtaining a solution to a PDE. Figure 1 illustrates a steel sheet which is initially at a uniform temperature of 100°. Two opposite sides are insulated, and the other two sides are suddenly raised to 200°K. The following data can be used:

$$k = 25.6 \text{ Btu/hr ft}^{\circ}\text{F (thermal conductivity)}$$

$$D = 450 \text{ lb/ft}^3 \text{ (density)}$$

The dimensions of the sheet are 1 ft X 1.2 ft, with the short sides being maintained at 200°. We will consider two cases:

$$C_p = 5.145 \text{ Btu/lb}^{\circ}\text{F (average heat capacity)}$$

$$C_p = 3.37 + 0.0071(T) \text{ (temperature-dependent heat capacity)}$$

$$k/DC_p(\partial^2 T/\partial x^2 + \partial^2 T/\partial y^2) - \partial T/\partial t = 0$$

$$\partial T/\partial x = 0., x = 0$$

$$\partial T/\partial x = 0., x = 1.0$$

$$T = 200., y = 0$$

$$T = 200., y = 1.2$$

$$T = 100., t = 0$$

The equation would be linear for case 1 and non-linear for case 2.

The SALEM input listing for case 1 is shown in Figure 2. Card #1 is the partial differential equation, where T is temperature and TM is time. Cards 2-6 specify the boundary and initial conditions. Card #7 specifies the print interval for each independent variable; X=.2 means that temperatures are to be calculated at X=0, .2, .4, .6, .8, and 1.0; Y=.2 means that temperatures are to be calculated at y = 0, .2, .4, .6, .8, 1.0, and 1.2 for each value of x. This specifies a grid of 6 X 7 = 42 points. TM = .1 specifies a complete solution for each time unit of 0.1. Card #8 indicates that the PDE is parabolic. Card #9 specifies the final value of time for which a solution is to be produced. Card #10 indicates the calculation program which is to be used to solve this PDE; the number is obtained from a standard library list.

Figure 3 shows the SALEM-produced solution at

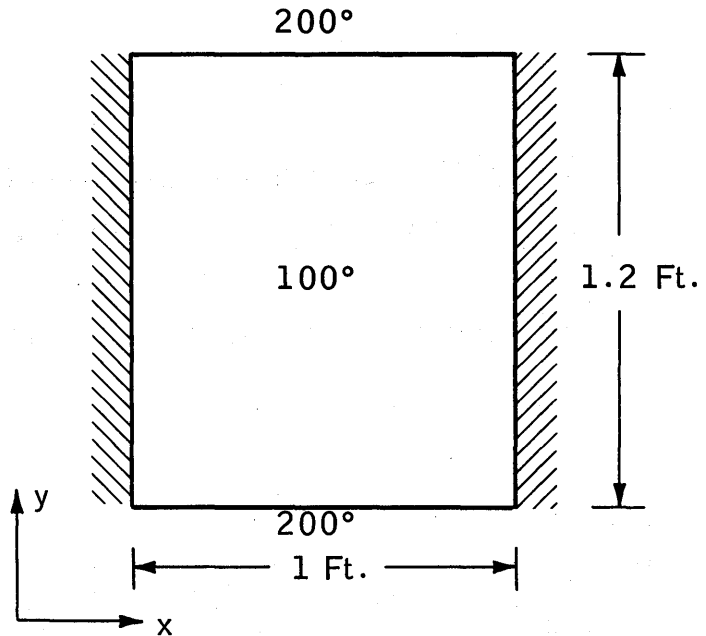


FIGURE 1—Steel sheet

SALEM input list

1. (.62)*D2(T,X) + (.62)*D2(T,Y) - D1(T, TM) = 0
2. BOUNDARY CONDITION, D1(T,X) = 0., X = 0.
3. BOUNDARY CONDITION, D1(T,X) = 0., X = 1.
4. BOUNDARY CONDITION, T = 200., Y = 0.
5. BOUNDARY CONDITION, T = 200., Y = 1.2
6. INITIAL CONDITION, T = 100., TM = 0.
7. PRINT INTERVAL, X = .2, Y = .2, TM = .1
8. TYPE PARABOLIC
9. SOLUTION END, TM = .2
10. USE SUBROUTINE NUMBER 20
11. END

FIGURE 2—SALEM input list—case 1

SALEM solution for TM = 0.

 X					
	0	.2	.4	.6	.8	1.0
Y = 0	150.0	150.0	150.0	150.0	150.0	150.0
Y = .2	100.0	100.0	100.0	100.0	100.0	100.0
Y = .4	100.0	100.0	100.0	100.0	100.0	100.0
Y = .6	100.0	100.0	100.0	100.0	100.0	100.0
Y = .8	100.0	100.0	100.0	100.0	100.0	100.0
Y = 1.0	100.0	100.0	100.0	100.0	100.0	100.0
Y = 1.2	150.0	150.0	150.0	150.0	150.0	150.0

SALEM SOLUTION FOR TM = .2

 X					
	0	.2	.4	.6	.8	1.0
Y = 0	200.0	200.0	200.0	200.0	200.0	200.0
Y = .2	173.0	173.0	173.0	173.0	173.0	173.0
Y = .4	152.7	152.7	152.7	152.7	152.7	152.7
Y = .6	145.8	145.8	145.8	145.8	145.8	145.8
Y = .8	152.7	152.7	152.7	152.7	152.7	152.7
Y = 1.0	173.0	173.0	173.0	173.0	173.0	173.0
Y = 1.2	200.0	200.0	200.0	200.0	200.0	200.0

FIGURE 3—SALEM solution—case 1

time $t = 0$ and $.2$. The solution at $t = 0$ provides a check on the initial conditions. Note that the temperature gradient is zero in the y -direction; this is expected because of the perfect insulation at $x = 0$ and $x = 1$.

Figure 4 shows the SALEM input listing for case 2. Note that the constant 0.62 has been replaced by the variable A in the equation coefficients. A is defined in card 11, a special ARITH statement. The remainder of the listing is identical to case 1, with the exception of the subroutine (card 10). 20N refers to the nonlinear version of subroutine #20. Figure 5 shows the SALEM-produced solution for case 2.

Table I—Classes of equations presently solved by SALEM

One dimensional parabolic

$$a_1 \partial^2 C / \partial x^2 + a_2 \partial C / \partial x - a_3 \partial C / \partial t + a_4 C = a_5$$

One dimensional hyperbolic:

$$a_1 \partial^2 C / \partial x^2 - a_2 \partial^2 C / \partial t^2 = a_3$$

Two dimensional parabolic

$$a^2 \partial^2 C / \partial x^2 + a_2 \partial^2 C / \partial y^2 + a_3 \partial C / \partial x - a_4 \partial C / \partial t + a_5 C = a_6$$

elliptic:

$$a_1 \partial^2 C / \partial x^2 + a_2 \partial^2 C / \partial y^2 + a_3 \partial C / \partial x + a_4 C = a_5$$

The coefficients can be zero, constant, or functions of any dependent or independent variable. This means that nonlinearities of the form $f(C) \partial C / \partial y$ are allowed. Allowable boundary and initial conditions:

$$a_1 \partial C / \partial x + a_2 C = a_3, x = a_4$$

a_1, a_2 , and a_3 can be zero, constant, or functions of the independent variable. a_4 must be zero or constant.

Rectangular, cylindrical, and spherical coordinate systems may be used.

SALEM input

1. (A)*D2(T,X) + (A)*D2(T,Y) - D1(T, TM) = 0.
2. BOUNDARY CONDITION, D1(T,X) = 0., X = 0.
3. BOUNDARY CONDITION, D1(T,X) = 0., X = 1.
4. BOUNDARY CONDITION, T = 200., Y = 0.
5. BOUNDARY CONDITION, T = 200., Y = 1.2
6. INITIAL CONDITION, T = 100., TM = 0.
7. PRINT INTERVAL, X = .2, Y = .2, TM = .1
8. TYPE PARABOLIC
9. SOLUTION END, TM = .2
10. USE SUBROUTINE NUMBER 20N
11. ARITH, A = 3.19/(3.37 + .0071*T)
12. END

FIGURE 4—SALEM input list—case 2

SALEM solution for TM = .2

 X					
	0	.2	.4	.6	.8	1.0
Y = 0.	200.0	200.0	200.0	200.0	200.0	200.0
Y = .2	176.7	176.7	176.7	176.7	176.7	176.7
Y = .4	159.7	159.7	159.7	159.7	159.7	159.7
Y = .6	153.6	153.6	153.6	153.6	153.6	153.6
Y = .8	159.7	159.7	159.7	159.7	159.7	159.7
Y = 1.0	176.7	176.7	176.7	176.7	176.7	176.7
Y = 1.2	200.0	200.0	200.0	200.0	200.0	200.0

FIGURE 5—SALEM solution—case 2

SALEM also possesses a full complement of error diagnostics to facilitate usage. Errors in problem formulation and programming, and problems encountered in the numerical solution (i.e., instability and convergence failure) automatically terminate the execution. An explanatory comment is provided for the user.

Methods used in calculation programs

A brief description of calculation techniques will be given. Every effort was made to use simple, general methods in developing the library of subroutines, both to guarantee convergence and for programming simplicity. As a first example, we will consider the method used to solve the equation:

$$a_1 \partial^2 C / \partial x^2 - a_2 \partial C / \partial t = F \quad (1)$$

$\partial^2 C / \partial x^2$ is replaced with a second order central difference representation, and $\partial C / \partial x$ is replaced with a first order backward difference representation.

$$\begin{aligned} \partial^2 C / \partial x^2 &= (C_{r+1,s+1} - 2C_{r,s+1} + C_{r-1,s+1}) / h^2 + 0(h^2) \\ \partial C / \partial t &= (C_{r,s+1} - C_{r,s}) / k + 0(k) \end{aligned}$$

r refers to x, s refers to t

Substituting these representations into equation (1), we obtain the following implicit finite difference equation:

$$\begin{aligned} B C_{r+1,s+1} - (2B + 1) C_{r,s+1} + B C_{r-1,s+1} &= C_{r,s} + F' \\ B &= a_1 k / a_2 h^2 \\ F' &= F k / a_2 \end{aligned} \quad (2)$$

If we had nonderivative boundary conditions:

$$\begin{aligned} C &= a_3, x = 0 \\ C &= a_4, x = 1 \\ C &= a_5, t = 0 \end{aligned}$$

we could write equation (2) for each internal mesh point $C_{r,s+1}$ $0 < r < 1$. If there are N internal mesh points,

this will result in N equations in N unknowns. The set of equations is tridiagonal and can be solved explicitly by the method of Thomas.⁴ This procedure is repeated for each time step.

If we have the general boundary condition:

$$a_3 \partial C / \partial x + a_4 C = a_5$$

we can replace the derivative term with a second-order difference representation:

$$a_3 \frac{(C_{r+1,s+1} - C_{r-1,s+1})}{2h} + a_4 C_{r,s+1} = a_5 \quad (3)$$

In this case, we do not directly know the boundary value, so we must write equation (2) for each mesh point, including boundary points. But this requires knowledge of the value of points outside the boundary. These can be calculated from equation (3):

$$C_{r+1,s+1} = 2ha_5/a_3 - (2ha_4/a_3)C_{r,s+1} + C_{r-1,s+1}$$

Therefore, for each time increment, we will have $N+2$ tridiagonal equation in $N+2$ unknowns, which can be solved by the method of Thomas.

Finally, to increase the speed and accuracy of the calculated solution, we use truncation error correction (often referred to as the "deferred approach to the limit"). Examining the error due to replacing the time derivative with a finite difference representation, we can write:

$$\begin{aligned} \text{calculated answer} + \text{truncation error} \\ = \text{actual answer} \end{aligned} \quad (4)$$

We can estimate the truncation error in the time direction from Taylor series:

$$\text{truncation error} = \frac{k}{2} \frac{\partial^2 C}{\partial t^2} (\eta) \quad t_0 \leq \eta \leq t_0 + k$$

Arbitrarily let k , the step size in the time direction, equal one.

$$\text{truncation error (k = 1)} = \frac{1}{2} \frac{\partial^2 C}{\partial t^2} (\eta) \quad (5)$$

For $k = 2$:

$$\text{truncation error (k = 2)} = \frac{2}{2} \frac{\partial^2 C}{\partial t^2} (\eta') \quad (6)$$

From (4), (5), and (6) we can develop two more equations:

$$\begin{aligned} &\text{calculated answer (k = 1)} \\ &+ \text{truncation error (k = 1)} \\ &= \text{calculated answer (k = 2)} \\ &+ \text{truncation error (k = 2)} \end{aligned} \quad (7)$$

$$\frac{\text{truncation error (k = 1)}}{\text{truncation error (k = 2)}} = \frac{1}{2} \frac{\partial^2 C(\eta) / \partial t^2}{\partial^2 C(\eta') / \partial t^2} \quad (8)$$

If the calculated answers for $k = 1$ and $k = 2$ are sufficiently close, then $\partial^2 C(\eta) / \partial t^2 = \partial^2 C(\eta') / \partial t^2$, and combining (6) and (7):

$$\begin{aligned} &\text{calculated answer (k = 1)} \\ &- \text{calculated answer (k = 2)} \\ &= \text{truncation error (k = 1)} \end{aligned}$$

This leads to a powerful solution technique:

1. Calculate the solution using the time and distance intervals specified by the user.
2. Halve the time interval, and recalculate the solution using two time steps.
3. If the solution from step 1 and step 2 differ by at most a specified error, proceed to step 4; otherwise, halve the time step again and repeat.
4. Calculate the truncation error and add this to the most recently calculated solution to obtain the final answer.

By an analogous procedure, we can estimate the truncation introduced by replacing the x -derivative with the second-order finite difference representation

$$\begin{aligned} &\text{truncation error (h = 1)} \\ &= [\text{calculated answer (h = 1)} \\ &- \text{calculated answer (h = 2)}] / 3. \end{aligned}$$

A study has been made to determine the efficiency of this procedure relative to the same procedure without truncation error correction. The following problem was used:

$$\begin{aligned} 2\partial^2 V / \partial x^2 - \partial V / \partial t &= 0 \\ V &= 0, t = 0 \\ V &= t, x = -2 \\ V &= t, x = 2 \end{aligned}$$

We desire a solution for five time steps of $\Delta t = .1$. Several runs were made with $\Delta x = 0.8$, one with truncation error correction, and several with no truncation error correction, at decreasing values of Δt . The calculated answers at $t = .5$ are tabulated for $x = -1.2$ and -1.4 in Table 2.

After expending three times as much computation time as the version with truncation error, the answer is still in error by 5%, compared to 0.3% for the corrected

Table II—*Truncation error correction study*

	Results		IBM 360/30 Calculation time, sec.
	x = -1.2	x = -.4	
Exact solution	.1875	.0696	
Salem with truncation error correction	.1876	.0698	11
Salem without truncation error correction			
$\Delta t = .1$.2016	.0904	2
$\Delta t = .05$.1955	.0816	3
$\Delta t = .025$.1924	.0770	5
$\Delta t = .0125$.1908	.0746	9
$\Delta t = .003125$.1896	.0728	36

solution. Even if Δt were much smaller, a 3% error would exist, due to the truncation error in the x direction. Reducing the mesh size to $\Delta x = .4$ and retaining $\Delta t = .003125$ reduced the error to less than 1%, but the solution time was 144 seconds, or 13 times as long as the method with truncation error correction. For long, complex problems this advantage could be enormous.

All one-dimensional parabolic and hyperbolic PDE's are solved using variations of the above techniques. Two-dimensional parabolic and elliptic equations are solved using the Peaceman-Rachford method, which is documented elsewhere. In general, the method is an alternating-direction technique which results in sets of tridiagonal equations. The truncation error correction technique has been adapted to the Peaceman-Rachford equations, and has resulted in increased speed of solution while maintaining a given accuracy.

SUMMARY

SALEM is more than just a programming system for the automated solution of PDE's. It is a flexible simulation language with many user-oriented programming features. It is also modular in construction so that once a numerical technique is successfully developed for a particular type of PDE, a subroutine implementing the numerical algorithm can easily be added to the system. Hopefully the library of subroutines will be expanded and shared by the users, thereby reducing the duplication of effort in the development of numerical methods for the solution of PDE's.

Future research

Development of SALEM is continuing at Lehigh

University. Emphasis is being placed on the following areas, which currently limit the applicability of SALEM:

- Irregular geometries.
- First order flow equations, where convection terms are more important than diffusion terms.
- Coupled partial differential equations.
- Nonlinear boundary conditions.
- Nonlinear terms of the form $\frac{\partial}{\partial x}(D(C)\partial C/\partial x)$ and $\frac{\partial}{\partial x}(D(C)C)$

As of this writing, these additional features are being incorporated; and the program will be withheld until the changes are complete. The authors will publish papers relating to applications of SALEM in the near future, and these will demonstrate how simply a novice can use the system.

More information concerning SALEM is available in reference 6, or by contacting:

S. M. Morris
Esso Research and Engineering Co.
P. O. Box 101
Florham Park, New Jersey 07932
Phone: 201-474-6379

or:

W. E. Schiesser
Department of Chemical Engineering
Lehigh University
Bethlehem, Pennsylvania 18015
Phone: 215-867-5071, ext. 229

BIBLIOGRAPHY

- 1 H E PETERSON F J SANSOM L M WARSHAWSKY
MIMIC—A digital simulator program
SESCA Internal Memo 64-12 Wright-Patterson Air Force Base Ohio 1964
- 2 R L LINEBARGER
DSL/90
Simulation September 1966
- 3 R D BRENNAN H SANO
Pactolus—A digital simulation program for the IBM 1620
1964 Fall Joint Computer Conference AFIPS Conference Proceedings 26 1964
- 4 L LAPIDUS
Digital computation for chemical engineers
McGraw-Hill Book Company New York 1962
- 5 D W PEACEMAN H H RACHFORD JR
The numerical solution of parabolic and elliptic differential equations
JSIAM 3 1955
- 6 S M MORRIS
SALEM—A programming system for the simulation of systems described by partial differential equations
PhD Dissertation Department of Chemical Engineering Lehigh University available from University Microfilms Inc 300 North Zeeb Road Ann Arbor Michigan 48103

TAF—A steady state, frequency response, and time response simulation program[†]

by THOMAS E. SPRINGER and OTIS A. FARMER

University of California
Los Alamos, New Mexico

INTRODUCTION

For over a decade, engineers have used the analog computer to simulate systems, models of which could be represented by groups of nonlinear simultaneous first-order differential equations. As the digital computer became more prominent, it was often used to cross check, to set up, and to solve problems run on the analog computer. Many digital codes were developed to complement, check out, and even replace the analog computer. Codes such as DAS, MIDAS, PARTNER, DYSAC, MIMIC, DSL/90, and others came into wide use, and some of the reasons for their development have been ably reviewed by Brennan and Linebarger* and later by Clancy and Fineberg.**

Nuclear rocket engine development at the Los Alamos Scientific Laboratory led to use of digital computer programs in conjunction with analog computer programs. The equations solved were those representing the propellant system, the heat exchanger, the reactor kinetics, and the controls of nuclear rocket engines tested at the Nuclear Rocket Development Station in Nevada. The analog computer simulation was checked by means of a digital computer. Originally this was done by programming and solving digitally, first, the steady-state equations representing the system and, second, the linearized, La Place transformed differential equations representing the system. The steady-state results from the first code and the transfer func-

tions from the second code were checked with results from the analog computer, and corrections were made until both computer results agreed. The first code required solving a set of non-linear algebraic equations, usually by iteration. The second code required solving a complex matrix at each frequency for which the transfer functions were to be evaluated. It soon became desirable to obtain time solutions on the digital computer. To do this, the derivatives of the system were defined, the initial conditions were obtained by solving the non-linear steady-state equations, and the equations were integrated by using a fourth-order Runge-Kutta digital integration method. Most of the various subsystem studies involved programming three separate codes as well as linearizing and LaPlace transforming by hand the equations.

Of the three classes of digital solutions—steady state, frequency response, and time response—the last was the easiest to program. It was only necessary to define the derivatives of the system in any order, and to integrate the derivatives with the Runge-Kutta subroutine. However, the programming of FORTRAN formats and of read-and-write statements for defining input-output data remained a necessary chore for all three classes of solutions.

As the number of problems increased, so did the need for a flexible digital code that would quickly solve for steady state, frequency response, and time response. The only information needed to obtain these three classes of digital solutions are the derivatives of the system and their inputs. This led to the development of the TAF (Time and Frequency) code in which the derivatives of the system are defined in one FORTRAN subroutine called DER; the variables are identified, and certain preliminary calculations are performed in a second user-written subroutine called INPUT. The TAF code reads input data; finds steady state, transfer

*Work performed under the auspices of the U.S. Atomic Energy Commission.

**R. D. Brennan and R. N. Linebarger, "A Survey of Digital Simulation: Digital Analog Simulation Programs," *Simulation*, Vol. 3, No. 6, (December, 1964).

**J. J. Clancy and M. S. Fineberg, "Digital Simulation Languages: A Critique and Guide," *Fall 1965 Joint Computer Conference*, Vol. 27, Part I, Spartan Books, Inc. (1965)

functions, and time response; and lists or plots on micro-film the outputs.

The digital codes mentioned earlier (DAS, MIDAS, etc.) that simulate the analog computer do not compute transfer functions (unless a sine wave is fed into the system) and must integrate to find a steady-state solution. However, many of these codes do not require an understanding of digital computation. The user, as a rule, only defines the analog computer elements and all the inputs to these elements. For an engineer unacquainted with digital coding but familiar with analog computing, this offers an immediate advantage over the TAF code, which requires some knowledge of FORTRAN. However, in defining a problem, the user must first describe the system in differential equations. If these are partial differential equations and if techniques are to be used that only allow ordinary differential equations, the original equations must be reduced into a set of first-order, non-linear differential equations. For us it appeared simpler to enter these equations directly into the computer instead of to enter an analog flow chart from which the computer extracts the original equations. Admittedly, the construction of flow charts may point out many mistakes; however, needless effort may be expended in their preparation if the equations themselves can be entered directly into the computer.

The TAF code is not a new simulation language. It is written in FORTRAN and requires that the user define the derivatives of the system in a FORTRAN subroutine called DER. The programmer may use the integration variables and may define the derivatives in any order in programming the DER subroutines. However, any other variables the programmer chooses to use in algebraic equations must be defined before they are used. It would be possible, of course, to define the system model to be simulated by defining the equivalent analog computer wiring diagram using, for example, the Continuous System Simulation Language.* A user only familiar with analog computers could then easily obtain fast digital steady-state solutions and frequency response solutions as well as the time response solutions that other codes presently give him. For the reason mentioned in the previous paragraph, however, we chose not to provide a new simulation language.

DISCUSSION

Approach

The basic criterion in developing the TAF code was the ability to compute steady-state, frequency re-

sponse and time response from one group of non-linear simultaneous first-order differential equations. The code is intended for initial value problems only. The equations can be described by

$$\frac{dy_i}{dt} = f_i(y_1, y_2, \dots, y_j, \dots, y_n, x_1, \dots, x_k, \dots, x_m) \quad (1)$$

where y_i is a set of integration or state variables, and x_k is a set of system inputs defined as functions of time. The system-defining equations themselves are assumed to be already contained in Equation 1.

Integration codes for steady-state and time-response calculations from differential equations are in common use, but do not include frequency response. We had to develop, therefore, a method of calculating frequency response from the same set of equations. This method uses a complex matrix whose elements are automatically defined.

Assume a set of steady-state variables, y^o_i , for which all $\frac{dy_i}{dt} = 0$. To compute transfer functions, we linearize the equations for a given set of inputs, x_k .

$$\frac{d}{dt} \delta y_i = f_i(y_j^o, x_k) + \sum_j \frac{\partial f_i}{\partial y_j} \delta y_j + \sum_k \frac{\partial f_i}{\partial x_k} \delta x_k \quad (2)$$

where δy_i is the variation in y_i from steady-state when the input is varied by δx_k . By deleting the steady-state terms and by forming the La Place transform of δy_i and δx_k , we obtain the matrix equation, shown below for three variables and two inputs. Capital letters denote the LaPlace transform.

$$\begin{bmatrix} \frac{\partial f_1}{\partial y_1} - s & \frac{\partial f_1}{\partial y_2} & \frac{\partial f_1}{\partial y_3} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} - s & \frac{\partial f_2}{\partial y_3} \\ \frac{\partial f_3}{\partial y_1} & \frac{\partial f_3}{\partial y_2} & \frac{\partial f_3}{\partial y_3} - s \end{bmatrix} \begin{bmatrix} \delta Y_1 \\ \delta Y_2 \\ \delta Y_3 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} \delta X_1 & \frac{\partial f_1}{\partial x_2} \delta X_2 \\ \frac{\partial f_2}{\partial x_1} \delta X_1 & \frac{\partial f_2}{\partial x_2} \delta X_2 \\ \frac{\partial f_3}{\partial x_1} \delta X_1 & \frac{\partial f_3}{\partial x_2} \delta X_2 \end{bmatrix} \quad (3)$$

*J. C. Strauss, et al, "The SCI Continuous System Simulation Language (CSSL)," SIMULATION, Vol. 9, No. 6, December, 1967.

If $\delta X_k = 1$, then δY_i will be the transfer function $\frac{\delta Y_i}{\delta X_k}(s)$. With $j\omega$ substituted for s , the complex matrix may be solved for various specified values of ω .

Equation 3 represents the linearized Laplace transformed system equations. The unique feature of the TAF code is the ability to evaluate the elements of this matrix equation automatically. The derivatives of the system, f_i , may be "reasonably" non-linear. By "reasonably" we imply the same conditions that are generally used when one applies linear control theory to non-linear systems. If the system has abrupt discontinuities such as stops on the controllers or a branch of equations at certain points, the frequency response should not be evaluated in the immediate vicinity of these points. This is reasonable since the Laplace transform is only defined for linear systems.

The computer evaluates the partial derivatives with respect to y_j by the numerical formula

$$\frac{\partial f_i}{\partial y_j} = \frac{f_i(y_1^0, \dots, 1.003y_j^0, \dots, y_n^0, x_k) - f_i(y_1^0, \dots, y_j^0, \dots, y_n^0, x_k)}{\Delta y_j} \quad (4)$$

in which $\Delta y_j = .003y_j^0$. The derivative $\frac{\partial f_i}{\partial x_k}$ is evaluated similarly with $\Delta x_k = .003x_k$. The choice of .003 is arbitrary. For most of the engineering problems we have programmed, it has proved to be a good compromise between derivative non-linear effects and digital computer round-off error. The former occurs if Δy_i is too large; the latter if Δy_i is too small. In the previous paragraph, we mentioned avoiding the "immediate vicinity" of abrupt discontinuities. By "immediate vicinity" we, therefore, mean that no state variable shall be within 0.3 percent of the discontinuity.

If $y_j^0 = 0$, then $\Delta y_j = .003$ and similarly for x_k^0 . Because these partial derivatives are evaluated at steady state, one might assume that the second term in the numerator of Equation 4 could be eliminated since it equals zero. However, in general, the term is not exactly zero, and its omission can lead to large errors.

The transfer functions just described were computed about a steady-state that is consistent with the inputs to the system. The TAF code offers two possibilities for computing steady-state: (1) by integrating until the system achieves equilibrium, and (2) by a modified Newton-Raphson method. In general, the first possibility requires a much longer computing time.

The second method was developed from Equation 2. Here, y_j^0 implies an initial guess as to the steady-state

condition, and δy_j implies an error correction to y_j to set the time derivatives to zero. In Equation 2, both the left-hand side and δx_k are zero in steady-state. The error, δy_j , is then found by solving the matrix equation.

$$\begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \frac{\partial f_1}{\partial y_3} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \frac{\partial f_2}{\partial y_3} \\ \frac{\partial f_3}{\partial y_1} & \frac{\partial f_3}{\partial y_2} & \frac{\partial f_3}{\partial y_3} \end{bmatrix} \begin{bmatrix} \delta y_1 \\ \delta y_2 \\ \delta y_3 \end{bmatrix} = \begin{bmatrix} -f_1(y_j^0) \\ -f_2(y_j^0) \\ -f_3(y_j^0) \end{bmatrix} \quad (5)$$

Since the equations of the system are not necessarily linear, the steady-state solution may not be found in one iteration of the matrix equation only. Various techniques have been used to converge on steady state with the least number of iterations.

The user of the TAF code may wish to ignore certain parts of his problem: he may shut off a bypass flow valve or turn off part of an electric network. In such cases, some of the derivatives of the system will be identically zero. This will lead to a row of zeroes in the coefficient matrix. There may also be a free integrator in the system with a zero input, which would lead to a column of zeroes in the coefficient matrix because no derivative is a function of the variable representing the integrator output. In the steady-state solution, the error being solved in the variable whose derivative is zero must also be zero. Thus, the row of zeroes and the corresponding column may be deleted from the matrix, whose order would then be reduced by one. Likewise, a column of zeroes implies a free integrator whose output is arbitrary. The initial guess of the value for that variable is as good as any. The error correction can be considered known to be zero, and all the columns of zeros and their corresponding rows can, therefore, be deleted.

In the matrix solution for transfer functions, this same reasoning can be applied to delete a row of zeroes but can no longer be applied to a column of zeroes, since physically, a free integrator should have infinite gain at zero frequency. After deleting these rows of zeroes, the matrix will no longer be singular.

Refinements

The TAF code has been used successfully in solving various types of systems for as many as 80 integration variables (y_i) and three different inputs (x_k). The code has been refined over the years to include several secondary features such as the handling of implicit equations. These equations correspond to a high-gain

algebraic loop on the analog computer. For example, assume that $g(y_1, y_2, h) = 0$, and that $dy_i/dt = f_i(y_1, y_2, h)$ where g is a function that cannot be solved explicitly for h . The digital-analog simulator codes use different methods for handling such implicit functions. The simplest method is to use values of h from the previous time step. For codes not computing partial derivatives and for time increments much smaller than the time responses one wishes to observe, this method will prove satisfactory. One cannot compute the matrix elements for steady-state or frequency response with such an equation defining the derivatives, since the variables are changed by the factor of 0.003 one at a time (see Equation 4). In a time solution, such treatment only introduces a time lag in the implicit variable. Obviously, it cannot be applied when computing partial derivatives.

Another method of handling implicit functions is by iteration. A Newton-Raphson, *Regula Falsi*, or other method may be used to determine that value of h for which $g(y_j, h) = 0$. Since the value of h changes only a small amount when the y 's change by a small amount following an integration step, a third method of handling implicit functions is possible and has been implemented in TAF. If h is treated as an integration variable, then the derivative dh/dt may be computed automatically by the equation

$$\frac{dh}{dt} = \frac{g\left(y_j + \frac{dy_j}{dt} \Delta t, h\right)}{g(y_j, 1.003 h) - g(y_j, h)} \left(\frac{0.003 h}{\Delta t} \right)$$

If several implicit functions are coupled together, a more complex solution results.

All the integration variables y_j are incremented together by their respective time derivatives multiplied by the time step in the evaluation of $g\left(y_j + \frac{dy_j}{dt} \Delta t, h\right)$.

The above equations are programmed automatically in a closed subroutine called EMDER, which defines the derivative of an implicit variable.

Coding

Figure I shows a basic block diagram of the TAF code. Since the entire code is written in FORTRAN IV, a user must understand the basic FORTRAN language. There are only two subroutines that require coding: INPUT and DER. Their coding has been greatly simplified by using specially designed subroutines that only require input data in addition to COMMON statements. Other parts of the program require only COMMON statements and the allocation of permanent

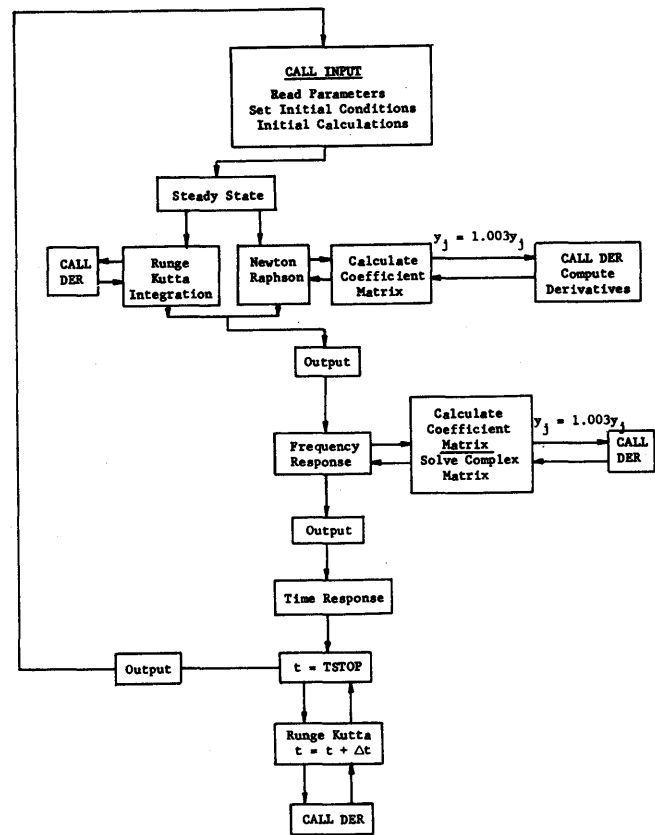


FIGURE 1—Mass on a spring schematic

blocks of storage for all integration variables parameters, integers, etc.

If input parameters are identified by FORTRAN names, they must be added to the DER and INPUT COMMON lists. This permits the use of assigned names in programming DER and in reading the data cards.

The actual coding of the DER and INPUT subroutines can best be explained on a sample problem. However, let us explain first some of the specially designed subroutines that are available.

BCDCON. This subroutine is used for data input and data output. FORTRAN names can be associated with storage locations and used with input and output data, as shown in the Appendix.

The program is used by

```
CALL BCDCON (R1, R2, ---, C1, ---,
             Z(1), Z(2), ---, Z(n))
```

where R1, R2, C1, etc. are FORTRAN names assigned to variables, and Z(1), Z(2), Z(n) is a block of allocated storage. Variables may be a multisubscripted array, and any number may be defined.

Defined data can be read into the program by using only the FORTRAN names followed by assigned data in any format.

Consecutive runs only require a redefinition of the input data to be changed.

To print data with FORTRAN names, one only needs to specify the i -th location of the first and of the last Z_i to be printed.

Many other options to print data (e.g., specified transfer functions, specified integration variables, etc.) have been built into the program. A sample printout is given in the Appendix.

ENTER. This FORTRAN function is a table look-up routine for implementing non-linear algebraic functions. (The tabulated data had been stored when reading-in the input.) Coding requires only the following statement.

$$V = \text{ENTER}(J,U)$$

where J is the index number of the table, and U and V are the independent and the dependent variables, respectively.

TCNSET and TCNTRL. A system being studied for TAF coding frequently contains controllers in the form of the following transfer function.

$$\frac{K_1 \left(1 + \frac{s}{\omega_1} \right) \left(1 + \frac{2\zeta_2 s}{\omega_2} + \frac{s^2}{\omega_2^2} \right)}{s \left(1 + \frac{s}{\omega_3} \right) \left(1 + \frac{2\zeta_4 s}{\omega_4} + \frac{s^2}{\omega_4^2} \right)}$$

Previously, this required the development of sub-routines for transforming the transfer functions into first-order differential equations and then for solving the resultant set of controller equations as a subset to the equations in DER.

In TCNSET and TCNTRL, controllers are defined by input data, and the transformation to first-order differential equations occurs automatically as part of the program. TAF can handle up to 15 different controllers.

The only coding in DER is

$$\text{CALL TCNTRL}(I, U, V)$$

where I refers to the number of the controller being used, and U and V are the input and the output of the controller.

Several other closed subroutines are being used in addition to those previously described. A complete description of all TAF programs can be found in the User's Manual, to be published around December, 1968.

The example presented in the Appendix has been chosen to show the typical coding involved in defining a system that can be represented by differential equations.

ACKNOWLEDGMENTS

Special acknowledgement is given to P. Blake and P. A. Seeker of Westinghouse Astronuclear Laboratory, Pittsburgh, Pennsylvania; and to J. Hafer, Los Alamos Scientific Laboratory, for their contribution of ideas and presentation of the sample problems.

APPENDIX

TAF sample problem

Problem: Mass on a spring, as shown below

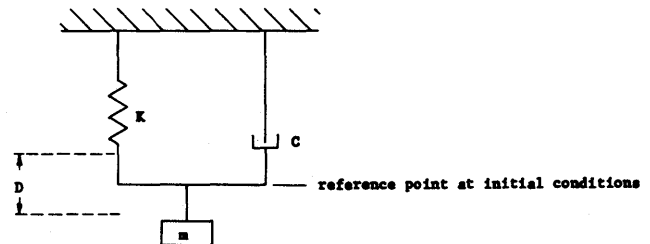


FIGURE 1

The system of Figure 1 can be represented by the equation of motion for a linear damped spring mass system.

$$m \frac{d^2(D)}{dt^2} + C \frac{d(D)}{dt} + KD = f$$

where m = mass, lbm

C = damping constant, lb_f/sec

K = spring constant, lb_f/sec²

f = forcing function, lb_f-in./sec²

D = displacement, in.

The problem is set up for three solutions represented by Equations (1), (2), and (3).

$$\frac{d(D)}{dt} = v$$

(1)

$$\frac{dv}{dt} = -2\zeta\omega v - \omega^2 (D - x)$$

$$\frac{D(s)}{x(s)} = \frac{\omega^2}{s^2 + 2\zeta\omega s + \omega^2} \quad (2)$$

$$\frac{d(D)}{dt} = v \quad (3)$$

$$\frac{dv}{dt} = -2\zeta\omega v - \omega^2(2 - x)$$

where $\omega = \sqrt{\frac{K}{m}}$

$$\zeta = \frac{C}{2\sqrt{K_m}}$$

$$x = \frac{f}{K}$$

These examples have been chosen to demonstrate specific features of TAF. Equation (1) represents the linear, damped, spring-mass system as a coupled set of 2 first order ordinary differential equations. Equation (2) represents the same problem in LaPlace transform notation to exemplify the TCNTRL (I, U, V) feature in TAF. Equation (3) represents a *non-linear* spring-mass system with velocity squared damping to illustrate the ability of TAF to obtain a steady state and a frequency response of non-linear systems.

In each set, the integration variables are displacement (D) and velocity (v), and the system input is displacement (x)

In the TAF program, we can find all three solutions simultaneously since three inputs and 80 integration variables are available. However, before coding, the variables must be identified with FORTRAN names. The FORTRAN names for the derivative and integration variables are listed below for all three solutions.

FORTRAN NAMES

Variable Name	Solution 1		Solution 2		Solution 3		
D	D	DISPL	Y(1)	TCNDIS	Y(4)	DNL	Y(5)
v	V	VEL	Y(2)			VNL	Y(6)
$\frac{d(D)}{dt}$	DDT		YP(1)			DNLDLT	YP(5)
$\frac{d(v)}{dt}$	VDT		YP(2)			VNLDT	YP(6)
x			X(1)		X(2)		X(3)

The equivalence and BCDCON statements in INPUT allow the use of different names and, at the same time, allow the main code to use specified values for Y_i, YP_i, Z_j . For instance D, Y(1) and DISPL are all the same variable. Y(1) is used by the main program. D may be used in coding equations and DISPL will be printed with the correct data value. The INPUT subroutine listing shows the use of different names for the same variable. Any of the three names may be used for input data or in coding, and the first name given in the BCDCON list will be printed to identify the output data.

In addition to those listed above, other variables with their assigned FORTRAN names and storage assignments are:

Variable	FORTRAN NAMES		
ω	WO	NTFRQ	Z(1)
ζ	ZETA	ZETA	Z(2) Storage
m	XM	MASS	Z(3)
K	XK	SPGCI	Z(4) Assignment
C	XC	DAMPF	Z(5)

INPUT Subroutine. The FORTRAN listing includes all the coding needed in this subroutine for solving all three sets of equations simultaneously. The input data are read by SYMBOL, and ω and ζ are computed from m, K, and C.

```

EQUIVALENCE (Y(1),D), (Y(2),V), (YP(1),DDT), (YP(2),VDT), (Z(1),WO)
1, (Z(2),ZETA), (Z(3),XM), (Z(4),XK), (Z(5),XC)
EQUIVALENCE (Y(5),DNL), (Y(6),VNL), (YP(5),DNLDT), (YP(6),VNLDT)
CALL BCDCON(48HDISPL,VEL,DDT,ACCEL $
1 ,D,V,DDT,VDT)
CALL BCDCON(36HNTFRQ,ZETA $
1 ,WC,Z(2))
CALL BCDCON(36HMASS,SPGCT,DAMPF $
1 ,XM,XK,XC)
CALL BCDCON(24HTCNINP,TCNDIS $ ,Y(3),Y(4))
CALL BCDCON(51HDNL,VNL,DNLDT,VNLDT $
1 ,DNL,VNL,DNLDT,VNLDT)

CALL SYMBOL(I,5)
IF(IRUN.GE.1)RETURN
WO = SQRT(XK/XM)
ZETA = XC/(2.*XM*WO)
CR(1,1)=WO
CR(2,1)=ZETA

```

The BCDCON statements and EQUIVALENCE statements could be left out if all coding included only $Y(I)$, $YP(I)$, $Z(J)$, and $X(K)$ as variables.

DER Subroutine The FORTRAN listing includes all the coding needed in this subroutine for solving all three sets of equations, simultaneously. The first four equations are coded from (1) and (3). The solution of (1)

shows an example of solving first-order differential equations, and the solution of (3) shows an example using second-order non-linear equations. The solution of Eq. (2), which should be identical to the solution of Eq. (1), is given to show how controller problems may be solved by using TCNTRL. The input data name, INFO, defines the number of terms in the numerator and denominator and DR and CR define their value.

```

EQUIVALENCE (Y(1),D), (Y(2),V), (YP(1),DDT), (YP(2),VDT), (Z(1),WO)
1, (Z(2),ZETA), (Z(3),XM), (Z(4),XK), (Z(5),XC)
EQUIVALENCE (Y(5),DNL), (Y(6),VNL), (YP(5),DNLDT), (YP(6),VNLDT)
C - - - - -
VNLDT=-2.*WO*ZETA*VNL*ABS(VNL)-WO**2*(DNL-X(3))
DNLDT = VNL
VDT = -2.*ZETA*WC*V -WO**2*(D-X(1))
DDT = V
CALL TCNTRL (1,X(2),OUT)

```

Input Data. The input data consist of indices to define options and the number of variables $Y(I)$, $Z(J)$, and $X(K)$; of controller data; input variables; initial conditions of integration variables; frequencies (ω) at

which frequency response data are to be calculated; and the time response variables at starting (TS), stopping (TSTOP), and interval (DT) times. Data cards are read until 1 is encountered in Column 1 of a data card.

```

NV 6  NZ 5  NI 3  NC 1  IY1(1) 3  NYT(1) 1  2  3  4  NSKIP 200
NPASS 25  ISS 2  IFREQ 1  ITIME 0  ICSS 1  ICD 1  IYP 1  NFREQ 19
NYTPR 4  NYFPR 6  NFPL0T 6  NYF 1  2  3  4  5  6  CVERR .5
INFO(1) 00001  DR(1,1) 1.  CR(1,1) 1.414214  CR(2,1) .035360678
X 16.1  16.1  16.1  Y(4) 16.1
MASS 1.  SPGCT 2.  DAMPF .2828428  VEL 0.  DISP 15.
W .01 .04 .063 .1 .15 .25 .4 .63 1. 1.414214 1.5 2.5 4. 6.3 10.
15. 25. 40. 63.

```

Output Data The first set of printed data lists the indices, defines the problem conditions, and the input

Y(I), Z(J), and X(K). A typical printout of input data follows.

```

1  SAMPLE PROBLEM  MASS ON A SPRING
TAF RUN NO.  1          CASE NO.  0          TIME  1.238      0/ 0/ 0
ISS  IFREQ  ITIME  ISPNCH  IFPNCH  ITPNCH  ITC  ICSS  IYP  IOFF  IBKFO
  2    1    0    0    0    0    0    1    1    0    0
NV   NZ   NI   NC  NYFPR  NZFPR  NYTPR  NZTPR  NDATA  NTPLOT  NPASS
  6    5    3    1    6    0    4    0    0    0    25
IY1 =  3    0    0    0    0    0    0    0    0    0    0
INFO =  1    0    0    0    0    0    0    0    0    0    0
NYF =  1  2  3  4  5  6  0  0  0  0  0  0  0  0  0  0  0
NZF =  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
NYT =  1  2  3  4  0  0  0  0  0  0  0  0  0  0  0  0  0
    
```

A typical printout of variables after the steady-state solution is calculated follows.

```

STEADY STATE VARIABLES AFTER 1 PASSES
THESE ARE DIFFERENTIAL EQUATION Y#S
1 DISPL  .161E+02  2 VEL  0.  3 TCNINP  .161E+02
6 VNL  0.  7 0.  8 0.
THESE ARE X#S. INPUT VARIABLES
X1 .161E+02  X2 .161E+02  X3 .161E+02
THESE ARE ALGEBRAIC EQUATION Z#S
1 NTRQ .141E+01  2 ZETA .100E+00  3 MASS .100E+01
    
```

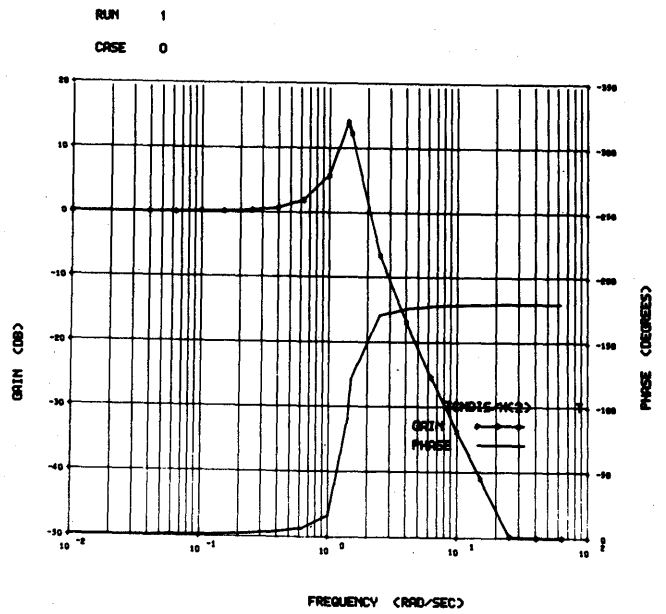
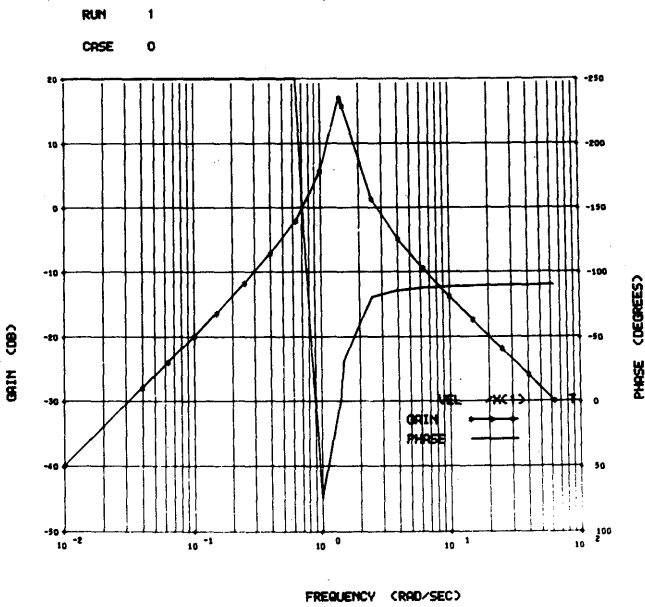
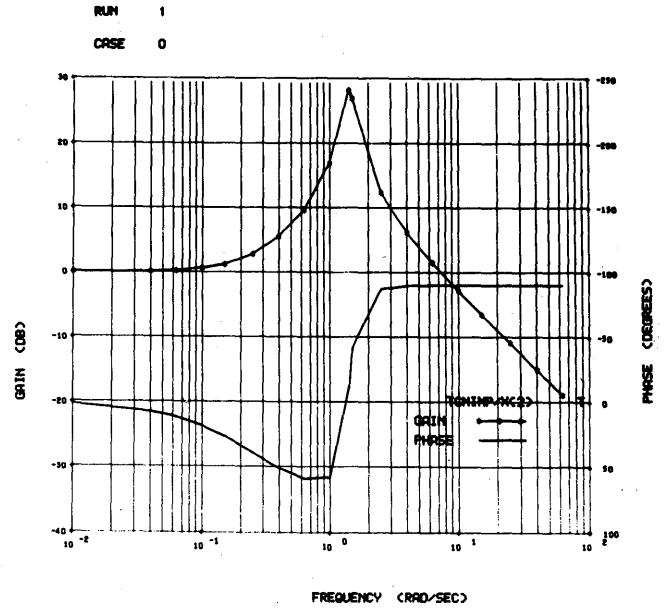
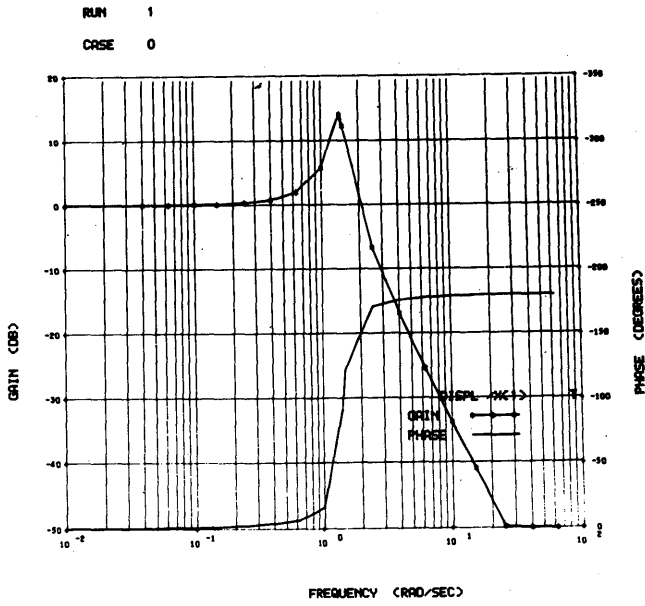

Typical printouts of the transfer function solution and the time response solution are shown below.

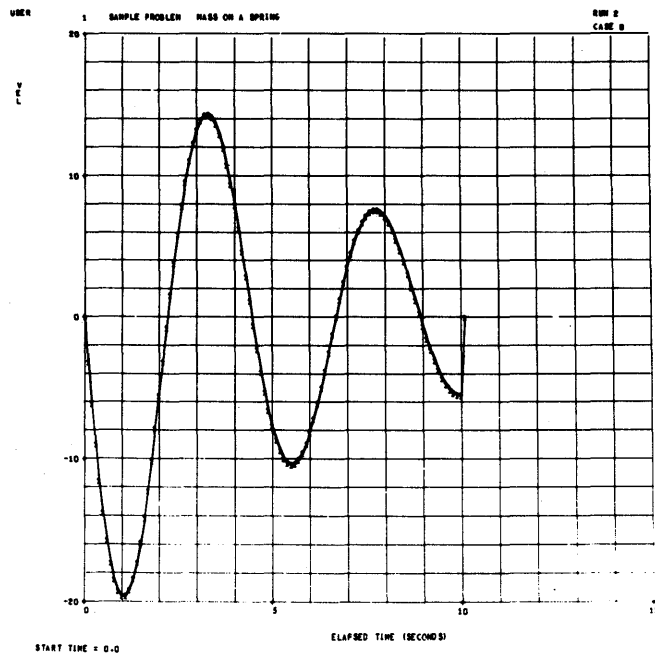
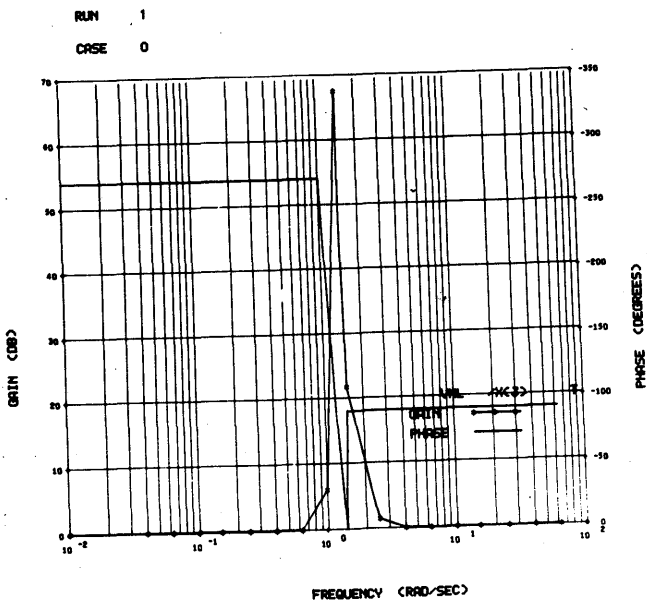
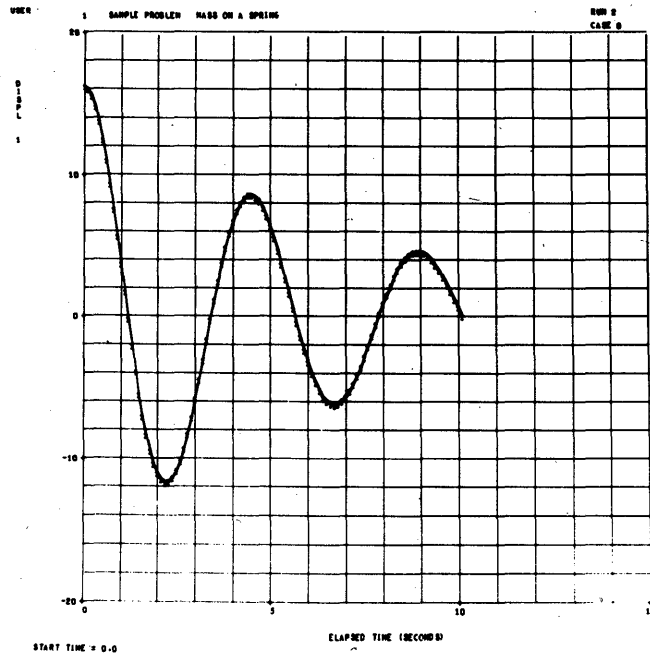
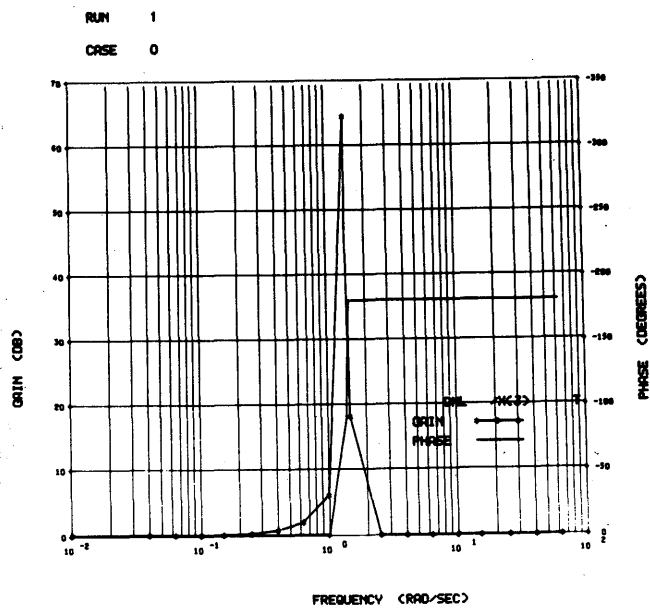
TRANSFER FUNCTIONS DISPL /X(1)			
(RAD./SEC.)	MAGNITUDE	DECIBELS	PHASE
.100E-01	.100E+01	.000	-.081
.400E-01	.100E+01	.007	-.324
.630E-01	.100E+01	.017	-.511
.100E+00	.100E+01	.043	-.814
.150E+00	.101E+01	.096	-1.229
.250E+00	.103E+01	.270	-2.090
.400E+00	.108E+01	.708	-3.519
.630E+00	.124E+01	1.868	-6.343
.100E+01	.192E+01	5.686	-15.793
.141E+01	.500E+01	13.979	-90.000
.150E+01	.406E+01	12.173	-120.509
.250E+01	.464E+00	-6.666	-170.554
.400E+01	.142E+00	-16.930	-175.380
.630E+01	.530E-01	-25.513	-177.293
.100E+02	.204E-01	-33.807	-178.347
.150E+02	.897E-02	-40.947	-178.910
.250E+02	.321E-02	-49.869	-179.350
.400E+02	.125E-02	-58.050	-179.594
.630E+02	.504E-03	-65.948	-179.743

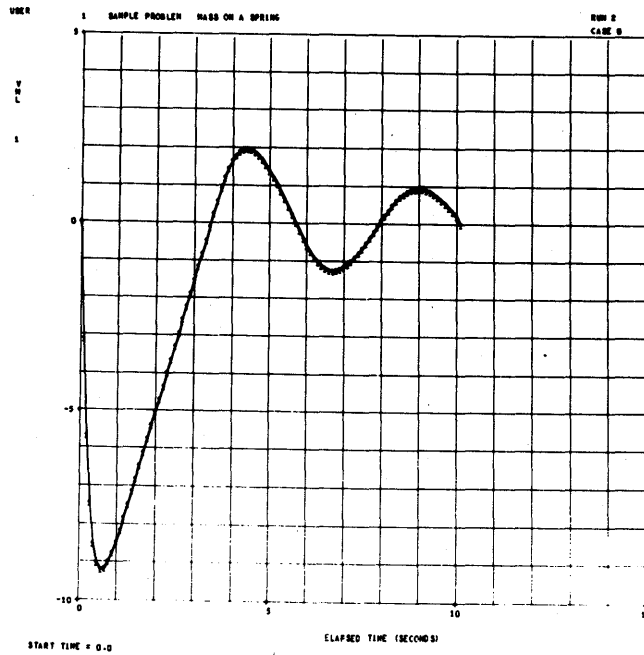
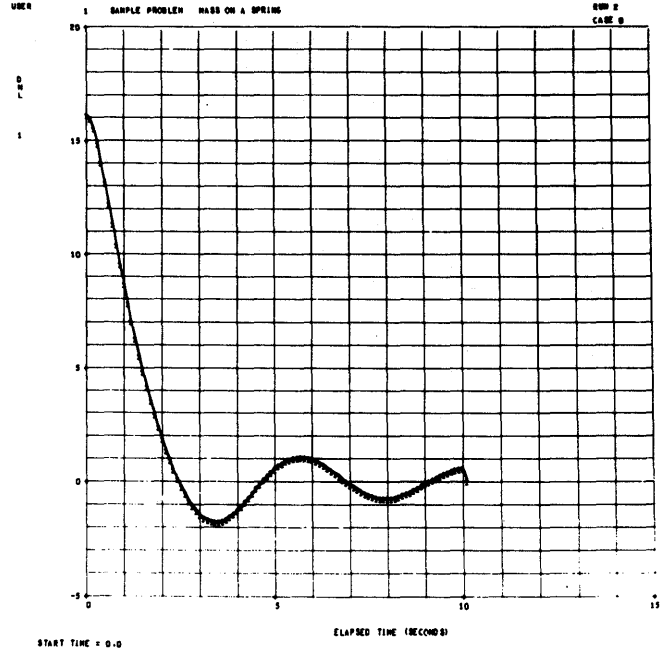
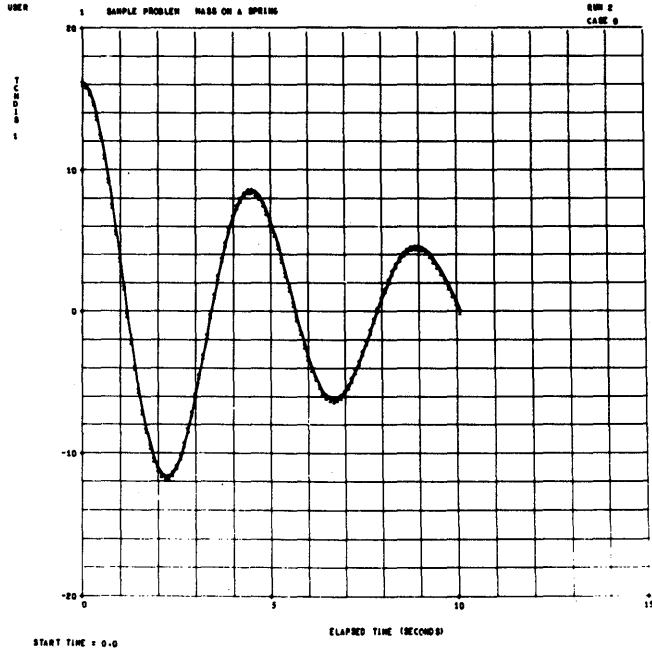
TIME	DISPL	VEL	TCONDIS	DNL	VNL
0.	.1610E+02	0.	.1610E+02	.1610E+02	0.
.1000E+00	.1594E+02	-.3164E+01	.1594E+02	.1594E+02	-.3115E+01
.2000E+00	.1547E+02	-.6178E+01	.1547E+02	.1550E+02	-.5685E+01
.3000E+00	.1471E+02	-.8986E+01	.1471E+02	.1483E+02	-.7465E+01
.4000E+00	.1368E+02	-.1154E+02	.1368E+02	.1403E+02	-.8519E+01
.5000E+00	.1241E+02	-.1379E+02	.1241E+02	.1315E+02	-.9039E+01
.6000E+00	.1094E+02	-.1571E+02	.1094E+02	.1223E+02	-.9210E+01
.7000E+00	.9284E+01	-.1727E+02	.9284E+01	.1131E+02	-.9169E+01
.8000E+00	.7495E+01	-.1844E+02	.7495E+01	.1040E+02	-.9001E+01
.9000E+00	.5608E+01	-.1922E+02	.5608E+01	.9513E+01	-.8759E+01
.1000E+01	.3663E+01	-.1960E+02	.3663E+01	.8651E+01	-.8474E+01

A complete set of Bode plots to show the frequency response of all three solutions is included in the following computer plots. The time response of the integration

variables has also been included to show the typical plots available to users of the program.







The automated medical history

by WILLIAM WEKSEL

Cytek Information Systems Corporation
New York, New York

and

PAUL N. SHOLTZ

IBM Corporation
Rochester, Minnesota

and

JOHN G. MAYNE

Mayo Clinic
Rochester, Minnesota

INTRODUCTION

It has often been suggested that computer technology could help solve problems in medicine. The Automated Medical History (AMH) system is designed to help the physician collect data from the patient. The system's objective is to lessen physician involvement in routine activities, thereby increasing his availability to provide patient care. The AMH should help alleviate the chronic shortage of medical personnel even as it extends the physician's capabilities to collect patient information.

Collecting data from patients involves many related activities that can be roughly categorized as history taking, physical examination, and laboratory tests. It is difficult to determine precisely the relative importance of these activities. However, physicians generally believe that the patient's medical history has basic importance in interpreting medical findings. For this reason, the medical history is a logical starting point for developing techniques to aid the physician in his data collection activities.

We propose that information technology be used as an adjunct to the traditional physician-patient relationship, specifically for collecting medical his-

tory data. The feasibility of using computer technology has been demonstrated and discussed in another paper.¹ This paper describes the AMH system and briefly reviews the system's data collection capabilities. It also discusses the possibility of using the AMH to evaluate the medical data collected.

The Automated Medical History (AMH)

The AMH is a medical questionnaire that is presented to a patient on a display terminal. An IBM 1050 Data Communication System is used to enter patient identification information and to print the summary of each patient's responses. This standardized, legible, condensed summary is sent to the physician before the physician-patient interview.

The display terminal projects photographic images that are stored on 16 mm color film (Figure 1). This terminal is a prototype of the recently announced IBM 2760 Optical Image Unit. Images can be selected in any order by the computer. Patient responses are entered on the display terminal with an electronic light pen. An array of response areas is arranged in a 10 by 12 matrix, making possible 120 unique responses on each frame.

Colors emphasize the three functional areas of



Figure 1—Experimental graphic display terminal

each frame, which are the question, the response alternatives, and the instructions (Figure 2).

Color photographs and pictorial techniques are also used to supplement written explanations of medical terminology. For example, if a patient does not understand a question about “skin cancer” or some other skin condition, a picture of the condition can be shown (Figure 3). Pictorial techniques also allow the patient more freedom of expression. For example, this illustration permits him to specify the location of abdominal pain (Figure 4).

Computer-administered questioning permits the effective use of question branching. This means a questionnaire can be tailored to the individual patient, so that factors such as sex, age, education, and ability to understand questions can be accounted for. Moreover, the patient answers only those questions which pertain to his own medical problems.

Question branching is based on responses given by the patient. Specific uses of branching in the AMH will be considered later in this paper.

Computer administration permits control over

Are you taking any medicines now or have you taken any within the past month?

Yes

No

→ Go back

Figure 2—Frame



This is an example of a skin cancer. Do you understand?

Yes No

→ Go back

Figure 3—Skin condition

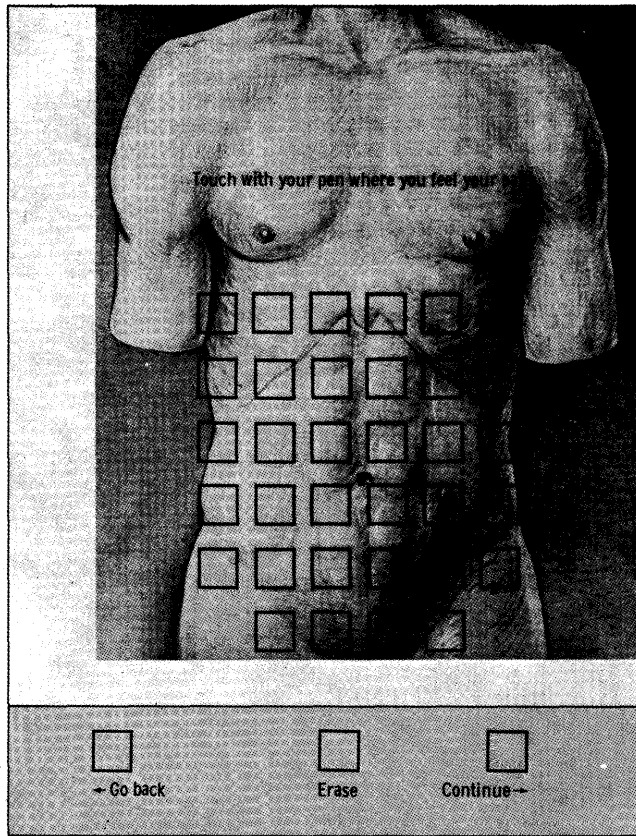


Figure 4—Body

the patient while he is answering the questions. That is, the patient must answer *all* questions because the system can be designed so that the questioning process will not proceed until an answer is given. The computer can also monitor patient performance by recording such things as undue delays and excessive erasures.

AMH system description

The graphic display terminal and the IBM 1050 Data Communication System are controlled by an IBM 7040 Data Processing System. To permit efficient use of the computer, the operating system was modified to permit concurrent operation of terminal service programs and background programs. Experience indicates only 1% to 2% of available computer time is required for terminal activities.

A special purpose language was developed to permit computer control of the graphic display terminal activity at each step of the questionnaire. This language was specifically designed to simplify the display of particular film frames, the manipu-

lation of light pen coordinates, and the control of branching operations. Also included were extensive text composition facilities, for use in construction of the summary statement.

The experimental graphic display terminal seems ideally suited to this particular application. It has been operated successfully by completely untrained users (patients) who had very brief instructions. There are two reasons for the suitability of this terminal: first, very high quality colored images are obtained by photographic projection, and second, a very simple and direct method of response is made possible by the use of the light pen.

The questionnaire

The questionnaire consists of broad, inclusive, screening questions organized according to conventional body systems. These questions attempt to detect the presence of actual or potential disease states. The questions are all of the fixed choice type. Response alternatives vary from a simple "yes," "no" to multiple choice check lists describ-

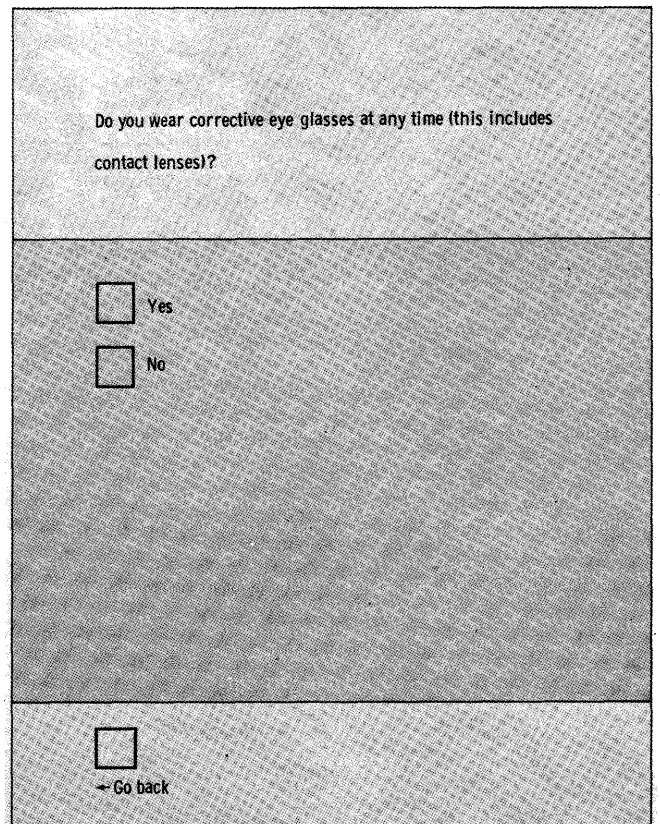


Figure 5—Question types

ing symptoms and conditions. The format usually is dictated by the objectives of a specific question (for example, see Figures 5, 6, 7). The number of questions asked varies from patient to patient because of question branching. It ranges from 226 to 302 questions for females, and from 212 to 283 for males.

Currently, there are four uses for branching in the AMH:

- 1) To ask the patient questions that pertain only to his specific problems.
- 2) To explain questions and medical terminology that the patient does not understand. For example, if a patient does not understand a question about "yellow jaundice" (Figure 8), he is shown this explanation (Figure 9). If he indicates that he understands, he is shown the original question again. If not, he goes on to further questions. The fact that he did not understand the "yellow jaundice" question is recorded in the summary.
- 3) To give the patient greater latitude when

Which of the following phrases best describe the speed of your heartbeat?

I am not usually aware of the speed of my heartbeat.

My heartbeat is sometimes very fast.

My heart seems to beat very fast all of the time.

My heartbeat is sometimes very slow.

I occasionally have attacks of very rapid heartbeat, which usually start suddenly and stop suddenly.

None of the above describe it.

← Go back Erase Continue →

Figure 6—Question types

Which of the following bring you to see the doctor now?

<input type="checkbox"/> Muscles or joints	<input type="checkbox"/> Overweight
<input type="checkbox"/> Back (spine) and neck	<input type="checkbox"/> Fever
<input type="checkbox"/> Skin trouble	<input type="checkbox"/> Headaches
<input type="checkbox"/> Brain	<input type="checkbox"/> Allergy
<input type="checkbox"/> Emotions (nervousness)	<input type="checkbox"/> Swollen glands
<input type="checkbox"/> Kidneys and urine bladder	<input type="checkbox"/> Hernia
<input type="checkbox"/> Glands (thyroid or others)	<input type="checkbox"/> None of the above bring me to see the doctor now.
<input type="checkbox"/> Sex organs	

← Go back Erase Continue →

Figure 7—Question types

describing his medical problem. For example, if a patient indicates he has abdominal discomfort (Figure 10), question branching provides various descriptions of such discomfort (Figure 11). Although many of these terms are redundant and have no differential diagnostic value, the patient can give his answer in words familiar and meaningful to him.

- 4) To evaluate the significance of a positive response, using branching questions to distinguish between common occurrences and significant medical symptoms. For example, if a patient complains of a headache, he is questioned on aspects of the headache such as severity, frequency, duration, and his concern about it. These questions permit assessment of whether or not the headache is a significant symptom that requires medical follow-up or whether it is an unimportant occurrence that requires no further investigation. This is an important use of question branching that

Have you ever had yellow jaundice?

Yes

No

Don't understand

→ Go back

Figure 8—Yellow jaundice question

Yellow jaundice is a condition in which the skin and the whites of the eyes become distinctly yellow. When jaundice is present the bowel movement may become a pale putty color and the urine may become dark in color.

Do you understand?

Yes

No

I understand the explanation but I still don't understand the question.

→ Go back

Figure 9—Yellow jaundice explanation

takes the AMH beyond the task of data collection to that of data evaluation.

A condensed summary of the patient's responses is provided for the physician. Negative responses (unless pertinent to other positive responses) are not provided in this summary. This summary is a prototype of a standardized, legible medical record (Figure 12).

Evaluation of the AMH

One hundred fifty-nine Mayo Clinic patients were randomly selected for AMH administration. The highlights of this experiment were:

- 1) Patient reaction was *extremely* favorable. Of the 159 patients sampled, 154 participated; only two reacted unfavorably.
- 2) All patients were able to answer the questionnaire and operate the terminal. Mean patient age was 50.1 years (95% confidence limits were $48.5 < \mu < 51.7$) and mean education level was 11.8 years (95%: $11.4 < \mu < 12.2$). This suggests that a high educa-

tion level is not necessary for successful participation.

- 3) Physician reaction was favorable. The tone and extent of the criticisms received indicate physician acceptance of the AMH in its role as collector of medical histories.
- 4) Mean questionnaire completion time for patients (95%: $62.2 < \mu < 69.2$) was 65.7 minutes.
- 5) The AMH obtained 95% of the symptom information recorded by the physician in the traditional patient record. This study compared patient symptoms obtained by the AMH to those recorded by the examining physician in the patient's Mayo Clinic Medical Record.
- 6) Comparisons between the traditional patient record and the AMH summary with respect to "past surgery" and "past illness" information suggest that, given the total set of patient responses, the AMH data collection performance was significantly better than that of the physician (Table I).

Table I

Two-Way Comparison: AMH Data vs Physician-Recorded Data		
Item	Proportion of physician-recorded data also obtained by AMH (%)	Proportion of AMH-obtained data also physician-recorded (%)
Past surgery*	86	62
Past illness*	59	39
Past family illness and cause of death	57	62

*Difference statistically significant. $P > P_{.05}$

These results indicate that reliable medical history gathering techniques can be developed to aid the physician.

Use of the AMH for the evaluation of medical symptoms

We have already indicated that branching can be used to evaluate the significance of positive

Do you **NOW** (that is, within the past 3 months) have any type of abdominal discomfort?

Yes

No

→ Go back

Figure 10—Abdominal pain question

Which of the following best describes that abdominal discomfort?

<input type="checkbox"/> Sour stomach	<input type="checkbox"/> Belly pain
<input type="checkbox"/> Bloating feeling	<input type="checkbox"/> Sore belly
<input type="checkbox"/> Cramps	<input type="checkbox"/> Acid indigestion
<input type="checkbox"/> Nausea	<input type="checkbox"/> Knife-like pain
<input type="checkbox"/> Boring pain	<input type="checkbox"/> Twisted knotted sensation
<input type="checkbox"/> Pain in my side	<input type="checkbox"/> Other (not listed)
<input type="checkbox"/> Heart burn	

→ Go back Erase Continue →

Figure 11—Abdominal pain descriptions

symptoms. Such an evaluation has two aspects. Given a positive response to a question, should it be followed up? *If* it is followed up, what is the best way—more questions, laboratory tests, or physical exam? We will consider the first aspect of the problem.

We attempt to obtain additional information (on a positive symptom) that will help us decide whether or not the symptom is significant enough at the system review level to require more questioning. For example, if a patient indicates that he has headaches, we need information regarding headache severity, frequency, and duration, as well as the degree of the patient's concern about his headaches. Some combination of these items should form the basis for deciding whether or not the patient's positive response to the headache question requires follow-up.

We need to find which patient responses, singly or in combination, are the most important determinants of whether or not a physician follows up on his patient's complaint. This knowledge would allow us to assign relative weights to various pa-

AUTOMATED MEDICAL HISTORY

Summary Statement 05/31/67

S.S.#: 000-00-0001 Male White Born: 9/22/19
 Educ: high school Married Brown Hair Birthplace: Canada
 Brown Eyes Right handed

Wants general checkup. Referred by MD. Health concerns: stomach and swallowing. Self evaluation: not in good health. Previous illnesses: ulcer, mumps.

SYSTEMIC REVIEW

Ophthalm: Defective vision correctable with glasses. Wears glasses.
ENT: Freq. dysphagia with solids. Occ. dysphagia with liquids.
Cardiovascular: Notes cold-induced acral blanching.
GI: Appetite decrease past 3 mos. Dyspepsia with many foods, greasy or rich foods, spicy foods. Now troubled by abd. discomfort described as sour stomach, bloated feeling, cramps, nausea, boring pain, heart burn, belly pain, sore belly, acid indigestion, knife-like pain, twisted knotted sensation. Location: umbilical. Thinks he has ulcer. Complains of belching. Bowel movement less frequent past 6 mos. In past yr. had occas. constipation. Has noted blood in bowel movements. In past year stool occ. characterized by excessively putrid odor, smaller diameter. Within past 6 mos. bothered by itching and/or burning rectum. More than 3 mos. ago had abd. discomfort described as sour stomach, bloated feeling, cramps, nausea, boring pain, sore belly, acid indigestion. Location: umbilical. Past MD Dx: duodenal ulcer. Has had x-ray of stomach, gallbladder, complete GI tract.
Neuro: Troubled with headache past 6 mos.; onset over 5 yrs. ago. Not usually relieved by ASA or equiv. Has constant pain. Had same type headache in past.
Psych: Occas. trouble sleeping in past few wks.
Skeletal-Muscle: Troubled with upper back pain past 6 mos. Concerned re backache. Backache over 6 mos. ago.
Family: Blood relatives have died of high blood pressure. Has living blood relatives with high blood pressure.
Occup: tradesman.
Habits: Smokes cigarettes (less than a pack/day). Has smoked more than 20 years, inhales, increased cigarette smoking in past year. In past year was on ulcer diet(s). Diet(s) followed regularly; advised by MD. In the past month has taken unidentified medicine.

Pt. didn't understand questions about nasal polyps.

End of session.

Figure 12—Summary statement

tient responses when evaluating positive symptoms. This knowledge would also give us a statistical basis for determining the discriminative power of each individual question. Using this statistical basis, items with low discriminative power could be discarded.

In our investigation, patients with positive responses to questions regarding the incidence of "headache" were grouped. Then, on the basis of the information extracted from the patient record, this group of patients was subdivided into those who had a follow-up exam on headaches (hereafter referred to as the follow-up group) and those who did not have a follow-up exam on headaches (hereafter referred to as the non-follow-up group). "Follow-ups," then, had further investigation by the physician regarding their headache problem. In some cases, this was in the form of a head x-ray; in other instances it was a consultation with a neurologist regarding the headache problem, or a final diagnosis was made which suggested that the headache problem had been followed up.

Two statistical techniques were used to investigate the relative importance of the items used to evaluate the routine headache response. The first

of these was the "likelihood ratio" which is represented by the relative frequency with which a symptom or set of symptoms occurred in the follow-up groups (P_{fj}^s) as compared to the relative frequency with which the same symptom or combination of symptoms occurred in the non-follow-up groups (P_{nfj}^s). That is, the ratio was

$$\theta = \frac{P_{fj}^s}{P_{nfj}^s}$$

The assumptions underlying the use of θ and its use for item analysis has been dealt with by Neyman² and Collen³ and will not be discussed here. These authors indicate that the resulting θ can be used as the basis for establishing sets of values for which a diagnosis will be positive and sets of values for which a diagnosis will be negative. We have sought to examine only the discriminative value of the items in the question branches.

Table II repeats these values for the branch of questions used to evaluate a positive headache response. In this table, we show the relative frequencies of occurrence of different categories for headache duration, frequency, severity, incidence of relief with simple medications, and the patient's concern about his headache. Clearly, duration—

Table II

Discriminative Value of Features Which Describe Headaches (n = 77)				
Feature	P_f^s	P_{nf}^s	$\alpha = 0.05$ Statistical Significance	$\theta = \frac{P_f^s}{P_{nf}^s}$
Duration			$\chi^2 < \text{chi sq. n. s.}$	
< 1 year	0.33	0.41		0.81
Between 1-5 yr	0.30	0.20		1.5
More than 5 yr	0.36	0.39		0.92
Frequency			$\chi^2 > \text{chi sq.}$	
Increasing	0.52	0.25		2.1
Same	0.39	0.34		1.1
Decreasing	0.09	0.41		0.2
Perceived Severity			$\chi^2 > \text{chi sq.}$	
Increasing	0.36	0.20		1.8
Same	0.61	0.43		1.4
Decreasing	0.03	0.36		0.08
Relief with simple medicine			$\chi^2 > \text{chi sq.}$	
Yes	0.39	0.71		0.55
No	0.61	0.29		2.1
Patient concern			$\chi^2 > \text{chi sq.}$	
Yes	0.70	0.25		2.8
No	0.30	0.75		0.4

that is, whether a headache has lasted less than a year, between 1 and 5 years, or more than 5 years—seems to have no bearing on whether or not the physician follows up on a patient. There is no significant difference between the groups in this case. Whether or not the groups are significantly different is the basis for deciding whether or not the θ 's are of any value.

The other categories in the headache branch are more important, as significant differences were found between the patient groups. Notice that the ratio of the θ 's to one another is high from the positive to the negative response. Someone who has any one (or all) of the following: headaches that are increasing in frequency, headaches that are increasing in severity, or concern for his headaches without being able to obtain relief with simple medications, is likely to receive a follow-up examination.

How do these features relate to follow-up when they are combined? And, in combination, which of these items is the most important? To determine this, a multiple correlation analysis was done in which the criterion variable was membership in the follow-up group and the predictors were the items listed in Table II. The results of this analysis are listed in Table III. The multiple correlation is .54, which is statistically significant. The third column of this table, which lists the regression coefficients, shows the relative importance of each of the independent variables in influencing the criterion or dependent variable. We can see that the most important variable is the patient's concern, which has the largest regression coefficient. Note that this variable had the largest ratio between θ 's, as seen in Table II.

Table III

Prediction of Membership in Headache "Follow-Up" Group (criterion variable) Using Features Which Describe Headaches as Predictors.

Predictor Variables	Zero Order Correlation (with criterion)	Regression Coefficients
Duration	0.03	0.06
Frequency	-0.36	-0.07
Perceived severity	-0.34	-0.06
Relief w/simple medicine	0.31	0.18
Patient concern	-0.44	-0.30

Multiple R = 0.54
F value = 4.77, $F > F_{.05}$

Using this information, it is possible to assign relative weights to responses of future patients. The combination of these relative weights would be the decision procedure used to determine whether or not physician follow-up is required.

A similar analysis was performed with respect to "neck pain." In this analysis, only two items were in the branch: association of the pain with nervousness, and past occurrences of neck pain. The theory was that neck pain consistently associated with nervousness was not significant. Similarly, if the patient had previous neck pain, it was not as important as if the neck pain was a new phenomenon. Table IV bears this out. Patients with neck pain not consistently associated with nervousness were found in the follow-up group more often than in the non-follow-up group. The item regarding past occurrences was less important.

Table IV

Discriminative Value of Features Which Describe Neck Pain (n = 56)

Feature	P_f^s	P_{nf}^s	$\alpha = 0.05$ Statistical Significance	$\theta = \frac{P_f^s}{P_{nf}^s}$
<u>Association with nervousness</u>				
Occurs when nervous and at other times	0.70	0.22	$\chi^2 > \text{chi sq.}$	3.1
Can't really say when it occurs	0.25	0.44		0.5
Occurs mainly when nervous or upset	0.05	0.33		0.1
<u>Past occurrences</u>				
Trouble with neck pain in past	0.45	0.28	$\chi^2 > \text{chi sq.}$	1.6
No trouble with neck pain in past	0.55	0.72		0.7

These findings are consistent with the multiple correlation analysis presented in Table V. The association of neck pain with nervousness seems to be the most important variable.

It should be noted in both the headache and neck pain cases that although the multiple correlations are significant, they are low enough in magnitude to indicate that there are many other reasons which relate to being assigned to the follow-up group. Some of these reasons may be neurological complaints given in response to other questions.

Table V

Prediction of Membership in Neck Pain "Follow-Up Group" (criterion variable) Using Features Which Describe Neck Pain as Predictors.

Predictor Variables	Zero Correlation (with criterion)	Regression Coefficients
Association w/nervousness	-0.47	-0.28
Past occurrences	-0.17	-0.15

Multiple R = .50

F value = 8.6, $F > F_{.05}$

To establish these relationships with other questions is a much larger problem than the one we have considered in this paper. At present, however, it seems apparent that the techniques we have discussed will enable us to find criteria for deciding whether or not particular positive instances require further investigation or whether they can be considered unimportant occurrences.

SUMMARY

The Automated Medical History is a system to collect medical history information from patients

by using a computer-controlled display terminal. Computer administration permits effective use of branching techniques, thus making possible an individualized questionnaire for each patient. Patient responses are condensed into a concise summary statement for the examining physician.

The system was tested with 159 patients. Patients and physicians reacted favorably to the system. The AMH obtained approximately 95% of the information recorded by the physician in the patient's medical record.

The possibility of using the AMH to evaluate the significance of the data collected was also explored.

REFERENCES

- 1 J G MAYNE W WEKSEL P N SHOLTZ
*Toward automating the medical history**
Mayo Clinic Proceedings 43:1-25 January 1968
- 2 J NEYMAN
First course in probability and statistics
Henry Holt New York 1950 Chapter V
- 3 M F COLLEN
Machine diagnosis from a multiphasic screening program
Proceedings of the Fifth IBM Medical Symposium October 7-11 1963 Endicott NY

*Additional references (12) are cited in this paper.

The key to a nationwide capability for computer analysis of medical signals:

The dedicated medical signal processor*

by CESAR A. CACERES, GABRIEL KUSHNER,**
DAVID E. WINER and ANNA LEA WEIHRER

*U.S. Department of Health, Education and Welfare
Washington, D.C.*

INTRODUCTION

The Medical Systems Development Laboratory has demonstrated a system of computer analysis that encourages wider use of medical signals, reduces unit costs, and alleviates the shortage of physician manpower while concurrently improving the quality of interpretations. It is adaptable to hospital wards, outpatient departments, routine physical examinations, and health screening programs. Its development anticipates the worsening shortage of professionals able to analyze medical data.

A large number of those concerned with the practice of medicine recognize that automated analysis is the only feasible answer. At the same time, most medical-service groups are reluctant to install systems in their own institutions because existing computer hardware has not been adapted to meet their needs. Existing systems are expensive, bulky, and because of the flexibility for which they were designed require scarce computer technicians to operate and maintain. The need of most potential medical users is for quite the opposite: a small computer system that requires minimal maintenance and that can be operated by the personnel usually available to medical groups.

The aim of this paper, after a brief review of the background, is to describe a model for a com-

puter system designed for the needs of medical-service groups. The model is based on an existing "breadboard" system.

Background of need

The electrocardiogram was chosen as the model signal for automated pattern recognition and interpretation by computer because physicians are familiar with it and have 50 years of experience in relating its waveforms to specific meanings. But almost any other physiological signal—brain waves, vital capacity, heart sounds, and others—could have been selected. In fact, we have since demonstrated that each of these signals is subject to automated measurement and interpretation. The computer measurements are accurate, and clinicians can accept the interpretations as readily as they accept those of other clinicians.

Approximately 50 million electrocardiograms are interpreted annually in the United States by medical personnel. It is probable that changes in population age-groups and the impact of health legislation will double the volume of electrocardiograms long before it is possible to train the required specialists. Conventional techniques of electrocardiography would make the cost of the projected increase prohibitive for disease control and prevention purposes in terms of physician-technician man-years.

From a manpower viewpoint the most important factor is total reading time, including measurement and interpretation time by interns, residents, technicians, and finally that of the physician who signs the electrocardiogram report. Although some tracings can be read by cardi-

*From the Medical Systems Development Laboratory (MSDL) Heart Services Research and Development, National Center for U.S. Department of Health, Education, and Welfare, Washington, D.C.

**Also Associate Professor of Medicine, George Washington University, Washington, D.C.

ographers in a few minutes, *total* reading time—for problem as well as routine cardiograms—averages close to 15 minutes. Thus the reading of 50 million electrocardiograms, in terms of a 2,000-hour man-year, takes up an estimated 6,250 ECG reader man-years. In conventional systems, reading time accounts for 75% of the cost. Use of a computer system can reduce reading time 5- to 7 fold and facilitates the reduction of secretarial (bookkeeping, filing) and technician (recording) time.

The current computer system prints out all the significant measurements with a verbal diagnostic statement based on the criteria of a consensus of cardiologists. Electrocardiograms and spiograms are now being taken from patients in hospitals and clinics evaluating benefits of immediate on-line analysis by a computer system.

More than 300 groups on their own initiative have asked the Medical Systems Development Laboratory to make available to them the use of the computer program currently available for electrocardiogram analysis. Consultation has been given, but serious hardware limitations have restricted utilization within the medical setting. Duplication of laboratory type facilities such as our existing "breadboard" system is not generally practical in medical care units. Nevertheless, one group with a large staff (the Missouri Regional Medical Program), for lack of any other "off the shelf" alternative, has replicated the basic essentials of the breadboard processing configuration used at the Medical Systems Development Laboratory.

Alternative routes

It is possible to construct, from products now on the market, large regional centralized multi-purpose, multi-signal computer facilities. However, these would be impractical for routine clinical medical-signal analysis at this time because of their defects: staffing difficulties, the continuing high expense of non-local analog telemetry, high acquisition and maintenance costs, the absence of a workable time-sharing system for medical-care purposes, and the changing character of medical systems due to computer use itself. Although the technological problems can be solved, largely because of the last reason it may not be economic to expend great effort for an overall solution until 10 years hence.

This does not mean that we oppose large, re-

gionally centralized computer centers now. On the contrary they are to be encouraged, but as central storage and retrieval centers serving networks of small, dedicated computer systems. Translation of patient-care computer programs to the assembly, compiler, or machine languages of small computers for service-oriented hospitals of average size is thus a pressing vital step.

This concept implies a need for development of small single-purpose computers for processing medical signals. Our experience with prototype systems adapted for general-purpose computers has proved that a specially designed dedicated computer can best meet the anticipated needs of regional medical programs, multiphasic screening programs, medium-to-large hospitals, and local health departments.¹

Our experience with commercial groups suggests that the systems dedicated to medical-data processing can be commercially available in 2 years. A prototype, the Control Data Corporation "MESA" system, was developed from our specifications within a 6-month period about 1965 and served to clarify our concepts of the systems required for the '70s. The proposed systems should be of office-desk size, completely modular self-contained units.

A system with a processing capacity of 60,000 or more medical signals annually could be marketed by industry for a cost under \$100,000 (or rented for about \$36,000 per year). With support for the development of prototypes, a network of such computers could be established within 5 years to make routine computer analysis of electrocardiograms, spiograms, or other medical signals available anywhere in the United States. These could aid in increasing the quality of medical care and in controlling the rising cost of medical tests.

These single-purpose medical computers would facilitate the creation of storage and retrieval centers at strategic locations for much-needed medical data pools. This storage and retrieval is of course of secondary importance to immediate display for action. With data extracted by the dedicated clinical terminal from the human source and analyzed with defined accuracy, the analysis can be displayed, prepared for insertion in the patient's record, and put to immediate use. This is the basic reason for considering storage and retrieval centers as secondary in any service-oriented medical computer project.

Preliminary estimates of the basic cost of an electrocardiogram with a dedicated system, assuming a 10-year equipment amortization and 60,000 patients a year, show a unit processing cost of a few cents contrasted to dollars today. With mass volume, costs of the equipment are small when contrasted with costs of manpower training and utilization.

The breadboard system

At the Medical Systems Development Laboratory we have, as part of our work in automation of medical signals analysis, experimented with several different preprocessing and processing systems. We have kept in mind that in medicine, users or even combinations of users often do not generate sufficient quantities of data to warrant processing on large, fast computer systems. Costs for processing equipment are low as long as the system is kept busy, but are apt to be exorbitant if input rates are slower than throughput capabilities. Therefore we have chosen to work initially with dedicated, single-input systems.

Because computer personnel are not readily available at most hospitals or clinics, the system should not require skills beyond those of hospital technicians and nurses. Our experience indicates that processing can become medically reliable in nearly any properly equipped computer room, but even there only if the system is exceedingly simple to operate. Accordingly, our system specifications require no more of the technician than to load and start an analog tape playback and then check that a monitor light goes on.

Before we could begin to specify requirements for routine service equipment, it was necessary to determine many system requirements, for example, the sampling rate, the filter settings, etc. An experimental processing system was constructed having variable settings for nearly every component. After the first approximations were determined and necessary corrections made, there still remained the tasks of statistical validation, reliability and maintainability studies, human factors engineering, and analysis of operational problems encountered. To complicate matters, the original signal (the ECG) became only one of many undergoing automation at our lab. The system grew and changed, all the while becoming more versatile, but also more complex and difficult to operate and maintain. We recognized that this system would allow almost limitless varia-

tions in experimental and research techniques but was not applicable to clinical processing.

In routine operation the most difficult problem was with trouble shooting and maintenance. Basically the engineering solution hinges on the use of replaceable modules which can easily be exchanged in the event of failure. No wiring or special effort to connect the modules should be required. Repairs of components should be made at remote sites with well-equipped maintenance facilities while the user enjoys uninterrupted service. Basically the hospital or clinic should be unconcerned about maintenance except for acceptance of a preventive maintenance contract. The system must be no more complex to operate than a home television set. It should be operable with the same amount of instruction, since hospital personnel do not have the specialized skills of electronic technicians or computer operators.

The signal from the data acquisition devices is given to the first section of the system (the input unit), which contains an analog tape playback deck and monitor indicator. The tape deck uses $\frac{1}{4}$ -inch-wide tape on 7-inch reels and operates at $3\frac{3}{4}$ inches per second. After set-up, start/stop is initiated by computer control. The monitor indicator automatically glows if the tape being played was not recorded on a properly aligned data acquisition system. If the recorder head was not aligned to specifications, a 6750-Hertz flutter compensation signal used in our data acquisition systems will not be played back with sufficient amplitude. If the head is unaligned, the tape deck should stop automatically. Filter bandwidth for processing clinically used medical signals is DC to 45 Hz. The 3-db point should be at 45Hz. Rolloff above 45 Hz should be 24 db/octave.

To enable heart sound processing, a bandwidth of 40 to 1000 Hz (± 1 db) is usually suggested. A filter with the 3 db points at 20 and 2000 Hz with rolloff of 6 db/octave is recommended. Parenthetically we can state that the envelope resulting from the rectified signals may be adequate for analysis and brings all medical signals to the same range.

Amplifiers should be provided to compensate for losses incurred during transit of the signals through the filter networks and other media such as telephone lines, and to condition the signal strength and circuit impedance to the optimum values required by the analog-to-digital converter. One, a high input impedance amplifier (minimum

10 megohms), can be provided at the input to the filter networks. The second can be a signal level compensating amplifier at the output of the filter network. A calibrated gain control should be available with the compensating amplifier to enable adjustment of signal voltages to proper levels.

The experience gained from our experiments suggests commercial incorporation of some of the above listed functions either as the terminal end of a digital data-acquisition device (instead of an analog one) or the initial part of the dedicated computer as described here. The tape playback demodulator, amplifiers, filters, and/or telephone jacks are also in the input unit. An automatic answering feature can enable acceptance and recording of the telephone data without operator intervention. Automated control circuits must be provided to coordinate the various functions.

The second section is the medical-signal processor. Its stored programs should have the capability of modification by a procedure not normally available to the usual medical user. The system must utilize a minimum of external switches and human decisions. The hardware in this section consists of an analog-to-digital conversion unit, a central processing unit, data and program storage unit, printer, and the control panel. Our oscillator drives the analog-to-digital converter at a fixed sampling rate of 500 cycles per second per channel used. This sampling rate is adequate for most medical signals. An unsigned 10-bit output is adequate for accuracy purposes over the range of ± 2.5 volts. A control circuit causes the analog-to-digital converter to send a 10-bit word through the computer input register upon computer request. Initial input requests start and computer commands stop the analog tape deck.

The sub-modules must be packaged in a single housing that need occupy a space of not more than 20 square feet. The cabinet should resemble a desk and have a top usable as such. Weight of the unit, excluding the optional attachments, should not exceed 1,000 pounds, to be suitable for normal hospital and clinical building floor load.

The minimal functions that need be included are: input/output, arithmetic operations, peripheral control, logical and/or, shift, conditional

and unconditional transfers, and interrupts or timing devices.

A typewriter or other low-cost printing device, capable of a printout speed of at least 720 characters per minute, is all that is necessary for conventional work. The unit should operate under program control, with buffering a desirable feature.

An incremental plotter also under computer control could be utilized to produce a graphic output.

A capability should be provided for attachment of a telephone for transmission of digital data (i.e., the medical results obtained by the processor) to central storage and retrieval computers.

The internal memory can contain as little as a 16,000 18-bit word memory core. This would suffice for the operational program.

Provision for program entry or modification must be preplanned. Dedicated analysis systems must therefore be compatible with an existing commercially and readily available computer, to enable manufacturers to make program modifications to incorporate medical advances.

A random access storage unit can replace part of the core memory specified. Economy for the individual manufacturer should dictate the storage medium. The storage capacity will be dependent on the peripheral equipment and the computer's instruction repertoire. Approximately 16,000 18-bit word or 32,000 12-bit word capacity or equivalent would be ample.

The system should incorporate executive programs to control program selection and the flow of information between input and output devices, patient code recognition, and data formatting. Checkout programs for the on-line equipment and the data acquisition system should also be part of the software.

Utility programs to facilitate fault detection, modification of programs, and the addition of new programs are needed only by the manufacturer's checkout or maintenance crew.

The MSDL electrocardiogram program for the 160A consists of two parts. The first (pattern recognition) is approximately 6,000 lines in the OSAS assembly language. The second (diagnostic) is 8,000 assembly statements. The MSDL spiogram program consists of approximately 5,000 assembly statements. The generality and numerous logic paths incorporated in these pro-

grams indicate why translation and validation are time consuming.

SUMMARY AND CONCLUSIONS

Several factors should be considered to evaluate the system's adequacy for medical operations. The medical signal processor must permit easy installation and keep maintenance requirements to a minimum. The unit must be capable of being maintained by replacement of easily removed components and modules. Reliability of operation must be considered to be of prime importance in the design and manufacture of any medical signal processor. Commercial sales cannot be expected to be great without due emphasis on this aspect. The validation must demonstrate the ability of the system to handle approximately 60,000 medical signals over a 1-year period when operated on a continuous basis (exclusive of preventive maintenance time).

Large-scale screening or research projects both require utilization of more than one signal. In

lieu of "simultaneous" processing with one device or a serial system, small dedicated systems used in parallel can achieve the high throughput rates required to keep up with voluminous data. Currently we believe these will offer the most economic courses to follow to solve existing volume problems. A central storage and retrieval system can then collate the data.

The medical community can be provided with a standardized processing system for medical signals to meet volume requirements of health care needs. Diagnostic printouts can be made in the standard format and terminology familiar to every physician. This alone will provide invaluable quality control of medical service delivery. Combinations of small machines can be effective in producing the required volume and quality on a sound economic basis.

REFERENCES

- 1 M V MATHEWS
Choosing a scientific computer for service
Science 161:23 1968

A legal structure for a national medical data center*

by ROY N. FREED

Harbridge House, Inc.
Boston, Massachusetts

INTRODUCTION

Increasingly, medical professionals are concluding that a computerized national center to contain medical records of all persons has significant medical advantages. Moreover, such a project seems to be becoming technically feasible. However, uncertainty about the adequacy of physical and legal protections of the privacy of the stored data (and the related compatibility of the system with the Fifth Amendment bar to involuntary self-incrimination) appears to be the major stumbling block at this time. Study of that aspect reveals that it is possible to achieve a level of privacy and an observance of Constitutional requirements that are entirely satisfactory on an absolute basis and especially in light of both the great social benefits in store and the degree of protection customary for the particular information involved. This article provides a blueprint of the types of legal inhibitions that should be sufficient for the purpose in view of technical measures available.

To facilitate evaluation of the proposals being made, a fairly broad description of the operation of a national medical data center is set forth. Next, the types of potential jeopardies to privacy and Constitutional protections are indicated. Then, suggested technical protective measures are detailed. Finally, recommended legal steps and associated compliance audits are treated.

It is hoped that this discussion will stimulate an exchange of views on the important factor of privacy of the proposed recorded data so that a consensus can be reached fairly promptly on an acceptable approach. The potential medical benefits of the suggested central file are so great, both for the persons whose records would be available through the system and for society as a whole, that the project should be initiated at the earliest possible moment.

*The opinions expressed in this paper are entirely those of the author and are not to be attributed to Harbridge House, Inc.

The national medical data center

Many responsible and respected professionals concerned with health care believe strongly that a national center should be established to contain the medical records of as many persons in this country as possible. This section describes the type of system presently envisioned in sufficient detail to provide a basis for the legal proposals made later. Of course, if a system is set up, its actual features well might differ from the following description of the writer's concept in many respects.

Medical records would be kept in a very large central file for all persons who decide voluntarily, either personally or through their legal representatives (in the case of minors or incompetents), to take advantage of the opportunity. Such a decision would be manifested by signing a request form after at least reading a statement of the medical benefits and privacy risks involved in participation.¹ Participants** would be free to discontinue adding to their records at any time, simply by directing their doctors not to submit information. (The fact of discontinuance should be noted in the record.) They also could ask, at any time, for the purging of their part of the file. Since the patient's medical record has been considered to be the property of the doctor keeping it (subject to use for the benefit of the patient), the removed material probably would be sent to a healing arts practitioner designated by the withdrawing participant rather than to the participant himself.

Based upon voluntary participation, largely for their own benefit, of persons whose own records would be stored, the medical data center is different, in that respect, from practically all other proposed data centers.² However, the fact should have no significance in determining the extent to which privacy and Constitutional protections should be provided. It should not be neces-

**See end of paper for footnotes numbered 1-11.

sary to sacrifice those protections in order to enjoy better medical care.

Incidentally, patient consent might not be legally essential for practitioners to file medical information with the center. Practitioners may do so, in their own discretion (which essentially could represent a decision to use the center as their own file), subject only to their general legal obligation to protect the privacy of their patients. Lawyers, who have similar professional duties to persons they serve, may disclose client identity to outside data processors.³ The practitioners' obligation requires that they avoid disclosing medical information associated with the patient's identity to others without legal authorization. Such authorization covers disclosures of various types of identified sensitive information to the patient's family or prospective spouse (mental illness or venereal disease), government agencies (contagious diseases), and the police (apparent foul play). Practitioners who breach that obligation are subject to lawsuits for the penalty of money damages, which might not be provable. Despite the likelihood that they might be free to act on their own decision, most practitioners undoubtedly would prefer the validation of express patient consent.

It would have to be decided whether participants may pick and choose information to be included in their records while they are participating. There is much merit in deciding that only a practitioner may make the determination of what will be entered, especially if the data are to be used also by researchers and the Food and Drug Administration. This would tend to protect the reliability of the information. Moreover, that approach is similar to current practice with respect to records kept by doctors themselves.

Since this article is intended to treat only the privacy aspects of the system, rather than its technical or economic feasibility, no consideration will be given here to those two important facets. They undoubtedly will be discussed in detail elsewhere, after it has been shown how adequate privacy can be achieved. Actually, technical measures necessary for protection of privacy must be selected before accurate costs can be calculated.

The data would be stored in the center's records in binary code in a large computer system so that they can be located rapidly and accurately and transmitted directly to inquiring practitioners and can be supplemented easily as further data become available. Information would be sent in in binary code in some convenient manner and would be secured from the file by means of display-printer terminals (like TV screens) readily accessible to practitioners over the country. Very likely, data communication techniques would be used in both directions.

Each participant would be identified by his Social

Security number probably supplemented by the first four letters of his last name.⁴ Similarly, all practitioners in the country who would want to inquire about or add to the stored data would register themselves by filing their own Social Security numbers and information on their licenses to practice a healing art.⁵ The licensing authorities so identified would be notified of such registrations and would be requested to inform the center when the registrants cease to be practitioners as a result of death, disqualification, or other circumstances. Likewise, vital statistics bureaus would be requested to report deaths to the center for entry in participants' records and for correction of the list of practitioners. Because, as indicated below, practitioners might have to be issued replacement identification means from time to time, their codes might include their Social Security numbers plus supplementary numbers or symbols that might be varied occasionally.

Medical information entered in the system would be alphanumeric only (although it might produce graphical and tabular data as well as narrative statements). The precise formal nature of entries would be determined by a study of the feasibility of using standard abstracts, formats, and nomenclatures.⁶ The information undoubtedly would include vital statistics, physical conditions with medical significance, diseases, treatments, allergies, immunizations, doctors' and hospitals' names, and like details. It probably also would have descriptions and locations of important physical items containing information, such as X-ray films and tissue slides.

The stored data would make it possible to provide full medical histories to the participants' practitioners at all times, regardless of how much the particular person has moved or of the fact that he is being seen, in an emergency, by a practitioner unacquainted with him. (It would be advisable for persons participating in the system to carry their identification numbers or wafers at all times.) Under the system, it would be unnecessary for practitioners to keep records on their patients except for data in addition to that stored in the system. Hospitals also might be able to reduce the size of their files in the same manner, at least between confinements.

The ready availability of the data should make it possible to provide improved medical care in many cases based on more information. It also might furnish the means for learning of experiences with specific medications, for locating persons who have received treatments subsequently discovered to be harmful,⁷ and for performing similar functions.

Direct access, through the terminals, to the stored data of specific, identified participants would be permitted only to practitioners currently licensed, subject to the legal safeguards suggested below. To identify

themselves, practitioners would use embossed plastic wafers (like credit cards) or similar devices that can be read by machine. If additional precautions to control access are desired, it might be arranged for each participant to have an identification wafer that would have to be inserted into the terminals, along with the practitioner's wafer, to release his information. However, that measure might prove to be too cumbersome to be useful and might not be essential.

It would have to be decided whether participants would be permitted to receive print-outs of their data routinely, directly by mail, upon written request, rather than through their practitioners. It might be determined that, for medical reasons, at least some types of particularly sensitive medical information, if not the entire record, would not be released directly to participants. It appears to be current practice for medical file information to be made available to patients through practitioners (except for hospital records subject to inspection under statutory or other legal authority).

As indicated, it must be recognized that the center file would represent a medical information resource much too valuable to be reserved solely for clinical practitioners. It would be a veritable gold mine for researchers, the Food and Drug Administration, the Public Health Service, and others, since it could disclose data, probably unavailable elsewhere either at all or to a similar extent, on experience with particular treatments, on incidence of medical difficulties,⁸ and on other very pertinent matters. To protect privacy, such data could be made available directly by the center only to qualified researchers and Government regulatory officers and then without an indication of the participants' names. The computer can omit those names very easily. If some identification is required for follow-up, special code numbers, whose deciphering can be controlled by the center, could be used.

To satisfy Constitutional inhibitions against compulsory self-incrimination, direct access to the identified stored data furnished by participants would be denied specifically to all government agencies. In this manner, those agencies would be prevented from using the data in criminal proceedings against a participant without his voluntary action.

So much for the general nature of the system. It now is in order to identify and evaluate the types of jeopardies to privacy and Constitutional protections of the stored data that seem to be involved.

Jeopardies to privacy and constitutional protections

Privacy and Constitutional protection of data stored by the center ostensibly could be in jeopardy in a number of different ways. Probably the major concern

would be over possible intentional, direct access to personal data by government personnel or by many types of persons in the private sector, such as employers, credit agencies, insurance companies, and private investigators. Also important is the hazard of accidental disclosure of information to unauthorized persons. Finally, there is the danger of unauthorized access through the tapping of communications lines. It is helpful to assess the likelihood of those types of intrusions and to compare the potential jeopardies they represent with the present degree of privacy of the same kinds of medical data.

Since neither the kinds of data to be stored nor the jeopardies are novel, it is interesting to note why a unique problem seems to arise from the establishment of the proposed data center. To many, the sheer assembling of the information in one place converts a matter of degree into a matter of real difference. Solely by maintaining the current fragmentation of medical files, it appears as if the information is less readily reachable by persons not entitled to it. This probably is true if there is a real danger that the central file will be used improperly to locate specific persons for wrongful purposes by categories of information (such as type of disease, income, or location of residence).

Actually, the present repositories, which are doctors' offices and hospital record rooms, are no more physically secure than the proposed data center and they even might be less so in many specific situations. Furthermore, they are vulnerable not only to physical intrusions and ruses but also to use of government influence, especially of police agencies. It is important to determine if there is any actual difference and hence any greater danger to privacy and other legal rights from operation of a national center.

Properly, the vulnerabilities of privacy through present and proposed recordkeeping systems cannot be compared in general. The comparison can be made only in terms of specific recorded information sought to be protected. Information so assessed has two aspects pertinent to this inquiry, its variety (or comprehensiveness) and its location.

In many situations, only one particular medical item would be sought by an unauthorized person, and it would be in only a single file. That file might be more or less secure than the proposed central repository, depending upon the circumstances. It well could be much less secure in view of traditional handling of files by doctors and hospitals. If the person seeking the information knows where the file is located, then it might be more vulnerable than a central file. However, if he does not know, then it is uncertain whether locating and securing the individual file is more difficult than pene-

trating the central record. That generally is one type of circumstance.

On the other hand, where the improperly desired data are stored among fragmented files of the present type, then the relative difficulties of entering many files and the proposed central file should be weighed. If it is difficult to identify which small files contain the data, then use of a central file at least indicates most readily the one that must be attacked.

The other risks indicated include inadvertent disclosure (unassociated with ruses) and tapping of communications lines. These risks probably are minimal in present methods of medical recordkeeping because information is transmitted by wire relatively infrequently and entirely sporadically and randomly and inquiries usually are made directly to people. When computers are used eventually, apart from the national center, to keep medical records for hospitals and quite likely for doctors in private practice as well, the danger of inadvertent disclosure will become greater unless, as is to be expected, special precautions are taken.⁹ Then, wire communication would be used much more than presently. In view of these likely developments, the dangers of a central file should be compared more properly with those of non-centralized computerized medical recordkeeping, rather than with present methods. And even in the ultimate situation, it is difficult to envision that wire tapping for selective information would be practical in view of the vast volume of communication that would be involved. Furthermore, in many situations tapping of data communications will be substantially more difficult than of voice messages.

The existence of legal inhibitions and of means for auditing compliance with them also is pertinent to a comparison of present and potential privacy jeopardies to medical records. Presently, legal bars to improper disclosure exist almost exclusively in the common law rulings of the courts, and penalties, which are civil rather than criminal, are uncertain because of the difficulties of valuing the damages suffered in terms of dollars. Also, few, if any, means are provided now for undertaking to detect privacy breaches. Hence, proposed legal measures well might be much more effective in discouraging intrusions and in setting up ways to discourage violations.

In point of fact, then, the absence of factual data on actual experience in the areas of risk identified makes it impossible to state with any conviction whether the present or the proposed method of storing medical data presents a greater danger to the privacy and Constitutional rights of the persons involved.

Nevertheless, despite the impossibility of concluding, at this time, that a central medical file introduces a new or greater risk in that respect, it is very much in order to

consider the numerous technical protective measures available in the operation of such a file.

Technical protective measures

The effectiveness of technical measures to protect privacy and Constitutional rights when a data center of any sort is operated is influenced by developments in electronic digital computer and communications technologies. As the technologies evolve, protective means that are more effective and economical become available. Of course, it is unreasonable to recommend the adoption of a weak system in the expectation that necessary features will come into existence, and that is not being done here. Actually, substantial protective measures exist already, and they can be expected to be improved. It appears that those present measures, in conjunction with legal steps recommended below, will achieve the full level of protection realistically required for a medical data system.

There are numerous physical protections against intentional and inadvertent file intrusions.¹⁰ They can be evaluated most effectively in light of the risks they are intended to cover. Some of those risks can be suggested by the types of precautions that are required. It would be necessary, for example, to restrict direct access to licensed practitioners, to bar impersonation of authorized practitioners, to prevent intentional physical intrusion into the area where the records are stored, and to avoid accidental delivery of information different from that requested. The specific measures that are feasible for those purposes can be examined.

Access to the file would be secured only by the use of an identification wafer issued to a licensed practitioner. When the center learns that a practitioner ceases to be authorized or that his wafer is out of his possession, the acceptability of that wafer could be terminated and inquiries with which it is used would be rejected.¹¹ The assignment of code identifications and the creation of wafers should be restricted to a small group of center personnel who are subject to rigid controls.

It probably would be desirable to restrict the terminals from which inquiries may be made so that access could not be gained from outside terminals. To exclude intrusion into the wire network by means of other terminals, authorized input-output devices might create unique signals that can be recognized by the central system. It even might be possible to arrange that individual practitioner's wafers would be acceptable only through particular terminals. This probably would be entirely satisfactory except in the rare cases in which a practitioner undertakes emergency care away from home.

It should be possible to establish strict security con-

trol over the premises in which the center's records are stored so that physical intrusions would be difficult.

If participants' wafers are required, along with those of practitioners, in making inquiries, programming measures would insure that only the proper files would be interrogated. In order to provide for emergencies in which the participant's wafer is unavailable, a special override should be provided for. If that approach is taken, uses of that privilege could be monitored to detect abuse.

All inquiries should be recorded in each participant's record, without exception, and the opportunity to make programming changes should be controlled strictly. In this way, unauthorized inquiries originating at the center itself would be revealed.

It also might be advisable to treat particularly sensitive data (such as that on mental illness or venereal disease) specially, so that it would be available only under very limited circumstances, rather than routinely.

Although communications lines normally are the weakest link because of the possibility of tapping, automatic encrypting of messages during transmission is possible. However, unless there is great danger of intrusion in that area, which seems to be very unlikely, the substantial cost of that measure probably could not be justified.

The array of technical steps to protect privacy just described indicates that computer technology provides means of considerable effectiveness in view of the particular risks involved. It also appears that the risks entailed in the proposed center are not clearly greater than those presently experienced or that will be encountered with increased computerization of fragmented records. In fact, the study of the feasibility of the center, which hopefully will take place soon, probably will reveal, as frequently is the case, that current arrangements for keeping medical records warrant greater attention to privacy, if the center is not set up soon.

Legal protective measures

Clearly, physical protections alone would not be sufficient. People involved in the operation of the system, including practitioners and center personnel, still would have considerable opportunities and incentives to breach the privacy of participants. Those people undoubtedly are the weakest elements in the system. Hence, the physical steps must be accompanied by legal measures to control the conduct of those people. A number of those measures are suggested at this point.

As in the case of the technical steps, it probably will be helpful first to identify the risks those legal restrictions are intended to cover. They include primarily the dangers that persons enjoying access to the stored data (including both practitioners and center personnel)

might misuse their status to pass information on to unauthorized persons and that unauthorized persons might try to force their way into the system. To minimize those risks, it is advisable to establish a series of penalties that will have very substantial deterrent values.

Since the center would function nationwide and since its sponsorship is not yet identified, it is difficult to suggest the precise controlling legal structure that should be sought. There are a number of possible routes to arrive at the most reasonable set of legal rules and means for enforcing compliance compatible with practical limitations. These routes include legislation, contract (including regulations adopted by the center), reliance upon the common law (as created by the courts), and various combinations of those approaches. As usual, each of the routes has its deficiencies.

Although Federal legislation might be attractive because it ostensibly would promulgate all the required rules in one fell swoop for the entire country, there are a number of drawbacks to that panacea. Resort to Federal help is resisted in some influential quarters. Furthermore, many measures might be considered inappropriate for Federal action, particularly as they would add to the business of the Federal courts. Where legislation is deemed essential, the alternative is state enactment. But persuading all states to adopt laws individually could involve an extended, if not futile, effort. Support of the National Conference of Commissioners on Uniform State Laws might be enlisted in such an effort.

Since involvement in the center's program would be attractive to practitioners and center employees, much reliance could be placed on contracts with them. However, contractual commitments would bind only their parties, thus leaving many types of troublesome persons unrestrained by that means. Furthermore, penalties for contract breach might be less effective than criminal sanctions and valuable injunctive relief might be unavailable.

By their rulings in individual cases not covered by statute law, courts customarily have provided substantial protections of legal rights through the common law. For example, they have molded most of the various rights of privacy and some of the related privileges against disclosure of information. Thus, much assistance to the center's operations could be expected from the courts in the absence of legislation. However, the nature of the legal rules that will arise in that manner cannot be anticipated in advance and cannot be expected to be uniform among the states, at least for some time to come.

While the strategy is being planned for achieving the necessary legal rules that would make the contemplated

center viable, it should be helpful to note the specific undesirable conduct vis-à-vis the center's operations that should be discouraged. The acts appropriate for criminal or civil condemnation are set out below, and possible sanctions are considered.

Ideally, the following conduct should be made criminal, but much protection would be achieved if such action of center personnel and practitioners were barred merely by contract:

1. Delivery by a practitioner or center personnel of any information secured from the center to a person who is not authorized to receive it under then existing law.
2. Improper intrusion into the system in any way, either directly or through practitioners or center personnel.
3. Counterfeiting of an identification wafer of a participant or practitioner.
4. Use by a practitioner of his identification wafer while he is not authorized to do so.
5. Use of someone else's identification wafer under any circumstances.
6. Receipt or use of specifically identified information secured from the center for any purpose other than (a) to provide medical care to the person from whose record it came or (b) to transmit it under circumstances to which a legal privilege applies under then existing law.

Similarly, it should be made the tort of invasion of privacy against a participant (or at least a breach of contract, as suggested above) for a practitioner or center employee to deliver his medical information to any person not specifically authorized to receive it either by the participant or by the laws on privilege. It likewise should be made such a tort for a person to receive medical information under those circumstances. Depending upon the circumstances, such a person might be guilty of inducing breach of the contract of the center.

To make the foregoing criminal, civil, and contract prohibitions effective, the following penalties probably should be imposed by statute and remedies provided by contract:

1. Substantial fines and imprisonment should be established as punishment for violations.
2. Civil suits for money damages and for injunctions should be authorized to be brought by the injured participants or by the center on their behalf. Otherwise, the center would have to sue for breach of its contract and liquidated damages should be provided for to assure that the penalty will be significant.
3. Practitioners who misuse information they secured

from the center should be barred from use of the system.

4. The writing of malpractice insurance to cover civil or criminal losses suffered by practitioners due to illegal conduct specified above might be forbidden.
5. Data secured from the center that were supplied by a participant to his practitioner should be unavailable to police and inadmissible in criminal proceedings against a participant without his consent.
6. All discovery measures (including orders for the production of documents, subpoenas duces tecum, interrogatories, and depositions) directed to the national center and relating to stored data should be forbidden.

Brief explanation probably is in order concerning the recommendation that specified conduct be made criminal. On the one hand, some acts are made criminal that are only civil wrongs presently, such as the improper delivery by a practitioner of medical information about a patient. The more stringent approach seems to be warranted in view of the notion that greater harm can be perpetrated by using the assembled data. On the other hand, the securing of information from center personnel or employees of practitioners or hospitals improperly by paying money or giving other valuable consideration probably is covered by state statutes forbidding commercial and government bribing. However, penalties for that conduct usually are relatively small and hence probably are ineffectual. Reference was made to the possibility that delivery of center data by a practitioner to an outsider might be privileged. Statutes and case law authorize a doctor to tell persons standing in the position of the family and even a prospective spouse that his patient has a venereal disease, a mental illness, or another unusual medical condition.

The precedent of the collection of census information should indicate that operation of a national data file is not ipso facto a danger to privacy. The Census Bureau apparently has adhered scrupulously to legal bars to its disclosure of specific information to anyone. With all of the Congressional oversight of Government agencies and the many incentives to engage in muckraking, no departure from the rules has been revealed. However, that experience probably is relevant only as the reliability of center personnel is concerned.

The penalties and civil remedies also warrant a few words. Criminal penalties should be severe enough to discourage violations. Substantially all violations would be inspired by hope of personal gain and hence might be deterred by making the risk sufficiently great. Although many privacy breaches are remedial by money damages today, injunctive relief normally is not

thought of. That remedy should be provided specifically. It would be useful in preventing further harm through the misuse of purloined medical data, for example, by requiring the taking of action on an insurance or credit application without regard to the information secured improperly or the purging of the recipient's files of such information.

More novel is the recommendation that guilty practitioners be barred from using the center's system. That could be a severe and very effective penalty since use of the system would be an essential element of medical practice. Participants would want current medical data entered in the file and hence would shun a practitioner who could not accomplish that, especially if he were barred for data misuse.

The denial of malpractice insurance coverage for criminal privacy breaches involving center data is a severe penalty that might harm the injured participant more than it helps to prevent his injury. It might be assumed that lack of such insurance protection will deter practitioners from flirting with privacy breaches by exposing their personal resources to legal judgments. Many practitioners who would engage in violations probably would not be influenced by such a prospect and might be so marginal financially, as well as ethically that they could not compensate participants they injured.

Respect for the spirit of Fifth Amendment protection would seem to require that data in the center's records furnished directly by a participant to a practitioner not be reachable by government prosecutors. That being so, prosecutors should not be able to use such data in evidence against participants.

Finally, it might be advisable to authorize the center to bring civil suits on behalf of participants whose privacy has been breached, especially if the intrusion were perpetrated by its own personnel. This remedy would be similar to that provided under the Wage and Hour Law for recovery of back pay through Government suits. Since many injured persons might lack resources to finance a suit, the assistance of the center could be very appropriate. Very similar results could be achieved if the center were to pay the legal expense of participants who seek such relief directly, but the center's cost might be much greater. Of course, if the center would want to sue because of breach of its contracts, it would need no special authorization.

Because injured participants frequently would not know of the privacy breaches (as usually they do not know presently of harm suffered at the hands of credit agencies using incorrect information), provision should be made for audits of compliance with the legal rules. Even if participants were able to secure copies of their own data, many could not analyze the entries that

reveal inquiries and identify any suspicious patterns. Moreover, they could not see the records of others, which might be necessary to uncover the abuse. Hence, only the center itself or an outside responsible agency, in the nature of an ombudsman, could conduct effective audits to apprehend guilty practitioners or staff personnel.

The audits would include such action as analyses of inquiries of practitioners to look for suspicious access, checks on internal activities involving program changes that might be used for improper inquiries, and periodic direct communication with participants and practitioners to verify the genuineness of inquiries. A correctly designed and efficiently executed audit program should be able to detect most of the relatively few privacy breaches that will occur. Awareness of the effectiveness of such a program should deter attempts to perpetrate breaches.

Also pertinent to privacy of center data is the availability of that data in legal proceedings, to which brief reference already has been made. Presently, in many states, the physician-patient privilege bars disclosure by the doctor, without the patient's consent, of information given by the patient to the doctor in a medical relationship. In civil actions, the patient himself usually may refuse to reveal the information he so gave to his doctor. In criminal cases, however, it would be protected from involuntary disclosure by the patient by the right against self-incrimination.

There is good reason to apply to the records kept in the proposed center the present privilege rule and related protections against disclosure even by the patient himself involuntarily. For that purpose, discovery measures, such as orders for the production of documents, subpoenas duces tecum, and interrogatories, would not be authorized to be directed to the center for any data. They would be available against the participant himself and his practitioners, within the framework of existing protective rules. Since doctors can be compelled, through legal process, to testify about medical items other than information revealed by the patient, which would not exclude the fact that a consultation occurred, discovery measures should be permitted to be directed to the practitioner.

When patients are confronted with discovery related to center data, they could rely upon their privilege and Constitutional protections, just as they do now. Similarly, at present, it frequently is left to the doctor to raise the barrier of the privilege in the first instance, although that patient's lawyer very often does so. The same procedure well could apply with respect to center data. Hence, discovery measures would be usable with respect to information in the proposed system with little, if any, need to adopt special protective rules.

Both these results can be achieved largely by regarding the center records legally to be the files of practitioners rather than separate files with independent legal status.

CONCLUSION

This review indicates that it is in order to pursue investigations of the feasibility of a national medical data center without concern that privacy and Constitutional considerations constitute an insuperable barrier. Available technical protective means bolstered by acceptable legal measures assure at least as much respect for individual rights in medical data as is enjoyed at present, if not actually more. The great health care benefits foreseen by proponents of the project warrant early attention to it and would seem to provide whatever justification is needed for the legislative action recommended.

FOOTNOTES

1. This approach resembles the requirement that persons investing in securities subject to a registration statement filed with the Securities and Exchange Commission receive a copy of the prospectus.

2. It thus is distinguished from employment and credit clearing houses.

3. Informal Opinion 1002, Committee on Professional Ethics,

American Bar Assn., 54 A. B. A. J. 474 (May 1968).

4. This is the system used by the Internal Revenue Service in its ADP system for processing tax returns.

5. Realistically, such participation would be open to many types of practitioners in addition to medical doctors. Hence, not only the name of the person submitting data but also the nature of his treatment approach should be recorded to guide subsequent inquirers in evaluating the record.

6. Such a study is in progress under Burgess L. Gordon, M.D., on the staff of the American Medical Association. See Gordon, B. L., Biomedical Language and Format, 39 *Medical Record News*, No. 2, (April 1968), p. 34.

7. In *Schwartz v. United States*, 230 F. Supp. 536 (E. D. Pa. 1964), it was held that doctors have a duty to review prior records readily available and to warn patients of earlier treatment discovered to be harmful.

8. At its Atlanta office, the National Communicable Disease Center of the Public Health Service collects data pertinent to its surveillance of epidemics.

9. For a discussion of some such precautions, see Freed, R. N., "Written signatures block the computer revolution," *Modern Hospital* (July 1968), p. 103.

10. Ware, W., "Security and privacy in computer systems," Peters, B., "Security considerations in a multi-programmed computer system," Ware, W., "Security and privacy: similarities and differences," and Peterson, H. and R. Turn, "System implications of privacy." Proceedings AFIPS 1967 Spring Joint Computer Conf., pp. 279-300.

11. Such attempted misuse should trigger an investigation into an apparent effort to violate the system, especially if identification wafers are used with the result that innocent errors in the handling of identification codes are avoided.

A research center for augmenting human intellect*

by DOUGLAS C. ENGELBART
and WILLIAM K. ENGLISH

Stanford Research Institute
Menlo Park, California

1 SUMMARY

1a This paper describes a multisponsor research center at Stanford Research Institute in man-computer interaction.

1a1 For its laboratory facility, the Center has a time-sharing computer (65K, 24-bit core) with a 4.5 megabyte swapping drum and a 96 megabyte file-storage disk. This serves twelve CRT work stations simultaneously.

1a1a Special hardware completely removes from the CPU the burden of display refreshing and input sampling, even though these are done directly out of and into core.

1a1b The display in a user's office appears on a high-resolution (875-line) commercial television monitor, and provides both character and vector portrayals. A relatively standard typewriter keyboard is supplemented by a five-key handset used (optionally) for entry of control codes and brief literals. An SRI cursor device called the "mouse" is used for screen pointing and selection.

1a1b1 The "mouse" is a hand-held X-Y transducer usable on any flat surface; it is described in greater detail further on.

1a2 Special-purpose high-level languages and associated compilers provide rapid, flexible development and modification of the repertoire of service functions and of their control procedures (the latter being the detailed user

actions and computer feedback involved in controlling the application of these service functions).

1b User files are organized as hierarchical structures of data entities, each composed of arbitrary combinations of text and figures. A repertoire of coordinated service features enables a skilled user to compose, study, and modify these files with great speed and flexibility, and to have searches, analyses data manipulation, etc. executed. In particular, special sets of conventions, functions, and working methods have been developed to air programming, logical design, documentation, retrieval, project management, team interaction, and hard-copy production.

2 INTRODUCTION

2a In the Augmented Human Intellect (AHI) Research Center at Stanford Research Institute a group of researchers is developing an experimental laboratory around an interactive, multi-console computer-display system, and is working to learn the principles by which interactive computer aids can augment their intellectual capability.

2b The research objective is to develop principles and techniques for designing an "augmentation system."

2b1 This includes concern not only for the technology of providing interactive computer service, but also for changes both in ways of conceptualizing, visualizing, and organizing working material, and in procedures and methods for working individually and cooperatively.

*Principal sponsors are: Advanced Research Projects Agency and National Aeronautics and Space Agency (NAS1-7897), and Rome Air Development Center F30602-68-C-0286.

2c The research approach is strongly empirical. At the workplace of each member of the subject group we aim to provide nearly full-time availability of a CRT work station, and then to work continuously to improve both the service available at the stations and the aggregate value derived therefrom by the group over the entire range of its roles and activities.

2d Thus the research group is also the subject group in the experiment.

2d1 Among the special activities of the group are the evolutionary development of a complex hardware-software system, the design of new task procedures for the system's users, and careful documentation of the evolving system designs and user procedures.

2d2 The group also has the usual activities of managing its activities, keeping up with outside developments, publishing reports, etc.

2d3 Hence, the particulars of the augmentation system evolving here will reflect the nature of these tasks—i.e., the system is aimed at augmenting a system-development project team. Though the primary research goal is to develop principles of analysis and design so as to understand how to augment human capability, choosing the researchers themselves as subjects yields as valuable secondary benefit a system tailored to help develop complex computer-based systems.

2e This "bootstrap" group has the interesting (recursive) assignment of developing tools and techniques to make it more effective at carrying out its assignment.

2e1 Its tangible product is a developing augmentation system to provide increased capability for developing and studying augmentation systems.

2e2 This system can hopefully be transferred, as a whole or by pieces of concept, principle and technique, to help others develop augmentation systems for aiding many other disciplines and activities.

2f In other words we are concentrating fully upon reaching the point where we can do all of our work on line—placing in computer store all of our specifications, plans, designs, programs, documentation, reports, memos, bibliog-

raphy and reference notes, etc., and doing all of our scratch work, planning, designing, debugging, etc., and a good deal of our intercommunication, via the consoles.

2f1 We are trying to maximize the coverage of our documentation, using it as a dynamic and plastic structure that we continually develop and alter to represent the current state of our evolving goals, plans, progress, knowledge, designs, procedures, and data.

2g The display-computer system to support this experiment is just (at this writing) becoming operational. Its functional features serve a basic display-oriented user system that we have evolved over five years and through three other computers. Below are described the principal features of these systems.

3 THE USER SYSTEM

3a Basic Facility

3a1 As "seen" by the user, the basic facility has the following characteristics:

3a1a 12 CRT consoles, of which 10 are normally located in offices of AHI research staff.

3a1b The consoles are served by an SDS 940 time-sharing computer dedicated to full-time service for this staff, and each console may operate entirely independently of the others.

3a1c Each individual has private file space, and the group has community space, on a high-speed disc with a capacity of 96 million characters.

3a2 The system is not intended to serve a general community of time-sharing users, but is being shaped in its entire design toward the special needs of the "bootstrapping" experiment.

3b Work Stations

3b1 As noted above, each work station is equipped with a display, an alphanumeric keyboard, a mouse, and a five-key handset.

3b2 The display at each of the work stations (see Figure 1) is provided on a high-resolution, closed-circuit television monitor.

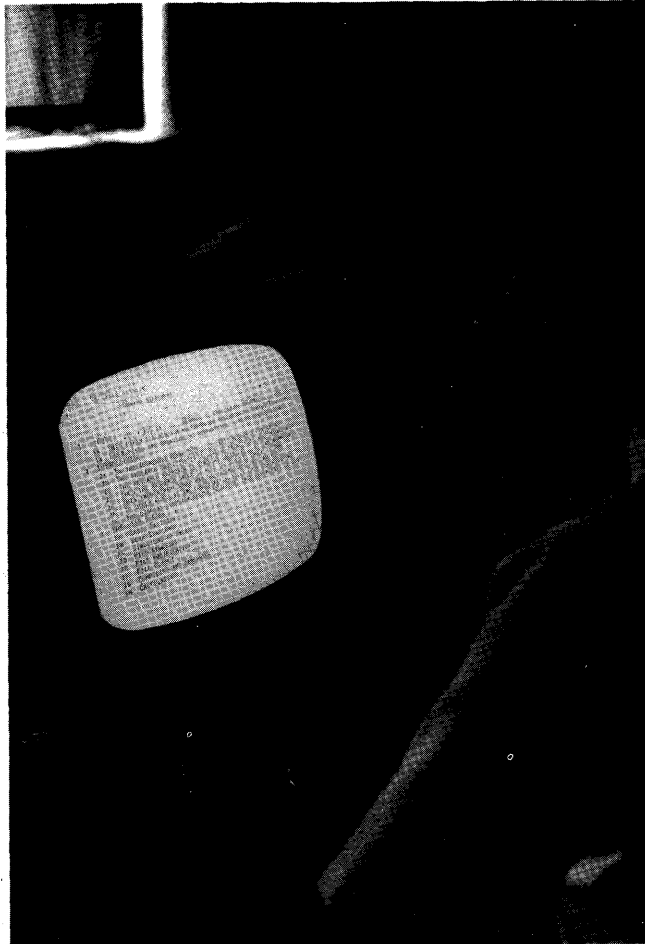


FIGURE 1—Typical work station, with TV display, typewriter keyboard, mouse, and chord handset

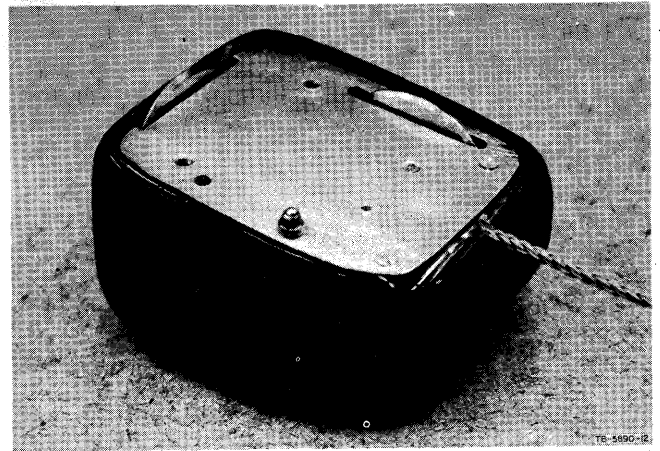


FIGURE 2—Underside of mouse

3b3 The alphanumeric keyboard is similar to a Teletype keyboard. It has 96 normal characters in two cases. A third-case shift key provides for future expansion, and two special keys are used for system control.

3b4 The mouse produces two analog voltages as the two wheels (see Figure 2) rotate, each changing in proportion to the X or Y movement over the table top.

3b4a These voltages control—via an A/D converter, the computer's memory, and the display generator—the coordinates of a tracking spot with which the user may "point" to positions on the screen.

3b4b Three buttons on top of the mouse are used for special control.

3b4c A set of experiments, comparing (within our techniques of interaction) the

relative speed and accuracy obtained with this and other selection devices showed the mouse to be better than a light pen or a joystick (see Refs. English 1 and English 2).

3b4c1 Compared to a light pen, it is generally less awkward and fatiguing to use, and it has a decided advantage for use with raster-scan, write-through storage tube, projection, or multiviewer display systems.

3b5 The five-key handset has 31 chords or unique key-stroke combinations, in five "cases."

3b5a The first four cases contain lower- and upper-case letters and punctuation, digits, and special characters. (The chords for the letters correspond to the binary numbers from 1 to 26.)

3b5b The fifth case is "control case." A particular chord (the same chord in each case) will always transfer subsequent input-chord interpretations to control case.

3b5c In control case, one can "backspace" through recent input, specify underlining for subsequent input, transfer to another case, visit another case for one character or one word, etc.

3b5d One-handed typing with the handset is slower than two-handed typing with the standard keyboard. However, when the user works with one hand on the handset and one on the mouse, the coordinated in-

terspersion of control characters and short literal strings from one hand with mouse-control actions from the other yields considerable advantage in speed and smoothness of operation.

3b5d1 For literal strings longer than about ten characters, one tends to transfer from the handset to the normal keyboard.

3b5d2 Both from general experience and from specific experiment, it seems that enough handset skill to make its use worthwhile can generally be achieved with about five hours of practice. Beyond this, skill grows with usage.

3c Structure of Files

3c1 Our working information is organized into files, with flexible means for users to set up indices and directories, and to hop from file to file by display-selection or by typed-in file-name designations. Each file is highly structured in its internal organization.

3c1a The specific structure of a given file is determined by the user, and is an important part of his conceptual and "study-manipulate" treatment of the file.

3c2 The introduction of explicit "structuring" to our working information stems from a very basic feature of our conceptual framework (see Refs. Engelbart1 and Engelbart2) regarding means for augmenting human intellect.

3c2a With the view that the symbols one works with are supposed to represent a mapping of one's associated concepts, and further that one's concepts exist in a "network" of relationships as opposed to the essentially linear form of actual printed records, it was decided that the concept-manipulation aids derivable from real-time computer support could be appreciably enhanced by structuring conventions that would make explicit (for both the user and the computer) the various types of network relationships among concepts.

3c2b As an experiment with this concept, we adopted some years ago the convention of organizing all information into explicit

hierarchical structures, with provisions for arbitrary cross-referencing among the elements of a hierarchy.

3c2b1 The principal manifestation of this hierarchical structure is the breaking up of text into arbitrary segments called "statements," each of which bears a number showing its serial location in the text and its "level" in an "outline" of the text. This paper is an example of hierarchical text structure.

3c2c To set up a reference link from Statement A to Statement B, one may refer in Statement A either to the location number of B or to the "name" of B. The difference is that the number is vulnerable to subsequent structural change, whereas the name stays with the statement through changes in the structure around it.

3c2c1 By convention, the first word of a statement is treated as the name of the statement, if it is enclosed in parentheses. For instance, Statement O on the screen of Figure 1 is named "FJCC."

3c2c2 References to these names may be embedded anywhere in other statements, for instance as "see (AFI)," where special format informs the viewer explicitly that this refers to a statement named "AFI," or merely as a string of characters in a context such that the viewer can infer the referencing.

3c2c3 This naming and linking, when added to the basic hierarchical form, yields a highly flexible general structuring capability. These structuring conventions are expected to evolve relatively rapidly as our research progresses.

3c3 For some material, the structured-statement form may be undesirable. In these cases, there are means for suppressing the special formatting in the final print-out of the structured text.

3c4 The basic validity of the structured-text approach has been well established by our subsequent experience.

3c4a We have found that in both off-line and on-line computer aids, the concep-

tion, stipulation, and execution of significant manipulations are made much easier by the structuring conventions.

3c4b Also, in working on line at a CRT console, not only is manipulation made much easier and more powerful by the structure, but a user's ability to get about very quickly within his data, and to have special "views" of it generated to suit his need, are significantly aided by the structure.

3c4c We have come to write all of our documentation, notes, reports, and proposals according to these conventions, because of the resulting increase in our ability to study and manipulate them during composition, modification, and usage. Our programming systems also incorporate the conventions. We have found it to be fairly universal that after an initial period of negative reaction in reading explicitly structured material, one comes to prefer it to material printed in the normal form.

3d File Studying

3d1 The computer aids are used for two principal "studying" operations, both concerned with construction of the user's "views," i.e., the portion of his working text that he sees on the screen at a given moment.

3d1a Display Start

3d1a1 The first operation is finding a particular statement in the file (called the "display start"); the view will then begin with that statement. This is equivalent to finding the beginning of a particular passage in a hard-copy document.

3d1b Form of View

3d1b1 The second operation is the specification of a "form" of view—it may simply consist of a screenful of text which sequentially follows the point specified as the display start, or it may be constructed in other ways, frequently so as to give the effect of an outline.

3d1c In normal, off-line document studying, one often does the first type of operation, but the second is like a scissors-and-

staple job and is rarely done just to aid one's studying.

3d1d (A third type of service operation that will undoubtedly be of significant aid to studying is question answering. We do not have this type of service.)

3d2 Specification of Display Start

3d2a The display start may be specified in several ways:

3d2a1 By direct selection of a statement which is on the display—the user simply points to any character in the statement, using the mouse.

3d2a2 If the desired display start is not on the display, it may be selected indirectly if it bears a "marker."

3d2a2a Markers are normally invisible. A marker has a name of up to five characters, and is attached to a character of the text. Referring to the marker by name (while holding down a special button) is exactly equivalent to pointing to the character with the mouse.

3d2a2b The control procedures make it extremely quick and easy to fix and call markers.

3d2a3 By furnishing either the name or the location number of the statement, which can be done in either of two basic ways:

3d2a3a Typing from the keyboard

3d2a3b Selecting an occurrence of the name or number in the text. This may be done either directly or via an indirect marker selection.

3d2b After identifying a statement by one of the above means, the user may request to be taken directly there for his next view. Alternately, he may request instead that he be taken to some statement bearing a specified structure relationship to the one specifically identified. For instance, when the user identifies Statement 3E4 by one of the above means (assume it to be a member of the list 3E1 through 3E7), he may ask to be taken to

3d2b1 Its successor, i.e., Statement 3E5

3d2b2 Its predecessor, i.e., Statement 3E3

3d2b3 Its list tail, i.e., Statement 3E7

3d2b4 Its list head, i.e., Statement 3E1

3d2b5 Its list source, i.e., Statement 3E

3d2b6 Its subhead, i.e., Statement 3E4A

3d2c Besides being taken to an explicitly identified statement, a user may ask to go to the first statement in the file (or the next after the current location) that contains a specified word or string of characters.

3d2c1 He may specify the search string by typing it in, by direct (mouse) selection, or by indirect (marker) selection.

3d3 Specification of Form of View

3d3a The "normal" view beginning at a given location is like a frame cut out from a long scroll upon which the hierarchical set of statements is printed in sequential order. Such a view is displayed in Figure 1.

3d3b Otherwise, three independently variable view-specification conditions may be applied to the construction of the displayed view: level clipping, line truncation, and content filtering. The view is simultaneously affected by all three of these.

3d3b1 Level: Given a specified level parameter, L ($L = 1, 2, \dots, ALL$), the view generator will display only those statements whose "depth" is less than or equal to L. (For example, Statement 3E4 is third level, 3E second, 4B2C1 fifth, etc.) Thus it is possible to see only first-level statements, or only first-, second-, and third level statements, for example.

3d3b2 Truncation: Given a specified truncation parameter, T ($T = 1, 2, \dots, ALL$), the view generator will show only the first T lines of each statement being displayed.

3d3b3 Content: Given a specification for desired content (written in a special high-level content-analysis language) the view generator optionally can be directed

to display only those statements that have the specified content.

3d3b3a One can specify simple strings, or logical combinations thereof, or such things as having the word "memory" within four words of the word "allocation."

3d3b3b Content specifications are written as text, anywhere in the file. Thus the full power of the system may be used for composing and modifying them.

3d3b3c Any one content specification can then be chosen for application (by selecting it directly or indirectly). It is compiled immediately to produce a machine-code content-analysis routine, which is then ready to "filter" statements for the view generator.

3d3c In addition, the following format features of the display may be independently varied: indentation of statements according to level, suppression of location numbers and/or names of statements, and separation of statements by blank lines.

3d3d. The user controls these view specifications by means of brief, mnemonic character codes. A skilled user will readjust his view to suit immediate needs very quickly and frequently; for example, he may change level and truncation settings several times in as many seconds.

3d4 "Freezing" Statements

3d4a One may also pre-empt an arbitrary amount of the upper portion of the screen for holding a collection of "frozen" statements. The remaining lower portion is treated as a reduced-size scanning frame, and the view generator follows the same rules for filling it as described above.

3d4b The frozen statements may be independently chosen or dismissed, each may have line truncation independent of the rest, and the order in which they are displayed is arbitrary and readily changed. Any screen-select operand for any command may be selected from any portion of the display (including the frozen statements).

3d5 Examples

3d5a Figures 3 and 4 show views generated from the same starting point with different level-clipping parameters. This example happens to be of a program written in our Machine-Oriented language (MOL, see below).

3d5b Figure 5, demonstrates the freezing feature with a view of a program (the same one shown in Figure 8) written in our Control Metalanguage (CML, see below). Statements 3C, 3C2, 2B, 2B1, 2B2, 2B3, and 2B4 are frozen, and statements from 2J on are shown normally with L = 3, T = 1.

3d5b1 The freezing here was used to hold for simultaneous view four different functionally related process descriptions. The subroutines (+ BUG1SPEC) and

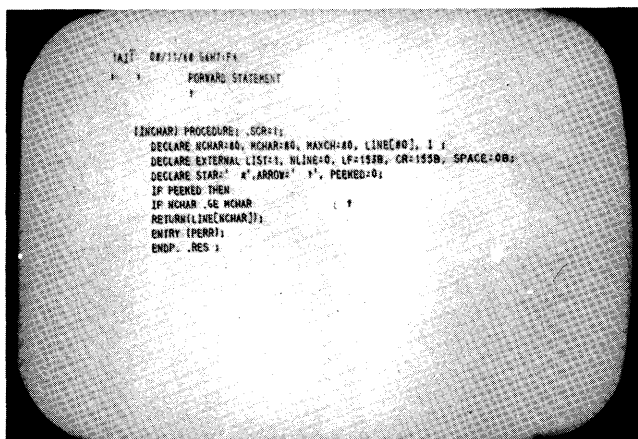


FIGURE 3—View of an MOL program, with level parameter set to 3 and truncation to 1

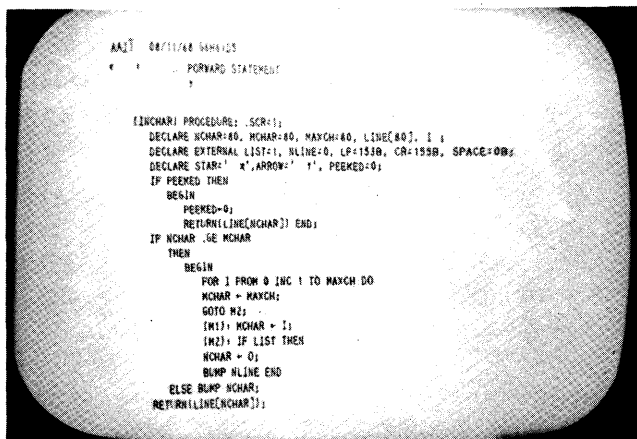


FIGURE 4—Same program as Figure 3, but with level parameter changed to 6 (several levels still remain hidden from view)

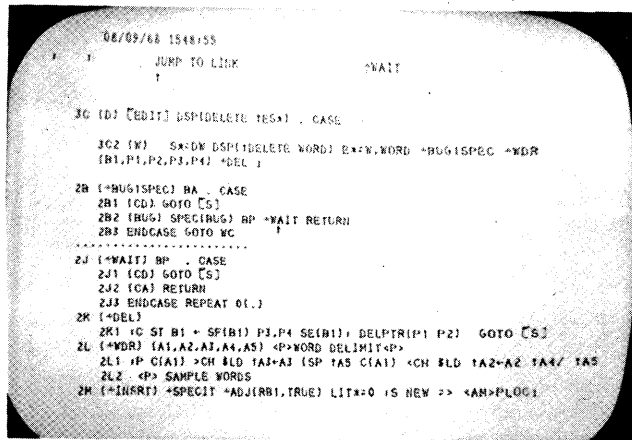


FIGURE 5—View of CML program, showing six frozen statements and illustrating use of reference hopping

(+ WAIT were located by use of the hop-to-name feature described above.

3e File Modification

3e1 Here we use a standard set of editing operations, specifying with each operation a particular type of text entity.

3e1a Operations: Delete, Insert, Replace, Move, Copy.

3e1b Entities (within text of statements): Character, Text (arbitrary strings), Word, Visible (print string), Invisible (gap string).

3e1c Entities (for structure manipulation): Statement, Branch (statement plus all substructure), Group (sublist of branches), Plex (complete list of branches).

3e2 Structure may also be modified by joining statements, or breaking a statement into two at a specified point.

3e3 Generally, an operation and an entity make up a command, such as "Delete Word." To specify the command, the user types the first letter of each word in the command: thus "DW" specifies "Delete Word." There are occasional cases where a third word is used or where the first letter cannot be used because of ambiguities.

3f File Output

3f1 Files may be sent to any of a number of different output devices to produce hard copy—an upper/lower-case line printer, an

on-line high-quality typewriter, or paper tape to drive various typewriters.

3f1a In future it will be possible to send files via magnetic tape to an off-line CRT-to-film system from which we can produce Xerox prints, Multilith masters, or microform records.

3f2 Flexible format control may be exercised in this process by means of specially coded directives embedded in the files—running headers, page numbering, line lengths, line centering, suppression of location numbers, indenting, right justification (hyphenless), etc., are controllable features.

3g Compiling and Debugging

3g1 Source-code files written in any of our compiler languages (see below), or in the SDS 940 assembly language (ARPAS, in which our compiler output is produced) may be compiled under on-line control. For debugging, we have made a trivial addition to the SDS 940's DDT loader-debugger so as to operate it from the CRT displays. Though it was designed to operate from a Teletype terminal, this system gains a great deal in speed and power by merely showing with a display the last 26 lines of what would have been on the Teletype output.

3h Calculating

3h1 The same small innovation as mentioned above for DDT enables us to use the CAL system from a display terminal.

3i Conferencing

3i1 We have set up a room specially equipped for on-line conferencing. Six displays are arranged in the center of a square table (see Figure 6) so that each of twenty participants has good visibility. One participant controls the system, and all displays show the same view. The other participants have mice that control a large arrow on the screen, for use as a pointer (with no control function).

3i2 As a quick means of finding and displaying (with appropriate forms of view) any desired material from a very large collection, this system is a powerful aid to presentation and review conferences.



FIGURE 6—On-line conference arrangement

3i3 We are also experimenting with it in project meetings, using it not only to keep track of agenda items and changes but also to log progress notes, action notes, etc. The review aid is of course highly useful here also.

3i4 We are anxious to see what special conventions and procedures will evolve to allow us to harness a number of independent consoles within a conference group. This obviously has considerable potential.

4 SERVICE-SYSTEM SOFTWARE

4a The User's Control Language

4a1 Consider the service a user gets from the computer to be in the form of discrete operations—i. e., the execution of individual “service functions” from a repertoire comprising a “service system.”

4a1a Examples of service functions are deleting a word, replacing a character, hopping to a name, etc.

4a2 Associated with each function of this repertoire is a “control-dialogue procedure.” This procedure involves selecting a service function from the repertoire, setting up the necessary parameter designations for a particular application, recovering from user errors, and calling for the execution of the function.

4a2a The procedure is made up of the sequence of keystrokes, select actions, etc.

made by the user, together with the interspersed feedback messages from the computer.

4a3 The repertoire of service functions, together with their control-dialogue procedures, constitutes the user's "control language." This is a language for a "master-slave" dialogue, enabling the user to control application of the computer's capabilities to his own service.

4a3a It seems clear that significant augmentation of one's intellectual effectiveness from the harnessing of computer services will require development of a broad and sophisticated control-language vocabulary.

4a3b It follows that the evolution of such a control language is a very important part of augmentation-system research.

4a4 For the designer of user systems, it is important to have good means for specifying the nature of the functions and their respective control-dialogue procedures, so that a design specification will be

4a4a Concise, so that its essential features are easily seen

4a4b Unambiguous, so that questions about the design may be answered clearly

4a4c Canonical, so that information is easily located

4a4d Natural, so that the form of the description fits the conceptual frame of the design

4a4e Easy to compose, study, and modify, so that the process of evolutionary design can be facilitated.

4a5 It is also important for the user to have a description of the service functions and their control-dialogue procedures.

4a5a The description must again be concise, unambiguous, canonical, and natural; furthermore, it must be accurate, in that everything relevant to the user about the service functions and their control-dialogue procedures is described, and everything described actually works as indicated.

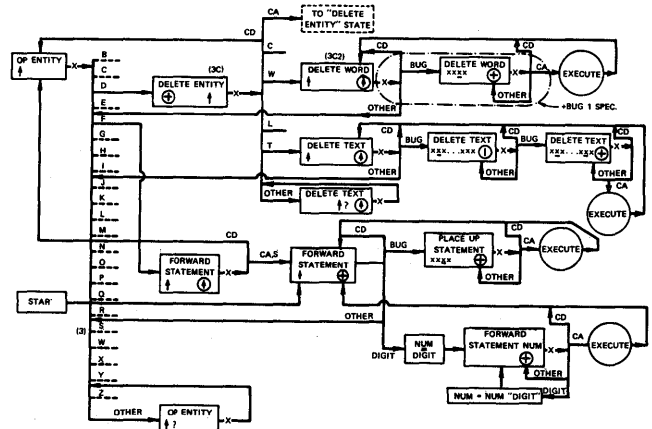


FIGURE 7—State-chart portrayal of part of the text-manipulation control language

4b State-Chart Representation of Control-Language Design

4b1 Figure 7 shows a charting method that was used in earlier stages of our work for designing and specifying the control-procedure portions of the control language. Even though limited to describing only the control-dialogue procedures, this representation nonetheless served very well and led us to develop the successive techniques described below.

4b2 Figure 7 shows actual control procedures for four service functions from the repertoire of an interactive system: Delete Word, Delete Text, Place Up Statement, and Forward Statement.

4b2a The boxes contain abbreviated descriptions of relevant display-feedback conditions, representing the intermediate states between successive user actions. Both to illustrate how the charting conventions are used and to give some feeling for the dynamics of our user-system control procedures, we describe briefly below both the chart symbols and the associated display-feedback conventions that we have developed.

4b2a1 The writing at the top of each box indicates what is to be shown as "command feedback" at the top of the display (see Figures 3, 4 and 5).

4b2a1a An uparrow sometimes ap-

pears under the first character of one of the words of Command Feedback.

4b2a1a1 This indicates to the user that the next character he types will be interpreted as designating a new term to replace that being pointed to—no uparrow under Command Feedback signifies that keyboard action will not affect the command designation.

4b2a1b "Entity" represents the entity word (i.e., "character," "word," "statement," etc.) that was last used as part of a fully specified command.

4b2a1b1 The computer often "offers" the user an entity option.

4b2a2 The circle in the box indicates the character to be used for the "bug" (the tracking spot), which alternates between the characters uparrow and plus.

4b2a2a The uparrow indicates that a select action is appropriate, and the plus indicates that a select action is inappropriate.

4b2a3 The string of X's, with underlines, indicates that the selected characters are to be underlined as a means of showing the user what the computer thinks he has selected.

4b2b There is frequently an X on the output line from a box on the chart. This indicates that the computer is to wait until the user has made another action.

4b2b1 After this next action, the computer follows a branching path, depending upon what the action was (as indicated on the chart) to reach another state-description box or one of the function-execution processes.

4c The Control Metalanguage

4c1 In search for an improvement over the state chart, we looked for the following special features, as well as the general features listed above:

4c1a A representational form using structural text so as to harness the power of our on-line text-manipulation techniques

for composing, studying, and modifying our designs.

4c1b A form that would allow us to specify the service functions as well as the control-dialogue procedures.

4c1c A form such that a design-description file could be translated by a computer program into the actual implementation of the control language.

4c2 Using our Tree Meta compiler-compiler (described below), we have developed a next step forward in our means of designing, specifying, implementing and documenting our on-line control languages. The result is called "Control Metalanguage" (CML).

4c2a Figure 8 shows a portion of the description for the current control language, written in Control Metalanguage.

4c2a1 This language is the means for describing both the service functions and their control-dialogue procedures.

4c2b The Control Metalanguage Translator (CMLT) can process a file containing such a description, to produce a corresponding version of an interactive system which responds to user actions exactly as described in the file.

4c3 There is a strong correspondence between the conventions for representing the control procedures in Control Metalanguage and in the state chart, as a comparison of Figures 8 and 7 will reveal.

4c3a The particular example printed out for Figure 8 was chosen because it specifies some of the same procedures as in Figure 7.

4c3b For instance, the steps of display-feedback states, leading to execution of the "Delete Word" function, can readily be followed in the state chart.

4c3b1 The steps are produced by the user typing "D," then "W," then selecting a character in a given word, and then hitting "command accept" (the CA key).

4c3b2 The corresponding steps are outlined below for the Control Metalanguage description of Figure 8, progressing from Statement 3, to Statement 3c, to

Statement 3c2, to Subroutine + BUG-SPEC, etc.

4c3b3 The points or regions in Figure 7 corresponding to these statements and subroutines are marked by (3), (3C), (3C2), and (+ BUG1SPEC), to help compare the two representations.

4c3c These same steps are indicated in Figure 8, starting from Statement 3:

4c3c1 "D" sets up the state described in Statement 3C

4c3c2 "W" sets up the state described in Statement 3C2

FIGURE 8—Metalanguage description of part of control language

3 (wc:) zap case

3A (b) [edit] dsp(backward ↑es*) . case

.
.
.

3B (c) [edit] dsp(copy ↑es*) :s true => <am>adj1: . case

3B1 (c) s*=cc dsp(↑copy character) e*=c,character +bug2spec
+cdlim(b1,p1,p2,p3,p4) +cdlim(b2,p5,p6,p7,p8)
+cpctx(b1,p2,p4,p5,p6) ;

3B2 (w) s*=cw dsp(↑copy word) e*=w,word +bug2spec
+wdr2(b1,p1,p2,p3,p4) +wdr2(b2,p5,p6,p7,p8)
+cpwdvs(b1,p2,p4,p5,p6) ;

3B3 (l) s*=cl dsp(↑copy line) e*=l,line +bug2spec
+ldlim(b1,p1,p2,p3,p4) +ldlim(b2,p5,p6,p7,p8) :c st b1+sf(b1) p2,
rif :p p2>p1 cr: then (cr) else (null) , p5 p6, p4 se(b1): goto
[s]

3B4 (v) s*=cv dsp(↑copy visible) e*=v,visible +bug2spec
+vdr2(b1,p1,p2,p3,p4) +vdr2(b2,p5,p6,p7,p8)
+cpwdvs(b1,p2,p4,p5,p6) ;

.
.

3b10 endcase +caqm ;

3C (d) [edit] dsp(delete ↑es*) . case

3C1 (c) s*=dc dsp(↑delete character) e*=c,character +bug1spec
+cdlim(b1,p1,p2,p3,p4) +del;

3C2 (w) s*=dw dsp(↑delete word) e*=w,word +bug1spec +wdr
(b1,p1,p2,p3,p4) +del ;

3C3 (l) s*=dl dsp(↑delete line) e*=l,line +bug1spec...

.
.
.

4c3c3 The subroutine +BUG1SPEC waits for the select-word (1) and CA (2) actions leading to the execution of the delete-word function.

4c3c3a Then the TWDR subroutine takes the bug-position parameter and sets pointers P1 through P4 to delimit the word in the text data.

4c3c3b Finally, the + DEL subroutine deletes what the pointers delimit, and then returns to the last-defined state (i.e., to where $S^* = DW$).

4d Basic Organization of the On-Line System (NLS)

4d1 Figure 9 shows the relationships among the major components of NLS.

4d2 The Tree Meta Translator is a processor specially designed to produce new translators.

4d2a There is a special language—the Tree Meta Language—for use in describing the translator to be produced.

4d2b A special Tree Meta library of subroutines must be used, along with the output of the Tree Meta Translator, to produce a functioning new translator. The same library serves for every translator it produces.

4d3 For programming the various subroutines used in our 940 systems, we have developed a special Machine-Oriented Language (MOL), together with an MOL Translator to convert MOL program descriptions into machine code (see Ref. Hay1 for a complete description).

4d3a The MOL is designed to facilitate system programming, by providing a high-level language for iterative, conditional, and arithmetic operations, etc., along with a block structure and conventions for labeling that fit our structured-statement on-line manipulation aids.

4d3a1 These permit sophisticated computer aid where suitable, and also allow the programmer to switch to machine-level coding (with full access to variables, labels, etc.) where core space, speed, timing, core-mapping arrangements, etc., are critical.

4d4 The NLS is organized as follows (letters refer to Figure 9):

4d4a The Control Processor (E) receives and processes successive user actions, and calls upon subroutines in the library (H) to provide it such services as the following:

4d4a1 Putting display feedback on the screen

4d4a2 Locating certain data in the file

4d4a3 Manipulating certain working data

4d4a4 Constructing a display view of specified data according to given viewing parameters, etc.

4d4b The NLS library subroutines (H) are produced from MOL programs (F), as translated by the MOL Translator (G).

4d4c The Control Processor is produced from the control-language description (D), written in Control Metalanguage, as translated by the CMLT (C).

4d4d The CMLT, in turn, is produced from a description (A) written in Tree Meta, as translated by the Tree Meta Translator (B).

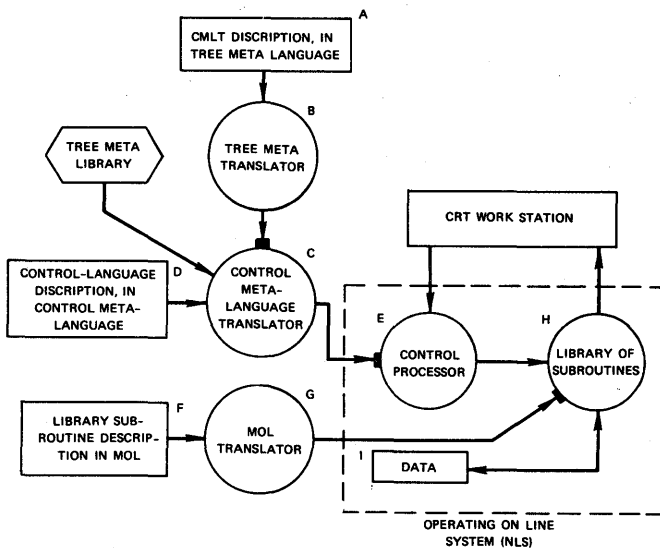


FIGURE 9—Basic organization of NLS showing use of compilers and compiler-compiler to implement it

4d5 Advantages of Metalanguage Approach to NLS Implementation

4d5a The metalanguage approach gives us improved means for control-language specification, in terms of being unambiguous, concise, canonical, natural and easy to compose, study and modify.

4d5b Moreover, the Control Metalanguage specification promises to provide in itself a users' documentation that is completely accurate, and also has the above desirable characteristics to facilitate study and reference.

4d5c Modifying the control-dialogue procedures for existing functions, or making a reasonable range of changes or additions to these functions, can often be accomplished solely by additions or changes to the control-language record (in CML).

4d5c1 With our on-line studying, manipulating and compiling techniques, system additions or changes at this level can be thought out and implemented (and automatically documented) very quickly.

4d5d New functions that require basic operations not available through existing subroutines in the NLS library will need to have new subroutines specified and programmed (in MOL), and then will need new terms in CML to permit these new functions to be called upon. This latter requires a change in the record (A), and a new compilation of CMLT by means of the Tree Meta Translator.

4d5d1 On-line techniques for writing and modifying the MOL source code (F), for executing the compilations, and for debugging the routines, greatly reduce the effort involved in this process.

5 SERVICE-SYSTEM HARDWARE (OTHER THAN SDS 940)

5a In addition to the SDS 940, the facility includes peripheral equipment made by other manufacturers and equipment designed and constructed at SRI.

5b All of the non-SDS equipment is interfaced through the special devices channel which con-

nects to the second memory buss through the SDS memory interface connection (MIC).

5b1 This equipment, together with the RAD, is a significant load on the second memory buss. Not including the proposed "special operations" equipment, the maximum expected data rate is approximately 264,000 words per second or one out of every 2.1 memory cycles. However, with the 940 variable priority scheme for memory access (see Pirtle¹), we expect less than 1 percent degradation in CPU efficiency due to this load.

5b2 This channel and the controllers (with the exception of the disc controller) were designed and constructed at SRI.

5b2a In the design of the hardware serving the work stations, we have attempted to minimize the CPU burden by making the system as automatic as possible in its access to memory and by formatting the data in memory so as to minimize the executive time necessary to process it for the users.

5c Figure 10 is a block diagram of the special-devices channel and associated equipment. The major components are as follows:

5c1 Executive Control

5c1a This is essentially a sophisticated multiplexer that allows independent, asynchronous access to core from any of the 6 controllers connected to it. Its functions are the following:

5c1a1 Decoding instructions from the computer and passing them along as signals to the controllers.

5c1a2 Accepting addresses and requests for memory access (input or output) from the controllers, determining relative priority among the controllers, synchronizing to the computer clock, and passing the requests along to memory via the MIC.

5c1b The executive control includes a comprehensive debugging panel that allows any of the 6 controllers to be operated off-line without interfering with the operation of other controllers.

5c2 Disc File

5c2a This is a Model 4061 Bryant disc, selected for compatibility with the continued 940-system development by Berkeley's Project GENIE, where extensive file-handling software was developed.

5c2b As formatted for our use, the disc will have a storage capacity of approximately 32 million words, with a data-transfer rate of roughly 40,000 words per second and average access time of 85 milliseconds.

5c2c The disc controller was designed by Bryant in close cooperation with SRI and Project GENIE.

5c3 Display System

5c3a The display systems consists of two identical subsystems, each with display con-

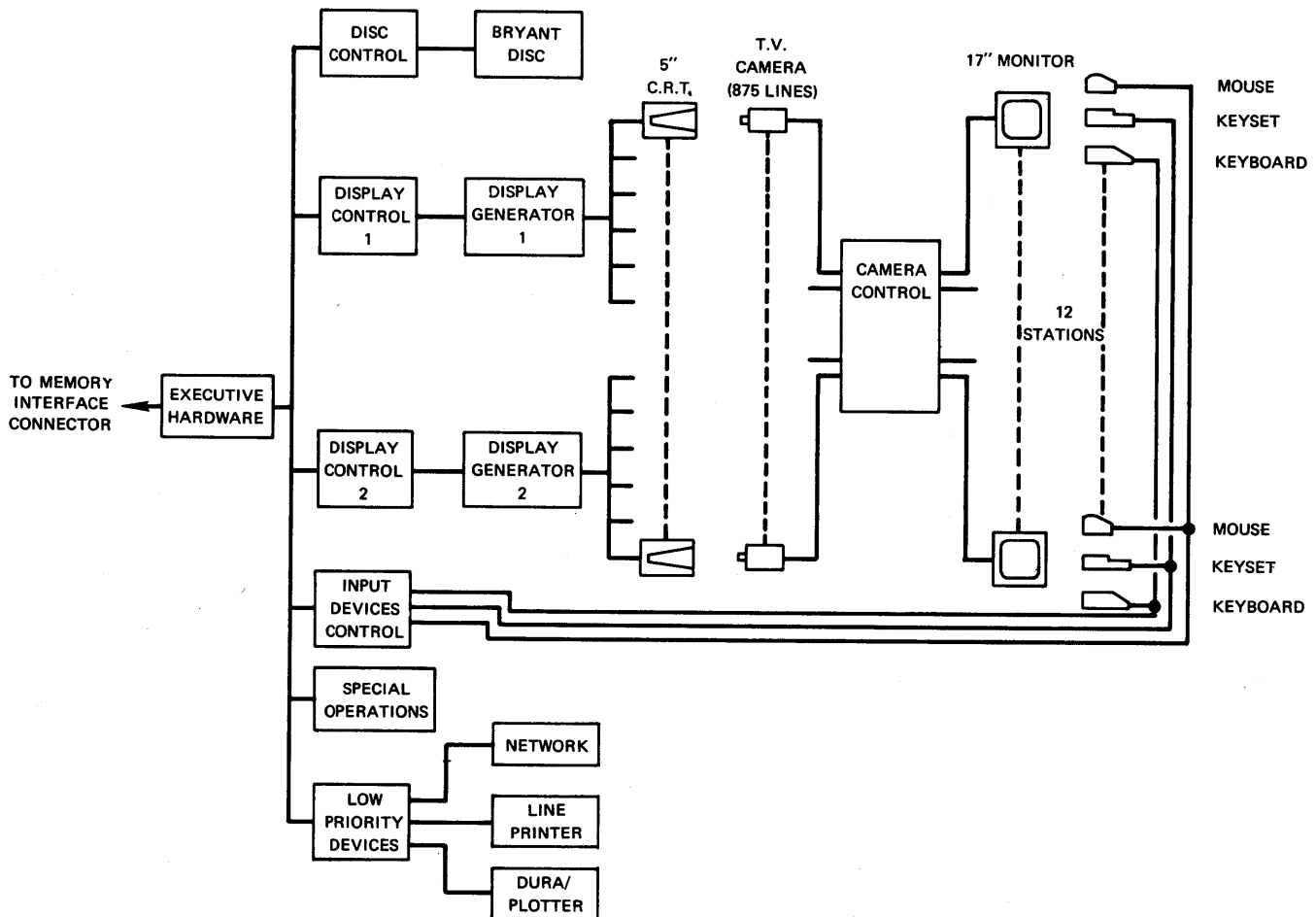
troller, display generator, 6 CRT's, and 6 closed-circuit television systems.

5c3b The display controllers process display-command tables and display lists that are resident in core, and pass along display-buffer contents to the display generators.

5c3c The display generators and CRT's were developed by Tasker Industries to our specifications. Each has general character-vector plotting capability. They will accept display buffers consisting of instructions (beam motion, character writing, etc.) from the controller. Each will drive six 5-inch high-resolution CRT's on which the display pictures are produced.

5c3c1 Character writing time is approximately 8 microseconds, allowing an aver-

FIGURE 10—Special devices channel



age of 1000 characters on each of the six monitors when regenerating at 20 cps.

5c3d A high-resolution (875-line) closed-circuit television system transmits display pictures from each CRT to a television monitor at the corresponding work-station console.

5c3e This system was developed as a "best solution" to our experimental-laboratory needs, but it turned out to have properties which seem valuable for more widespread use:

5c3e1 Since only all-black or all-white signal levels are being treated, the scan-beam current on the cameras can be reduced to achieve a short-term image-storage effect that yields flicker-free TV output even when the display refresh rate is as low as 15 cps. This allows a display generator to sustain about four times more displayed material than if the users were viewing direct-view refreshed tubes.

5c3e2 The total cost of small CRT, TV camera, amplifier-controller, and monitor came to about \$5500 per work station—where a random-deflection, display-quality CRT of similar size would cost considerably more and would be harder to drive remotely.

5c3e3 Another cost feature which is very important in some system environments favors this TV approach: The expensive part is centrally located; each outlying monitor costs only about \$600, so terminals can be set up even where usage will be low, with some video switching in the central establishment to take one terminal down and put another up.

5c3e4 An interesting feature of the video system is that with the flick of a switch the video signal can be inverted, so that the image picked up as bright lines on dim background may be viewed as black lines on a light background. There is a definite user preference for this inverted form of display.

5c3f In addition to the advantages noted above, the television display also invites the use of such commercially available devices as extra cameras, scan converters, video switches, and video mixers to enrich system service.

5c3f1 For example, the video image of a user's computer-generated display could be mixed with the image from a camera focused on a collaborator at another terminal; the two users could communicate through both the computer and a voice intercom. Each user would then see the other's face superimposed on the display of data under discussion.

5c3f2 Superimposed views from cameras focused on film images or drawings, or on the computer hardware, might also be useful.

5c3f3 We have experimented with these techniques (see Figure 11) and found them to be very effective. They promise to add a great deal to the value of remote display terminals.

5c4 Input-Device Controller

5c4a In addition to the television monitor, each work-station console has a keyboard, binary keyset, and mouse.

5c4b The controller reads the state of these

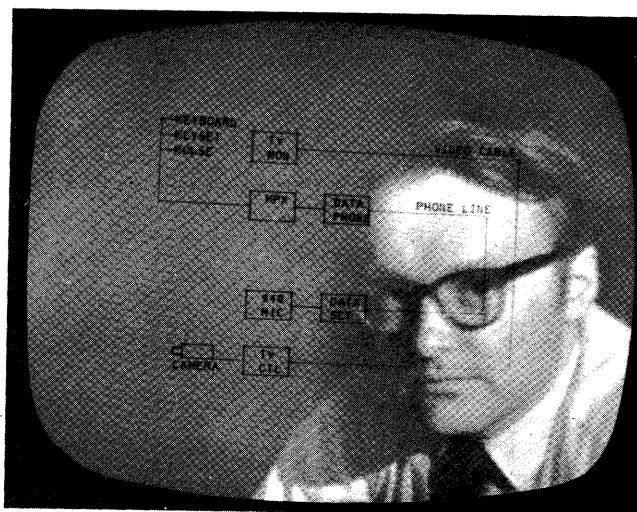


FIGURE 11—Television display obtained by mixing the video signal from a remote camera with that from the computer-generated display

devices at a preset interval (about 30 milliseconds) and writes it into a fixed location table in core.

5c4b1 Bits are added to information from the keyboards, keysets and mouse switches to indicate when a new character has been received or a switch has changed state since the last sample. If there is a new character or switch change, an interrupt is issued after the sample period.

5c4b2 The mouse coordinates are formatted as a beam-positioning instruction to the display generator. Provisions are made in the display controller for including an entry in the mouse-position table as a display buffer. This allows the mouse position to be continuously displayed without any attention from the CPU.

5c5 Special Operations

5c5a The box with this label in Figure 10 is at this time only a provision in the executive control for the addition of a high-speed device. We have tentative plans for adding special hardware here to provide operations not available in the 940 instruction set, such as character-string moves and string-pattern matching.

5c6 Low-Priority Devices

5c6a This controller accommodates three devices with relatively low data-transfer

rates. At this time only the line printer is implemented, with provisions for adding an on-line typewriter (Dura), a plotter, and a terminal for the proposed ARPA computer network.

5c6a1 The line printer is a Potter Model HSP-3502 chain printer with 96 printing characters and a speed of about 230 lines per minute.

6 REFERENCES

- 6a* (English 1) W K ENGLISH D C ENGELBART
B HUDDART
Computer-aided display control
Final Report Contract NAS 1-3988 SRI Project 5061 Stanford Research Institute Menlo Park California July 1965
- 6b* (English2) W K ENGLISH D C ENGELBART M L BERMAN
Display-selection techniques for text manipulation
IEEE Trans on Human Factors in Electronics Vol HFE-8 No 1 1967
- 6c* (Engelbart1) D C ENGELBART
Augmenting human intellect: A conceptual framework
Summary Report Contract AF 49 638 1024 SRI Project 3578 Stanford Research Institute Menlo Park California October 1962
- 6d* (Engelbart2) D C ENGELBART
A conceptual framework for the augmentation of man's intellect
In Vistas in Information Handling Vol 1 D W Howerton and D C Weeks eds Spartan Books Washington D C 1963
- 6e* (Hay1) R E HAY J F RULIFSON
MOL940 Preliminary specifications for an ALGOL like machine-oriented language for the SDS 940
Interim Technical Report Contract NAS 1-5940 SRI Project 5890 Stanford Research Institute Menlo Park California March 1968
- 6f* (Pirtle1) M PIRTLE
Intercommunication of Processors and memory
Proc Fall Joint Computer Conference Anaheim California November 1967

An approach to simulation model development for improved planning*

by JAMES L. MCKENNEY

Harvard University
Cambridge, Massachusetts

INTRODUCTION

This paper proposes an approach to planning which relies on involving the manager in the development of a simulation model. The purpose of this involvement is to improve the manager's understanding of his environment; and therefore, the appropriateness of his plans. The approach is based upon the premise that active planning is a learning process. This premise is the result of experience in several modeling projects and a clinical study of a manager developing a simulation model.¹

Active planning involves the manager in a continuous search and analysis of the significant causal factors influencing the future of his business. This search process is normally conducted by creating comprehensive plans which commit resources of the business for three to five years. These plans are then revised and extended as the environment changes, in accordance with some periodic schedule. The function of the simulation model in the planning process per se is to serve as an experimental device to allow the manager to evaluate alternative plans and in the process consider different concepts of his business. By concept we mean an articulation of the significant factors influencing the profitability of a firm and how they seem to relate to the firm. A simulation

model is an explicit representation of how these influences could affect the firm. As such it provides a manipulatable structure with which to represent alternative actions the firm could take and define the economic outcome of such responses. We suggest the model can also serve as a stimulus to create a broader range of testable concepts which improve as to their pertinence in explaining and identifying forces influencing the future of the firm through time, thus, a stimulus to learning. It is for this reason that simulation model development is a unique and powerful approach to planning.

Prior to the present era, planning structures or models were often rigorous economic models concerned with long-term predictability rather than specific opportunities. In addition, the model was typically characterized in symbolic terms foreign to the experience of the manager and difficult to relate directly to decision problems. He typically dealt with the model and the model builder as most other staff activities within a business by gauging their usefulness in resolving planning issues. If the staff man was persuasive, the manager over time would rely upon the modeler and his model to help him commit future resources. It would seem, we are approaching a stage of conceptual expertise wherein computer programs can be flexible enough to adapt to a manager's concepts of his business, and of equal importance a growing number of modelers have the capacity to operationally represent managerial concepts.

An operational model can serve as a dynamic set of hypotheses with which to develop a consistent codified set of concepts about a business. The model allows prompt evaluation with data or judgment and then easy modification of the hypotheses for further testing. The richness of the

*This study was supported in part by funds made available by the Ford Foundation to the Graduate School of Business, Stanford University. The conclusions, opinions, and other statements contained in this publication are, of course, those of the author. This working paper should not be quoted or reproduced in whole or in part without the written consent of the author. Comments are solicited and should be addressed directly to the author.

vocabulary available to the modeler and the broad spectrum of alternative methods of representing data, allow a full range of tools to be utilized in distilling the essence of the manager's concepts. Further, they can be programmed to generate output statements pertinent to decision problems. In addition, programs allow detailed documentation including flow-charts so that they can be understandable to the manager. We propose this opportunity to explicitly define and test business concepts which can be a powerful process to improve planning if the manager becomes involved in developing the model. In brief, we see a great advantage in bringing the power of the scientific method to the manager directly, by allowing him to assume an active role in the design and evolution of simulation models. An appropriate modeling project can combine the insight of the manager with the analytic capacity of the scientist.

Guidelines for model development

The following guidelines outline one approach to model development which has been successful in obtaining the active involvement of the manager in a project. The guidelines are as follows:

1. Simulation model development should be conducted as a project to aid the planning process.
2. An important characteristic of the project is the evolution of goals, uses and specification of the model, as it relates to the planning process.
3. The manager's intuition is typically the operational reference of the pertinent environment; therefore, one role of the model is to improve the consistency of the manager's intuition and to make him aware of new information requirements.
4. A prime function of the model is to amplify the intuition of the manager generating a spectrum of analyses for a range of codifiable conditions.
5. The entire project should be considered in part a learning experience both for modelers to communicate and define adaptable concepts and for the manager to consider how explicit statements and concise definition can improve his understanding of the world.

The sole criterion of the success of a simulation model for planning is the utilization of the model to allocate resources. An unused model, no mat-

ter how elegant, is a failure. However, it may be that the most significant use a manager makes of a simulation model is during its development, to refine his own concept of his business. The discussion below deals with how these guidelines serve to encourage the manager to contribute to the development of a model and what the modeler should have in mind to facilitate this contribution.

Using the guidelines to develop a planning model

How these guidelines might aid the development of a simulation model is presented in the following synopsis of a simulation project in an industrial firm. The firm, a consumer goods producer, with sales in excess of \$200 million was planning to enter the European market. This was its first venture overseas and the executives felt a need for improved decision-making procedures to cope with the unknown seemingly more complex situation. The staff aids to the executive committee had suggested that a simulation model might assist them in allocating capital overseas to assure an orderly and profitable entry into the new market.

A series of seminars conducted by the staff with outside consultants was initiated to explore methods of planning in general and the potential of simulation models in particular. A topic of one of the seminars was the evolutionary nature of simulation models with the expectant change in understanding of their environment by the individuals using the model. This session produced a lively interest in developing a model tailored to the needs of the executive committee. The eventual result was a tentative five-year program for the improvement of the firm's strategic planning. A three-year capital budget of approximately \$70,000 per year was allocated to a project for the development of a simulation model. Progress review periods were to be held every six months by the executive vice president of sales who was responsible for long-range planning. All vice presidents of the firm and members of the executive committee were to be the active managers of the project and they agreed to spend up to four hours per week to develop improved planning procedures.

The initial proposal developed in the seminar called for a tentative model to be operational in one calendar year at which time a redefinition of project goals and model specifications would take place. The initial planning problem the model

was to aid was the allocation of capital resources in order that the company could most effectively enter and become established in the European market.

The initial model, as specified by the executive committee, called for a representation of the necessary resources measured in required dollars for specific future calendar months. The model would simulate the production of goods in European countries to meet simulated demand for the company's products identified by price and type of product. It was the responsibility of the managers to identify the sequence in which products would be introduced, probable competitive actions, estimates of total market growth and available capital. The model was expected to derive a profit figure for the defined conditions and allow conditions to be easily changed to observe the impact of alternatives.

The modelers began the project by attempting to define what factors the managers considered in their planning decisions and which of these seemed best to include on the model. To accomplish this definition a series of meetings with each manager was held to identify the critical elements which influence his operation. Further, how these influences might be formulated in the simulation model. For example, the marketing vice president thought total market, market share, price level of products, and pattern of growth were the basic factors in describing a market. Research was then started on how one could create a statement that would describe the impact of these factors in the European market. A series of definition papers were developed for each country on how these factors would be combined to describe the total European market for the next ten years. In addition, the papers identified what data would be required and what reports would be generated from the model. A series of memoranda on the production, distribution and financial systems were generated to serve as a basis for seminars with all modelers and managers to discuss what should be included in the simulation model to generate more appropriate plans. After each seminar small groups would often discuss a specific aspect such as how a new product should be represented in the production process or how the time lags in the distribution system could be measured and made dynamic. The seminars were followed by additional two- to three-hour individual manager conferences with two or three members of the modeling group to insure the

manager's ideas were accurately represented in the model and to explain how the model was functioning with other managers' definitions.

A continual effort was made by the modelers to define all terms as clearly as possible for inclusion in a glossary that all participants received. The glossary established a common understanding on words all too often not well-defined such as: assumptions, sensitivity, programming, and planning horizon. The glossary also identified all time lags assumed in the model such as two weeks delay between order receipt and delivery. The glossary aided in educating the manager in a bit of modeling jargon and preventing the modelers from using terms without defining them. It was invaluable in documentation of the model.

Concurrent with the manager conferences, data were collected to define the specific form of the relations. Thus, total market data were obtained for each country in Europe and various schemes tested to resolve how to represent growth utilizing historical data. This required the modelers to work with the staff assistants of the managers in an analysis of present measures of what the managers felt to be important. The available accounting data often did not prove sufficient and, therefore, new information had to be created and stated in accessible format. For example, data were collected and distilled to develop a productive capacity model which related the total cost and elapsed time of producing a given quantity of product to the mix of products and the level of production. The elapsed time required to acquire additional productive capacity or change product mix was defined in accordance with how the manufacturing vice president thought capacity responded.

It was not until after six months of discussion during which time data were being collected to formulate the managers' concept into a model that the programming of the model for computer manipulation was started. Simultaneous with the programming effort, a second series of meetings were held with the managers on how they might utilize the simulation in their on-going planning procedures. It was felt important to maintain the managers' interest in model development as it was conjectured that during the programming process several revisions in the managers' model would be necessary. The individual meetings soon became formalized into bi-monthly planning meetings to discuss the state of the model and how it might be used to evaluate alternative resource

allocations in the European market. One question of burning interest was the capital requirements for establishing a plant in Belgium or Germany including cash flow characteristics. This initiated a project on rules for funds flow and how to represent legal delays. These discussions aided the modelers in defining appropriate time units, ranges of accuracy, specific output requirements and potential changes in the input variables. They served to keep the managers informed on the state of the model and its limitations.

As the model entered the final debugging stage the meetings focused more onto methods of testing the model for validity and formulating plans for evaluation by the model. It was decided to run the model one country at a time as each one seemed unique. In these later meetings the managers began to develop expertise in explicitly defining a feasible range of circumstances which could be tested on the model. A test set of data was developed representing the past two years and a set of parameters for generating the next eight years for Italy. This, in turn, caused the modelers to improve the model's ability to accurately represent a set of conditions. The results of this iterative process was an awareness of the importance of experimental design and new insight to the evolutionary aspect of the simulation project. Most individuals were convinced it was a rewarding experience.

The strong commitment was fortunate as early simulation runs proved to generate quantities of useless output. The first simulations were intended to represent the ten years of sales experience in the Italian market. The simulations on the average produced bizarre sales and production demands after the first or second simulated year. The one bright side was that the cash flows resulting from the sales were consistent with past experience.

The managers were not dismayed and suggested procedures which the modeler could incorporate which would aid in the understanding of simulation results. Typical error prevention procedures called for the managers to estimate for the next two, five, and eight years feasible product price ranges, and estimates of production capacity, given the present base of the company. These estimates served as minimum and maximum limits on capacity and sales. The model operated within these bounds to evaluate the proposed price struc-

ture, time of product introduction and other aspects of their plan. They then considered the output of the simulation in terms of these limits. If the output indicated the simulation results hit an upper limit and remained there, the planners discounted the answer, because of model deficiencies but would judge that the plan might be a better one than a plan which drove the model to the lower limits. These procedures have afforded a basis for jointly testing plans and their assumptions while evaluating the sensitivity of the simulation model to a variety of inputs in order to investigate the model's validity.

The simulation project today

There has been an obvious growth in the attitude of the modelers and managers as to what should be in the model and what should be excluded. A few of the original factors included as determinants of demand have been tested and found unimportant. But of more interest is the number of new factors that seem to be of a more basic and casual nature than the original factors. Originally, population had been considered as a basic determinant of demand; now age distribution, wealth distribution, geographical distribution, and other factors of the economy in a given country are being considered as determinants of market potential. Continual evaluation of factors in the model including the definition of assumptions and defense or explanation of these assumptions is now accepted by modelers and managers alike. Finally, measures specified at the start have been superseded by new ones. Specific dollar requirements and time specifications originally desired as outputs have been replaced by requirements of rate of market penetration or equity growth and likely range of profits. In general, most measures of performance are more sophisticated than when the project began.

The managers seem to be evaluating alternative plans with the model to support their intuition. They suggest that the model has improved their judgment by testing some variables which heretofore were thought very important and found wanting as indicators of future influential environmental forces. The model development in part has forced the managers to define their time assumptions explicitly and to codify cost assumptions to accommodate manipulation. This has resulted in an expansion of the accounting system

to allow an evaluation of future plans rather than only a reporting of the accumulated costs of past activities. For example, costs are recorded by product in accordance with length of time since introduction. This change has improved the firm's planning procedures and given a better data base for developing an improved model.

At present the model can almost be considered a professional goal for the management of the company as they are committed to its future development. They do not rely upon it for specific decisions, but seem to feel it a useful tool for improving their planning procedures. Perhaps at some future date they will rely upon it as a partner in decision making as well as process improvement.

Discussion of simulation model projects

A conclusion of this experience and other reported simulation developments would suggest that constituting the project as a research and development venture on the managing process provides a useful orientation.³ This orientation is important as it not only develops a useful model but it can engender an open attitude. This learning attitude is helpful when exploring how one can formulate heretofore nonexplicit relations. In addition, a development project by nature should commit a management to a sizeable budget over an extended period of time. The results of this expenditure are uncertain and, therefore, the project should regularly be appraised as to its effectiveness. This appraisal process is especially important in regard to simulations intended to aid the planning process.

The definition of criteria to evaluate improvement of the planning process is a difficult art and requires experimentation and attention. However, focusing on this aspect of the model's impact provides an appropriate perspective for considering the effectiveness of the model. The appraisal should allow adequate elapsed time for the development of a series of plans concurrent with the implementation of the model. During this time the defense for continuing financial support for the model probably rests on the degree to which it stimulates the management to consider their planning process. After the model is being utilized as an active aid, support should be judged on documental evidence produced by the managers involved. The model should not be judged solely

on appropriateness of results, number of plans evaluated, or mechanics of operation. These can be modified by utilizing different resources to develop the model. It should be judged on how effective it is in improving the planning process.

The prime reason for a preordained extended life of a planning model project is that the only constant characteristic of a simulation model is change. The product of this evolutionary process is assisted if the changing nature of the model is understood by all associated with the model from the very start of the project. Models with a tradition of change will encourage the managers to attempt to define hazy ideas and to experiment with formulating relationships, as conjectures can be changed if desired. It will also induce the modelers to design their procedures to accommodate changing definitions and specifications.

A tactic for developing useful simulation models which are adaptable is to start by working with the responsible managers to model the aspects of the business they feel important. A manager with significant budgetary responsibility is assumed to have had an adequate involvement with his environment to have developed an understanding of what elements are critical to the success of his operation. His measures of these elements often range from precise dollar figures to vague intuitive impressions, but all are important and real to him. It seems reasonable to accept the planner's notion of his business as fact and to attempt to substantiate his concept by programming a model to imitate the concept.

Normally, it is impossible to explicitly define all the factors a planner considers. In addition, individual managers will not be consistent with each other or emphasize the same aspects of the environment. To cope with these concomitant ambiguities, the model should be programmed to codify as many factors as possible with freedom for the manager to modify the impact and range of each variable. Where variables cannot be defined, provision should be made for direct planner influence as he sees fit. In the case above, the manufacturing vice president defined his estimates of production rate during start up at three-month intervals on his appraisal of product mix and volume in the simulated factory. This worked much better than any modeled relation attempted.

The goal of the modeler is to generate an abstraction that adapts comfortably to how the planner considers his resources allocation prob-

lem. The process of programming the well-defined variables should involve the planners and modeler in order that both:

- . Evaluate the sensitivity of the environment to change in the selected variables.
- . Discover new methods of combining variables.
- . Resolve inconsistencies or ambiguities between planners and the environment.

This latter process often serves as a basis for data collection to define missing relations or to test in the environment the validity of assumed relationships. A clear definition of how the simulation function is essential for the mutual consideration of relationships that govern the operation of the simulation.

CONCLUSIONS

An overt goal of most simulation projects for planning is that operationally the simulation model is to serve as an analytical tool for the manager. To serve effectively it must be formulated to produce results which are compatible with the planning procedures. The modeler and manager should continuously appraise what is more economical and effective for the model to accomplish versus the planner. At this point in time it does not seem economically feasible to model completely an environment as effectively as a good human decision maker. However, a simulation model can perform quickly and accurately a long involved sequence of well specified events to produce an answer in predefined terms. How these answers will be used is important in the development of the model and should be considered at each step of the program. The goal of the model developer is to develop a model which can amplify the manager's insight to a resource

allocation problem. At present the method of amplification seems to be a prompt evaluation of a variety of plans under a range of assumed conditions which the planner defines.

The most important reason for designing an adaptable simulation model development is the very survival of the simulation. An adaptable and changing model is essential if the model is to be used over an extended period of time. Assuming the model is to operate as an agent for improving the planning process, its main function may well be as a stimulus to search for a definition of what the manager has not included in the model. This improvement process seems to be one of continuous redefinition of the manager's concept of the pertinent forces in the environment and growth in the modeler's ability to adequately represent these forces. The model must continuously reflect the manager's improved concepts or fall into disuse by the decision makers. A successful stimulation project for planning will stimulate the continuous growth of the participants as evidenced by an improving model.

REFERENCES

- ¹ J L MCKENNEY
A clinical study of the use of a simulation model
The Journal of Industrial Engineering January 1967 p 30
- ² G L URBAN
A new product and decision model
Management Science April 1968 p B490
W W LEONTIEF
Proposal for better business forecasting
Harvard Business Review Vol 42 No 6 p 166
- ³ R D BUZZEL R A BAUER
Mating behavioral science and simulation
Harvard Business Review Vol 42 No 5 p 116
D W CRANE
A simulation model of corporation demand deposits
In K J Cohen and F S Hammer Analytical Methods in Banking
Irwin 1966

Operations research in a conversational environment

by MICHAEL M. CONNORS

*IBM Scientific Center
Los Angeles, California*

INTRODUCTION

The purpose of this paper is to discuss the types of operations research techniques and the kinds of operations research/management science problems that are possible in the conversational or interactive programming environment. It is widely agreed that very meaningful technological advances can be made by merging the things that a computer does best with the things that a human being does best. This is often discussed in terms of a man-machine symbiosis, cybernetics or "interfaces." The shortcomings in all discussions of this sort are twofold: first, a characterization of problem attributes which necessitates the power of an interactive environment is missing; and, second, very few practical or even implementable applications are ever discussed.

The present paper hopefully represents a divergence from past remarks in the area in the sense that both the first and second points will be treated. The second section discusses the relations between past and present activity in OR/MS and the available computer technology; this discussion is extrapolated to characterize the types of OR/MS activity and research which are indicated by an interactive or conversational environment. Building on the ideas in the second section, the following section discusses a potentially key application area which will be penetrable with interactive systems: scheduling of social processes. The last section discusses and illustrates the possibility of an interactive or conversational programming environment providing a framework for the enlargement of the technique of simulation to a generalized, optimizing tool. The ideas are presented by describing their potential application to multi-location, multi-echelon distribution systems.

Implications for research

To a large extent, the development of operations research techniques has been dictated by available com-

puter technology. In the beginning, the disciplines of operations research and management science treated situations which could be dealt with by analytical techniques. On the one hand, as the types of problems which could be solved analytically were solved, and, on the other hand, as computing capability matured into the batch mode technology, larger scale management science problems were tackled. The availability of batch mode technology caused theoretical research and applications development to emphasize general and widely utilizable algorithms and techniques. The *problems* or *class of problems* treated were common to enough different people that a single technique, algorithm or applications package was capable of treating a wide spectrum of situations. The question remains: what is the nature of this commonality?

The solution process for any problem can be thought of as a directed network: the nodes in the network represent decision points; the arcs are taken to represent the computational processes necessary to proceed from one decision point to the next. In this abstract conception of a solution process, the nodes (decision points) may be of two general types: open-loop or closed-loop. By an open-loop decision point, we mean a point which, when encountered in the solution process, dictates a prespecified set of actions. An example of this type of solution process is the linear programming algorithm where nodes are branches to error routines for infeasibilities, infinite solutions, and so on. In contrast, a closed-loop decision point is one which, when encountered in the solution process, dictates a set of actions conditioned on certain information. This information may be the state of the system or it may be a pattern of events emerging in the calculation. It may even be information derived from a source external to the system and the solution process. An example of a problem which requires a solution network with closed-loop nodes is the elective admissions scheduling problem which will be discussed in more detail in the next section.

The ideas in the preceding paragraph can be used to characterize the types of OR/MS problems which have been suitable for solution with batch mode technology. These are problems in which the solution can proceed without intervention from outside the solution algorithm. This is equivalent to requiring that the nodes be open-loop nodes or, at worst, closed-loop nodes in which the information required for the new decision is contained in the state description of the process under analysis. The commonality of *problems, or classes of problems* mentioned above can be thought of in terms common attributes of the structure of their solution processes: problems amenable to solution with batch mode technology are not merely those physical situations which are common to a number of people but rather those which have a common solution process *in terms of the required solution network*.

The third stage of development of management science and computer technology is in progress. The relevant computer technology is that of interactive or conversational programming. The availability of conversational technology (as well as graphic technology) allows the present research frontier to be directed away from open-loop solution processes and towards closed-loop solution processes. Through the blending of conversational and graphic technology with mathematics, the closed-loop solution processes enable the user to solve problems which were unsolvable in the previous technology. In fact, with batch mode technology, these problems were not of interest since researchers had problems amenable to solution by open-loop processes available for research purposes and so were not interested in considering the more complicated and less mathematically elegant closed-loop processes.

Certain elements of the research community have come to recognize the almost logical impossibility of developing an open-loop conditional branching network for "third generation" MS/OR problems. It is clearly possible to treat any particular branch of such a conditional network by open-loop mathematical or algorithmic techniques. Similarly, it may be possible to treat more than a few of the branches in the network with any single open-loop algorithmic technique. When it becomes necessary to treat more than just a few of the branches in such a network, more general ideas must be sought. A branching decision may require a broader information background, depend on pattern recognition capability or require interaction from outside the algorithm. The solution is not to attempt to embody the decision analysis in an algorithmic package but to relegate the decision making or branching decisions to another source. The availability of conversational programming technology makes it possible to imbed the researcher in the algorithm at each major branch

or decision point. In this way, interactive programming technology can be brought to bear on problems requiring closed-loop solution techniques. The result is that computations are relegated to algorithmic processes and (perhaps non-quantifiable) decisions are consigned to the researcher. It then becomes possible to develop closed-loop solution processes by concatenating several open-loop solution techniques with the human decision maker directing the flow of the process at the decision or conditional branch points in the solution network. In order to understand how this can be done, and also in order to understand the advantages that might accrue from utilizing this approach, it will be appropriate to consider two particular applications which represent penetration applications into this general area of operations research in a conversational environment.

Interactive scheduling: Social processes

The previous sections have characterized the environment of interactive scheduling models as being one in which the branching conditions of the computational process cannot be tested in an open-loop mode. This situation occurs in scheduling social processes because the scheduling procedure cannot be completed until the scheduler interacts with the people or persons being scheduled in order to ascertain the feasibility of the proposed schedule. If the proposed schedule is infeasible or undesirable, new constraints must be added to the problem statement and the computations must be performed again. An example of this type of scheduling problem exists in the area of scheduling elective admissions to a hospital.

The remainder of this section describes an elective admissions scheduling algorithm designed to operate in the real time, conversational environment inherent to a hospital admitting office. The algorithm has been initially implemented in a batch processing mode, is to be operated in parallel with an elective admissions scheduling system at the UCLA Hospital and will be installed in a teleprocessing/conversational situation if the algorithmic technique proves to be suitable for this purpose.

The problem of scheduling elective patients for admission into a hospital can be viewed as a problem in scheduling or allocating scarce resources. The elective admissions scheduling decision must achieve a balance between the patient's preference for type of facility and time of admission and the hospital's desire to achieve an allocation which will allow it to operate at maximum efficiency. The admissions scheduling problem is more general than those treated in the literature on scheduling theory for two reasons. The first complication re-

sults from the randomness associated with the arrival of requests for services and the randomness of the length of time the facilities are occupied once the admissions decision is made. The second complication arises from the fact that the scheduling process cannot proceed in an open-loop manner because the schedule must be verified for its suitability with the admittee.

The design of the algorithm has assumed certain environmental characteristics for the hospital, its admitting department and the admission scheduling process. The information flows assumed by the algorithm can be characterized as in Figure 1. Two of the assumptions implied by Figure 1 hold great implications for the design philosophy of the algorithm. The first assumption is that requests for admission into the hospital environment are received completely at random by the admissions office. Therefore, no attempt is made to forecast or predict the demand for services placed on the hospital. It is clear that a more efficient computational technique could be developed if it were possible to predict requests, arrival times and space requirements. However, this type of information is not available in the operating environment of an admissions office and so the algorithm does not require this type of input in order to effect a scheduling decision. The second assumption made by the computational procedure is that sufficient historical information exists to enable the algorithm to view each individual patient as a separate random variable with his own probabilistic characteristics. This assumption forces the decision maker to make a separate computation for each request that the admissions office receives. Since all of the information required to make this computation is available at the

time the request is made, it seems reasonable to perform the computation in a manner which will return the decision to the admittee essentially immediately. This, in turn, requires a conversational, teleprocessing approach.

Figure 1 indicates the information flows used in the computational process. Figure 1 does not however indicate the criterion by which the data are processed in order to reach a scheduling decision. Therefore, a third key assumption in the development of the algorithm is that the trade-off between patient satisfaction and hospital operating efficiency can be quantified. It is clear that patient satisfaction is related to the degree to which his needs for health services can be served while minimizing his personal inconvenience. It is also clear that the degree of hospital operating efficiency is related to the level of resources which the hospital must commit in order to satisfy these patient needs. On the patient's side it does not appear to be possible to quantify the level to which the hospital satisfies his health needs. On the other hand, it does appear to be possible to quantify the amount of personal inconvenience the patient suffers as a result of being scheduled into the hospital at certain points in time. Thus the measure of patient satisfaction is assumed to be related to the difference between his scheduled admission day, d , and the set of admission days he prefers, C . It is a somewhat easier question to quantify the operating efficiency of the hospital. For this purpose the hospital administrator is allowed to specify a nominal occupancy $n(d)$ for each day d which results in maximum utilization of the resources available to him. Then the operating efficiency of the hospital can be characterized by the difference between actual occupancy $o(d)$ and nominal occupancy $n(d)$ resulting from any particular schedule of admissions he has specified. The functional form of these criteria is taken to be quadratic. The optimization problem can then be stated as: subject to constraints to be discussed below, find the optimal admission day d^* such that

$$d^* = \min^{-1} \{ C_s (d - c) + E[C_d (o(d) - n(d))] \}$$

$$d \in P$$

$$c \in C$$

where P is the set of days in the planning horizon, C is the set of patient-specified days, C_s is the scheduling cost of deviating from the patient's set of convenient days and C_d is the cost of actual hospital occupancy deviating from nominal hospital occupancy. E is the expectation operator taken with respect to the probability density function of the hospital census for each day in the planning horizon.

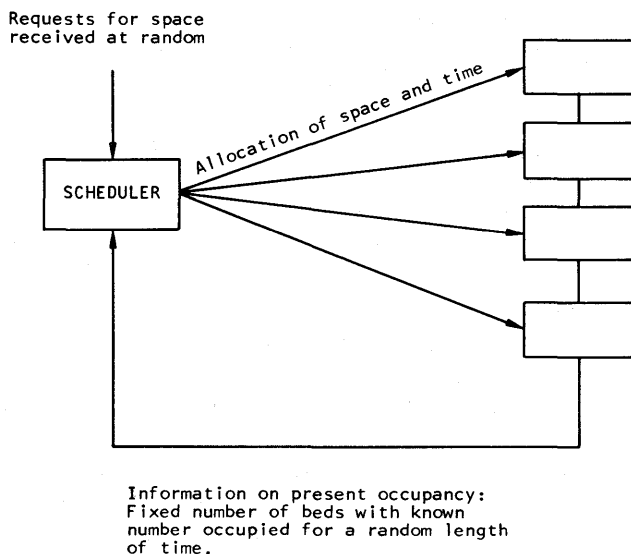


FIGURE 1—Simple model of elective admissions scheduling problem

The algorithm requires that certain types of information accompany a request for admission. In addition to information for identification purposes, additional information is required. This includes an admitting diagnosis of the patient according to the International Classification of Diseases Adapted for Indexing Hospital Records,¹ the patient's age, his sex and whether or not surgery is to be performed. The admittee must also specify a list of days on which he desires to be admitted into the hospital, his preference, if any, for type of accommodation, and specification of the medical service to which he seeks admission. This information is used to interrogate ¹ published by the Commission on Professional and Hospital Activities. This volume contains the mean and standard deviation of the length of stay of patients categorized by these variables. The mean and standard deviation are used to generate a probability density function according to the gamma density function. The gamma density function was selected for the length of stay distribution because of its previously observed success in fitting empirical length of stay data. It is often the case that the admittee's physician may be able to provide more detailed information on the length of stay and in this case the physician's estimate of the length of stay is taken to be the mean of the length of stay distribution and supercedes the information contained in the Professional Activities Study. In the event that an admitting diagnosis by the international disease classifications is not available, the operator of the algorithm specifies these values either arbitrarily or by administrative precedent. It may also occur that a given hospital may experience length of stay distributions which differ from national experience. In this case, the individual hospital can develop its own procedures for generating the required mean and standard deviation, as well as for generating different functional forms for the length of stay distribution. In any event, the algorithm is, at this point, in possession of a suitable set of patient attributes and can proceed to generate an appropriate schedule for the admittee.

The algorithm operates subject to both deterministic and probabilistic constraints. The deterministic constraints are that the admittee's attributes must match the attributes of those patients occupying any room into which he is scheduled. The admittee's attributes are binary or can be represented by combinations of binary variables. The attributes of a particular patient are clearly deterministic. But the question of matching these attributes with those of patients already occupying the facilities is probabilistic since the algorithm is only in possession of probabilistic information about the future utilization of these facilities. The algorithm recognizes this fact and for purposes of an admitting decision requires a matching of attributes only with the

patient most likely to be in the room on the proposed admission day. The fact that the person thought to be most likely to be in the assigned room when the admissions decision was made may not actually be there when the admission day arrives is recognized by the algorithm and is taken into consideration in all subsequent admissions decisions. The algorithm also insures that actual admissions are made only into facilities which are occupied by patients with attributes compatible with those of the admittee.

The operation of certain probabilistic constraints is also recognized. The hospital cannot schedule more patients for occupancy than there are beds available. However, the hospital census is not known with certainty. Therefore, the only constraint to overloading at the hospital level must be a probabilistic one: the probability of a hospital overload if the present request is admitted must be below a specified threshold level. It is clear that in order to evaluate this constraint, the algorithm must be in possession of a probability distribution for the hospital census on each day of the operating horizon. Since the probability distribution of the admittee is derivable from the admitting information, the required probability distribution can be calculated by known techniques. A description of the calculations involved in this process as well as a discussion of the assumptions necessary to justify this calculation is contained in reference². Analogous remarks apply to constraints on service overload for those hospitals organized on a service basis.

It is also regarded as undesirable to move patients once they are located in a room. Again, the occupancy of a given room is known only probabilistically, the constraint on moving patients must be a probabilistic one: the probability of having to move a patient out of a given room at some later time if the present request is admitted into that room on the day being considered must be below a specified threshold level. If these constraints are satisfied for a particular day, the algorithm evaluates the day as a prospective admission day. This results in the figure of merit depending on C_s and C_d for each day. The figure of merit associated with each feasible day is retained and is used in determining the order in which days are offered to the admittee during the negotiation process.

The algorithm then processes this information in order to produce a list of days which are feasible for admitting the patient. The list is ordered in decreasing order of desirability in terms of the figure of merit specified by the hospital. The days are produced on the output device and are then available to be negotiated with the patient until a suitable and mutually agreeable day is found. When a mutually suitable day is found the

patient is entered in the algorithm's admission log, the algorithm updates all relevant files and prepares itself to accept another request for admission. The algorithm has also kept track of the particular room and bed which will be best for the admittee for each feasible day and will tentatively reserve these facilities for him on the date selected for admission. In the event an acceptable day is found, options are available for changing various constraints, for example, a patient's demand for a private room, and re-entering the algorithm to find additional feasible admission days. In this latter event, the necessity for a conversational, real-time processing environment is apparent.

The elective admissions scheduling problem clearly necessitates a closed-loop solution process since the solution cannot be completed without a (non-quantifiable) external decision. In this application, the full potential of on-line computing technology has not been used. Only a very simple external intervention is necessary to make a fairly simple decision: admit the patient or re-enter the computational process (perhaps after changing some of the constraints on the computations). The more sophisticated applications of the conversational environment allow for closed-loop nodes at which the entire mathematical problem can be restructured or new algorithmic strategies can be employed. A vehicle for experimenting with these more advanced applications of on-line technology is discussed in the following section. While the material in the next section is a more sophisticated topic, it is of less interest as an applications topic since the majority of the research is still in an experimental stage whereas the work described in this section is demonstrable.

Generalized simulation

Experimentation with mathematical models of systems on the computer has been of interest to the operations research/management science profession since the advent of the high speed computer in the 1950's. The technique of simulation cannot be regarded as a generalized solution technique since the experimentation does not necessarily lead to an optimal solution of the problem. The quality of the result has depended upon the ability of the experimenter to observe the results of a simulation run, redirect the activity of the simulation and iterate this sequence until a maximal behavior was obtained from the simulated model.

The technique of simulation has two shortcomings which must be corrected before simulation can be regarded as an analytical tool rather than an experimental technique. The first deficiency is that simulation is not a general technique. The non-generality of the technique arises from the fact that the simulation approach involves setting up a model of a particular empirical

situation; the difficulty is that since the simulation or mathematical model has been tailor-made for the particular physical situation being examined, the model is obsolete once the particular problem at hand has been analyzed. In analysis, the designer or analyst can use general techniques to model or analyze a large number of different systems. The characteristic of analytical techniques is that they are capable of being brought to bear on functionally different classes of problems or functionally different problems within the same class of problems. Simulation techniques, on the other hand, are capable of being brought to bear only on problems which are parametrically different. It is clear therefore that in order to make simulation a general solution tool, it will be necessary to enable simulation to be used on functionally different systems. In this regard, it's fair to say that the simulation approach effects a trade-off between the level of *detail* that can be brought to bear on one particular problem and the *number* of particular problems that can be analyzed.

The second shortcoming of simulation is that the simulation approach does not prescribe what to do with variables once they have been introduced into the simulation; that is, the simulator is only capable of evaluating the particular configuration of policy functions and status variables that are presented within the context of the simulation. Simulation could be made a general or optimizing solution technique if it were possible to insure that the progression toward optimal parameters and/or policies converges. Conditions for convergence of parameters and policies in simulation often cannot be described in terms of the status variables or problem parameters. These conditions often depend upon relatively unquantifiable factors or on a set of factors not built into the model (e.g., the 'experience' factor). The conversational environment can contribute to a solution to this problem by providing a mechanism for directing the convergence procedure, namely, the experimenter. In other words, we seek a simulation environment in which the analyst can dynamically redirect the course of the simulation in order to approach and hopefully attain optimal behavior.

The remainder of this section discusses an approach by which the technology that is available in the conversational programming environment might be used to overcome the deficiencies of simulation. The greatest impediment to universal usefulness of simulation as an analytical technique lies in its lack of generalizability. In order to treat this deficiency, we suggest an idea proposed by Ginsberg, Markowitz and Oldfather⁴ called programming by questionnaire. In addition, we will discuss the implications held by the conversational programming environment for optimization in simulation. We will discuss these techniques by considering their

applicability to the idea of simulating a multi-location, multi-echelon distribution system.

Consider a multi-location multi-echelon distribution system as might be represented by Figure 2. The mathematical problem is to analyze the combinatorial interdependencies which can arise from the several ways in which the stocking points might be supplied with the several products through the several periods of time over which the process is being analyzed. The mathematical problem is substantially complicated by the fact that, as in all inventory/distribution systems, the demand at the customer level is a stochastic variable. Thus the problem is a very large scale combinatorial problem with the added difficulty that the combinatorics depend on stochastic variables. The complexity and the level of detail required for a thorough analysis are clearly implications for using simulation as an analyzing technique. If, however, we are discussing a *class* of multilocation, multi-echelon systems, simulations has the severe shortcoming that it is not capable of analyzing functionally different systems and it is also clear that every individual who has this type of distribution problem has a problem which is particular to him. It would therefore appear that if one were trying to develop a theory of distribution systems of this type, that simulation would not be an appropriate analytical device. However, the author and H. M. Markowitz have used recently developed techniques in simulation⁴ together with some of the capabilities of a conversational programming environment make it possible to develop a tool for analyzing functionally different distribution systems.

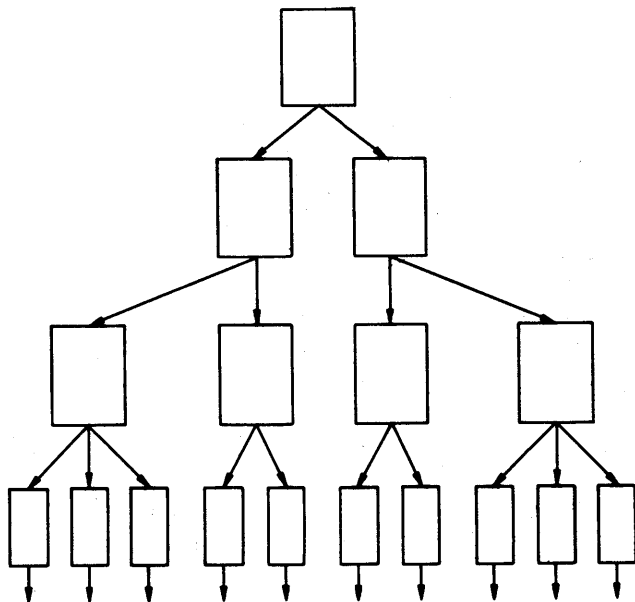


FIGURE 2—A multi-location² multi-echelon distribution system

Viewing Figure 2, there are obvious structural differences particular to each distribution system of this type. The number of stocking points, the direction of flow of goods and services, the relative orientation of stocking points, the number of products and the classes of customers faced by the system, all are particular to each individual user. In addition to the structural differences between various systems of this general character, there are functional differences from system to system. For each possible multi-location, multi-echelon system, each stocking point has its own order filling policies, its own replenishment policies, its own redistribution policies, and its own emergency replenishment policies. In addition, the system may operate under several different types of transportation policies, may be exposed to different types of random demand at various points in the system and may be subject to various cost structures throughout the components of the system. There may also be parametric differences between systems; that is, two systems may be functionally similar in terms of the operating policies mentioned above but the systems may differ in the parameters used by these policies. In order to be a useful analytical tool, a simulator must be cognizant of not only the structural differences between these systems but also of the possible functional and parametric differences that are associated with each such system.

An appropriate way to approach this problem is to construct the separate structural differences, to construct the separate policy functions, to construct the various random processes and then allow the user to configure his own simulator from among all of these possibilities at execution time. Clearly all of these capabilities could be built into a simulator which could choose the appropriate alternatives at execution time by means of logical tests. This obviously results in a simulation which is inefficient in the sense that large segments of code remain unexecuted due to their inapplicability in any one particular situation and in the sense that large amounts of storage are needlessly allocated in order to cope with the maximum storage demands placed on the simulator by any particular simulation.

As we have indicated, an appropriate way of solving this problem of compile time configuration of a simulation is to use the idea of programming by questionnaire introduced in Reference 4. The idea is to provide the individual analyst with a collection of tools or building blocks which he can configure into an appropriate model for any one of a very large number of distribution systems he may choose to analyze. The analyst is simply provided with a questionnaire by which he can indicate the characteristics of his particular distribution system. The questionnaire is processed by an editor program which draws upon a program source library containing

all of the various program modules necessary to successfully model the completed system. The editor program processes the questionnaire and uses the information drawn from the questionnaire to draw upon the program source library to configure a simulation deck. The analyst can then run this deck on a computer and be assured of an efficient simulation package yet one which has been tailor-made or particularized to his system from a large number of possible other systems. Figure 3 graphically portrays the interaction of the questionnaire, editor program and program source library.

The idea of this procedure appears to be intricate. In practice the most difficult part of this entire procedure is to define a questionnaire which adequately and accurately characterizes a general distribution system. Figure 4 presents a portion of an answer sheet for such a questionnaire.

The ability of the programming by questionnaire technique to solve the problem of non-generalizability in simulation is clear. What is not clear at this point is the relation of this technique to OR in a conversational environment. A partial answer can be given with refer-

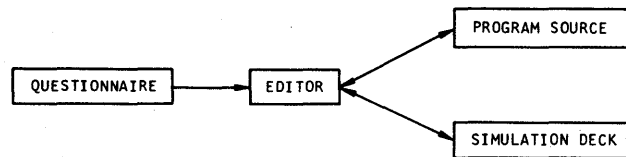


FIGURE 3—Interaction of components of DSS generator program

	DEPENDS ON				CAN BE GENERATED BY PROBABILITY DENSITY FUNCTIONS								
	ARE NOT USED	DEMAND SPECIFICATION	DEMAND POINT TYPE	ITEM TYPE	POISSON	NORMAL	UNIFORM	CONSTANT	TABLE LOOK UP	EXPONENTIAL	LOG NORMAL	ERLANG	USER
	1	2	3	4	5	6	7	8	9	10	11	12	13
INTER-DEMAND TIMES													
DEMAND QUANTITIES													
DEMAND PRIORITIES													
PARTIAL SHIPMENT POLICIES													
BACKORDER POLICIES													

FIGURE 4—Portion of answer sheet for DSS/I

ence to Figure 4. A major portion of using DSS lies in filling out the answer sheet. Figure 4 shows that some questions are conditioned upon the answers to preceding questions. A conversational environment could aid in the configuration of the required simulator by enabling the questionnaire to be answered via a graphic terminal and light pen. The cpu could analyze answers to past questions and present the correct follow-up questions. In addition, the terminal could provide additional information to the user during the process of filling out the questionnaire.

Up to this point in the discussion, we have only been concerned with the combination of the programming by questionnaire technique and an interactive system to produce a generalized Distribution System Simulator. The contribution of conversational technology to the generalizability of DSS is mostly through easing the clerical burden on the user. The contribution could become more substantial when we turn the discussion toward the question of a simulator with optimizing capability.

A Distribution System Simulator is an appropriate vehicle for studying optimization in simulation since the optimization procedure must choose among functionally as well as parametrically different systems. The question of choosing among parametrically different systems is one which has been dealt with before, e.g., which of several possible reorder points should be used at a given stocking point? Parametric optimization is a natural problem to treat in a conversational environment with graphic technology. The progress of the simulation can be observed on a graphic terminal and the program interrupted in order to change parameters.

The question of functional optimization is much more difficult because this implies introducing new functional policy and/or decision functions into the simulation, e.g., a new rule for allocating the results of a production run among stocking points. This in turn implies a dynamic recompilation of the simulator or portions of it. DSS has been designed to allow certain pieces of the simulator to be rearranged. In fact, all policy routines are of this form and so the simulator is appropriately designed for on-line experimentation. A remaining and, as yet unresolved, issue is whether the required dynamic recompiling of source code is feasible.

CONCLUSIONS

Two applications of OR/MS techniques have been discussed as illustrations of the types of solution processes made possible by a conversational environment. The applications differ in their depth of sophistication in a way which is inversely proportional to their state of

completion even though both applications exemplify the characteristic of problems requiring an interactive environment for solution, namely, a closed-loop solution network. The point of these applications is not that they are on-line adaptations of something that could just as well be done off-line but rather that the conversational programming environment makes the applications implementable in the first place. In addition, it is only through demonstrable applications of this type imbedded in a "theory of problems" for a conversational environment that research interest in interactive OR/MS can be developed.

REFERENCES

- 1 Commission on Professional and Hospital Activities
Length of stay in short-term general hospitals
McGraw-Hill Co New York 1966
- 2 M M CONNORS
A stochastic elective admissions scheduling algorithm
Los Angeles Scientific Center Technical Report 320-2616
July 1968
- 3 M M CONNORS H M MARKOWITZ
Simulation of multilocation multi-echelon distribution systems
In preparation
- 4 A S GINSBURG H M MARKOWITZ P M OLDFATHER
Programming by questionnaire
RM-4460-PR Rand Corporation Santa Monica April 1965

MEDIAC—An on-line media planning system

by LEONARD M. LODISH

*University of Pennsylvania
Philadelphia, Pennsylvania*

and

JOHN D. C. LITTLE

*Massachusetts Institute of Technology
Cambridge, Massachusetts*

INTRODUCTION

The problem of media planning in advertising is a natural one for the application of mathematical models and computers. A great deal of data are available on who reads (or sees or hears) what. There are considerable data on what kinds of people are prospects for which types of product. There are many different media options available. Many judgments must be made. It would seem that there should be some organized way of combining judgments and data into a model, setting an objective, and then optimizing it to produce a good media plan.

Indeed, a number of people and organizations have tried to do this. (A review of published work may be found in Little and Lodish.¹) We here report on a model that we have constructed and brought up as an on-line media planning system. Since May 1967 the system has been available on a commercial basis under the name MEDIAC. The implementation of MEDIAC has brought about evolutionary changes both in the system and its users. Interaction with media planners has led to an increase in both the complexity of the model and the operating efficiency of the system. At the same time the media planners have changed their methods of planning due to exposure to MEDIAC.

In this paper we shall briefly describe the model, its optimization, and the computer system. Then we shall give an example of an application and discuss some of our experiences with the system.

Model

The media planning problem may be stated as follows: Given a set of media options, a budget, and various data about the media and the audience to be reached, which options should be used and when should they be used in order to maximize profit or some related measure of performance? By a media option we ordinarily mean a detailed specification of the place, position, size, and other outward characteristics of an advertisement, but not the message and copy treatment. Thus, the role of media in the advertising process is to expose a chosen audience to the advertiser's messages in an efficient manner.

Relevant to this objective are at least the following phenomena:

- Market segmentation
- Sales potential of individual segments, i.e., their relative importance to the advertiser
- Media coverage in each segment
- Overlaps in media audiences, both across media and across time
- Forgetting by those exposed to the ads
- Diminishing returns at high exposure rates
- Media costs
- Intermedia differences in the values of an exposure
- Seasonality of sales potential and audiences

The purpose of our model is to combine the above phenomena into a flexible, consistent structure which the media planner can use to evaluate alternative media plans and concepts. The model will be verbally described below. For a detailed technical description, see.¹

First, it is supposed that market segments have been defined. Perhaps men and women have different sales potential and therefore represented as different segments. In addition, perhaps these are each broken down further into geographic regions and income classes. Segments may be defined in any manner as long as suitable media coverage data can be supplied for them.

Second, an advertising insertion in a given media creates a probability of exposure for a person in a specified market segment. The probability of exposure depends on the audience of the medium within the segment, and on the size, length, color, or other characteristics of the insertion. The exposure of a person to a medium is not independent of exposure to other media or to the same medium at another time, but depends on media overlap probabilities.

Third, the advertising exposure creates value in the people exposed, i.e., disposes them more toward buying the product. The amount of value created by one exposure depends on the medium and is called the exposure value of the medium.

Fourth, people forget. It is usually assumed that people forget a constant percentage of exposure value in each time period although more complicated relations are possible. Figure 1 shows how the retained exposure value of an individual might change with time.

Finally, people act. As the level of exposure for a person rises, so does the anticipated re-

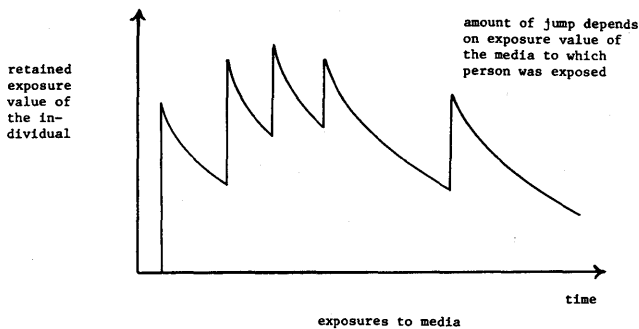


FIGURE 1—Retained exposure value as it might vary over time for a specific individual

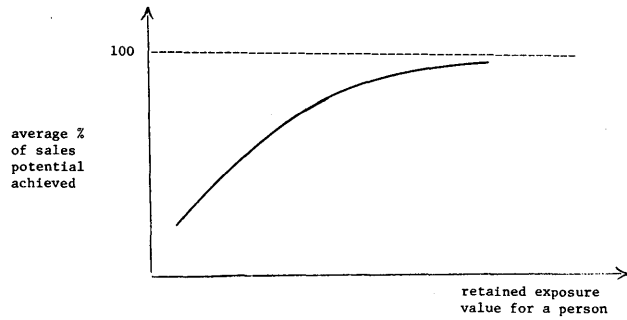


FIGURE 2—Customer response

sponse, but it does so with diminishing returns. The relationship might be as shown in Figure 2.

The anticipated total response in a time period is the sales potential of each market segment times the average percent of potential achieved by people in that segment summed over all segments. The average percent of potential achieved in a segment is calculated by an analytic approximation that involves the characteristics of the probability distribution of exposures to people within the segment. Thus the model, a simulation by some definitions, is not a simulation in the sense that individuals in a population are treated one at a time by the computer.

To summarize: advertising insertions in media generate exposures in the market segments. Exposures have value and raise the exposure level of some people in the segments, although this will decrease with time because of forgetting unless further exposures occur. The exposure level generates an anticipated response. Diminishing returns occur at high exposure levels.

Optimization

Heuristic methods are used to select and schedule media for large total response. The exact optimization problem is one of integer programming with a nonlinear objective function. Efficient exact methods are not available for this, but fairly simple heuristics appear to give very good results. The present routine starts from the set of required media (if any) and adds new media one at a time according to a criterion of largest incremental response per dollar, taking into account all media inserted up to that point. The routine stops when the budget is exhausted.

Computer system

The current version of MEDIAC contains four conversational programs which the user may call at his option:

- (1) *INPUT* accepts data from a teletype and stores it on a disk file. The required data and its format are requested by the computer in English. The user may elect options which help to eliminate repetitions of certain input.
- (2) *CHANGE* enables the user to change any stored data on the disk.
- (3) *PRINT* prints out the input data stored on disk in a management report format.
- (4) *MEDIAC* is the actual calculation program. Its input is a disk file prepared by *INPUT* or *CHANGE*. The user may use three different options in this program. He may request:

1. *Evaluation* of a particular schedule in terms of its expected response in total and its effects in each market segment and time period.
2. *Initial Ranking* of all available media options. This is based on the incremental value per dollar for each option in the absence of others in the schedule. The rankings can serve as an effective initial media screening device.
3. *Selection and Scheduling* of the media options. A detailed output of the effects of the generated schedule in each market segment during each time period is also generated at the user's option as well as a chart of the schedule by time periods.

The system is up on a time sharing computer utility with which users communicate via telephone lines. MEDIAC represents a step beyond the computer utility toward a mathematical model utility.

Example

The following example is a slightly-coded and cut-down version of an actual application. The run was made as part of the planning process in the development of a television schedule.

The product at hand is used almost exclusively by women. We have dubbed it "Princess Widgets." Heaviest use is among women aged 25-34 in A (i.e., metropolitan) counties, although signifi-

cant usage is found in other groups. The relevant market segments have been broken out by age and county type as follows:

Age: 25 and under / 25-34 / 35-49 / 50-64

County type: A / B / C + D (abbreviated C)

PRODUCT: PRINCESS WIDGETS
Budget \$400,000 5 Periods 12 Segments 9 Media

Media Data									
Media Option	ASOAPD	BSOAPD	DAGAME	NMOVIE	NSITUA	NADVEN	NSITUA	FRINGA	FRINGB
Cost/Insertion	6800	11100	7300	50000	46400	37900	46000	5000	4000
Exposure Value	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Probability of Exposure	.35	.35	.35	.40	.40	.40	.40	.35	.35
Upper Bounds	12	8	12	4	4	4	4	40	40

Audience Seasonality: None

Segment Data												
Market Segment	25 Und	A	A	25-34	35-49	A	A	B	B	B	C	
	25 Und	25-34	35-49	50-64	25 Und	25-34	35-49	50-64	25 Und	25-34	35-49	50-64
Population	4000	4470	6930	5860	2810	3140	4880	4120	3490	3910	6060	5120
Sales Potential	1.56	1.43	1.33	.98	1.49	1.36	1.25	.90	1.20	1.10	1.01	.72
Initial Exposure Value/Person	.100	.098	.082	.064	.101	.095	.082	.061	.085	.083	.070	.056

Potential Seasonality: None

Other Data
Memory Constant: .250
% Potential Realized at Saturation: 100.
1 Average Exposure: 50. 2 Exposures: 70. 3 Exposures: 80

All twelve combinations are considered. Existing TV shows representative of types expected to be available in the planning period are used in the analysis. The shows selected for consideration have been chosen with the objective of permitting a mix that will cover high potential groups efficiently. The media abbreviations are:

- | | |
|---------------------------|----------------------------------------|
| ASOAPD = Daytime Serial A | NSITUA = Night Situation |
| BSOAPD = Daytime Serial B | FRINGB = Fringe Spots in County Size A |
| DAGAME = Daytime Game | FRINGB = Fringe Spots in County Size B |
| NMOVIE = Night Movie | |
| NADVEN = Night Adventure | |

The following illustrations give a summary of input data for the problem and an abbreviated transcript of the console session at which this problem was run. Included are:

- (1) Input of the data.
- (2) Preliminary ranking of the media choices.
- (3) Detailed build up on the schedule for the given budget and other input.
- (4) Summary of final schedule.
- (5) Detailed results of retained exposure value by market segment and time period.

MARKET COVERAGES

	A 25und	A 25-34	A 35-47	A 50-64	B 25und	B 25-34	B 35-47	B 50-64	C 25und	C 25-34	C 35-49	C 50-67
ASOAP	.096	.068	.056	.052	.104	.073	.060	.056	.115	.081	.067	.063
BSOAP	.111	.096	.075	.079	.104	.090	.071	.074	.139	.120	.094	.099
DAGAME	.093	.085	.081	.089	.103	.094	.090	.098	.096	.088	.084	.09
NMOVIE	.321	.313	.249	.206	.302	.294	.235	.194	.274	.266	.212	.175
NWESTN	.129	.051	.104	.114	.144	.102	.116	.127	.208	.147	.167	.183
NADVEN	.209	.211	.192	.101	.241	.242	.220	.116	.166	.167	.153	.080
NSITUA	.272	.245	.205	.208	.241	.217	.182	.184	.205	.185	.155	.156
FRINGA	.120	.120	.100	.080	.000	.000	.000	.000	.000	.000	.000	.000
FRINGB	.000	.000	.000	.000	.120	.120	.100	.080	.000	.000	.000	.000

MEDIA DUPLICATIONS

	ASOAPD	BSOAPD	DAGAME	NMOVIE	NWESTN	NADVEN	NSITUA	FRINGA	FRINGB
ASOAPD	.036	.037	.035	.040	.034	.035	.037	.025	.025
BSOAPD		.048	.042	.051	.047	.048	.050	.041	.041
DAGAME			.043	.047	.042	.044	.046	.038	.038
NMOVIE				.133	.054	.059	.090	.050	.050
NWESTN					.067	.050	.070	.046	.046
NADVEN						.078	.071	.048	.048
NSITUA							.097	.050	.050
FRINGA								.050	.000
FRINGB									.050

TYPE DUPLICATIONS OF ASOAPD WITH
ASOAPSOADAGAMNMOVIEADUNSTIFRINFRIN
.XXX.XXX.XXX.XXX.XXX.XXX.XXX.XXX.XXX.XXX
*036.037.035.040.034.035.037.035.035

STOP

The data bank for this problem is now created.

The user then asks for an initial ranking of all the media options

--GO (G2A011)/@MEDIAC/

TYPE 1 IF INITIAL EXPOSURES ARE ZERO, OTHERWISE 2

--1,

TYPE 1 IF RANKING WANTED, 2 FOR FULL ALLOCATION

--1,

DAGAME T. P. 1 IMPACT/DOL= .151

DAGAME T. P. 2 IMPACT/DOL= .151

DAGAME T. P. 3 IMPACT/DOL= .151

;

ASOAPD T. P. 1 IMPACT/DOL= .133

;

FRINGA T. P. 1 IMPACT/DOL= .123

;

BSOAPD T. P. 1 IMPACT/DOL= .107

;

NMOVIE T. P. 1 IMPACT/DOL= .067

;

NADVEN T. P. 1 IMPACT/DOL= .065

;

NSITUA T. P. 1 IMPACT/DOL= .065

;

FRINGB T. P. 1 IMPACT/DOL= .054

;

NWESTN T. P. 1 IMPACT/DOL= .039

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

This time the user desires a full run of the MEDIAC selection and scheduling system.

--GO (G2A011)/@MEDIAC/

TYPE 1 IF INITIAL EXPOSURES ARE ZERO, OTHERWISE 2

--2,

TYPE INITIAL EXPOSURES/CAP. IN SEGMENT #25UND #F4.

--1,

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

If some media must be in the schedule, they are added here.

1	INSERT IN DAGAME TIME PER	5	COST	7300.	REALIZED POT.	4797.
1	INSERT IN DAGAME TIME PER	3	COST	14600.	REALIZED POT.	5913.
1	INSERT IN DAGAME TIME PER	2	COST	21900.	REALIZED POT.	6996.
1	INSERT IN DAGAME TIME PER	4	COST	29200.	REALIZED POT.	8055.
1	INSERT IN DAGAME TIME PER	1	COST	36500.	REALIZED POT.	9094.
1	INSERT IN DAGAME TIME PER	5	COST	43800.	REALIZED POT.	10058.
1	INSERT IN DAGAME TIME PER	2	COST	51100.	REALIZED POT.	11004.
1	INSERT IN DAGAME TIME PER	4	COST	58400.	REALIZED POT.	11926.
1	INSERT IN DAGAME TIME PER	1	COST	65700.	REALIZED POT.	12836.
1	INSERT IN DAGAME TIME PER	3	COST	73000.	REALIZED POT.	13725.
1	INSERT IN DAGAME TIME PER	1	COST	80300.	REALIZED POT.	14537.
1	INSERT IN DAGAME TIME PER	3	COST	87600.	REALIZED POT.	15322.
1	INSERT IN DAGAME TIME PER	5	COST	94900.	REALIZED POT.	16150.
1	INSERT IN DAGAME TIME PER	2	COST	102200.	REALIZED POT.	16994.
1	INSERT IN DAGAME TIME PER	4	COST	109500.	REALIZED POT.	17638.
1	INSERT IN DAGAME TIME PER	1	COST	116800.	REALIZED POT.	18329.
1	INSERT IN DAGAME TIME PER	3	COST	124100.	REALIZED POT.	18966.
1	INSERT IN DAGAME TIME PER	2	COST	131400.	REALIZED POT.	19673.
1	INSERT IN DAGAME TIME PER	1	COST	138700.	REALIZED POT.	20276.
1	INSERT IN DAGAME TIME PER	4	COST	146000.	REALIZED POT.	20855.
1	INSERT IN DAGAME TIME PER	1	COST	153300.	REALIZED POT.	21455.
1	INSERT IN DAGAME TIME PER	3	COST	160600.	REALIZED POT.	21955.
1	INSERT IN DAGAME TIME PER	2	COST	167900.	REALIZED POT.	22449.
1	INSERT IN DAGAME TIME PER	4	COST	175200.	REALIZED POT.	22947.
1	INSERT IN DAGAME TIME PER	1	COST	182500.	REALIZED POT.	23395.
1	INSERT IN NMOVIE TIME PER	5	COST	189800.	REALIZED POT.	23967.
1	INSERT IN NMOVIE TIME PER	3	COST	248000.	REALIZED POT.	26950.
1	INSERT IN DAGAME TIME PER	5	COST	299800.	REALIZED POT.	30002.
1	INSERT IN NMOVIE TIME PER	1	COST	307100.	REALIZED POT.	30393.
1	INSERT IN NMOVIE TIME PER	2	COST	362100.	REALIZED POT.	35270.
1	INSERT IN NMOVIE TIME PER	4	COST	417100.	REALIZED POT.	36129.
1						

```

MEDIAC GENERATED SCHEDULE
MEDIA PER. 1 2 3 4 5
DAGAME 7 6 6 6 6
NMOVIE XX X XX XX X

The system prints a trace of
the build up of the schedule
and then a graphical display
of the final schedule.

TYPE1 FOR DETAILED OUTPUT,ELSE2
-1.
SEGMENT TIME P EX VAL/CP SEG E.V. REALIZED POTENTIAL
:1 SUND 1 .52411 2356. 1248.
A25UNF 2 .43682 1747. 960.
A25SUN 3 .52831 2113. 1108.
.
.
.
COMPUTER TIME MINUTES
*STOP*

+
-LOG
TIME USED 00:07

The user desires to see
detailed output describing
average retained exposure
level/person and realized
potential in each segment
during each period of the
analysis.

Total time used, including
input, was 42 minutes of
terminal time and .97
minutes of CPU time.

```

A few comments are worth making on the final schedule. Under the conditions of the problem, daytime shows covered the desired audience most efficiently. (See initial rankings.) As a result, the computer took the highest ranking daytime show, DAGAME, and used a considerable amount. If higher diminishing returns had been used, less DAGAME would have been scheduled because of its duplication with itself. After using considerable DAGAME, the computer skipped over the other daytime shows to the much lower ranked NMOVIE. This is because NMOVIE picks up new people not covered well by daytime TV. If the budget had been substantially larger, more media would probably have come in. The media director whose problem this is thought that fringe spots would be useful. However, even when their exposure value was increased slightly and their cost decreased slightly in a sensitivity test, they did not come into the schedule. Apparently, after the other shows have been carefully picked, the spots have little to offer in this situation.

Discussion

What have we learned from the exposure of our model and system to media planners?

Real world problems are viewed as large and analytically complex by those responsible for solving them, even though their past approaches have been analytically quite simple. We are constantly being pressed to increase our capability particularly with respect to number of media alternatives, market segments and time periods. We have increased our capacity from 15 media, 20 segments, and 12 time periods to 40 media, 20 segments and

13 time periods and are currently working on a much larger version.

Although our system is computationally very efficient compared, say, to a simulation approach to media planning, we feel economic pressure to make the system and heuristics run faster so that bigger problems become feasible in relation to the research budgets of the users. We have made the system at least three times faster in the past year for medium sized problems and even faster relatively for problems as they become larger.

An unwelcome consequence of larger problems is that they become less practical to operate on a time shared basis. In some busy periods the user may wait for three or four hours for a run taking twenty minutes of central processor time. To overcome this we have set up a system of on-line ordered batch processing for these large problems. The user still runs the conversational input, change and print programs at the console to enter his data or make any data changes. He then calls another conversational program which constructs a disk file containing instructions for the options desired in the calculation program. The calculation program is run overnight in a batch mode and creates a disk file with the system output. The user then enters the time-sharing system in the morning and prints out the results at his terminal.

We have also felt pressure to include more phenomena in our model of the advertising process. We presently are working on including competitive advertising, the rub-off effect of other advertising by the same firm, and the synergistic effects of multiple media types. The research is constantly prodded by our users asking for the inclusion of new phenomena. When these phenomena are included, they will undoubtedly ask for others.

At the same time, we have the distinct impression that we should not introduce complications too fast. Although the concepts in our model are all familiar to media planners, the process of developing data on each and putting them all into a single structure is not easy and requires time to absorb. This must be done for the most basic phenomena before going on to new complexities.

The interactive aspect of our system has been valuable in building confidence in the system among non-technical users. The user directly monitors what goes into the computer without being subjected to the rather imposing tribal rituals demanded by punched cards, system programmers, and computer operators. The user feels that

the system is in his control at all times and he knows exactly what data the computer is using to work on his problem. If any input mistakes are involved, he holds himself responsible. Most people seem to have more confidence in themselves than in other interfaces with the computer.

How has the use of MEDIAC affected the way our users think about their media planning decisions? MEDIAC tends to move the intuitive approach of the media planner to a more productive level. More effort goes into formulating the problem constructively than into trying to perceive the answer in one jump. Not that users automatically accept the model's answers, they test the answers against their intuition, dig into the model to find out what caused any substantial discrepancies and, in the process, appear to be updating and enriching their intuition.

The use of a rather comprehensive model structure has caused some planners to face problems whose importance had previously been unacknowledged. For example, one user tried three different response functions for a problem, one reflecting very high diminishing returns with repeated exposure, the second a moderate amount of diminishing returns, and the third almost no diminishing returns. The MEDIAC generated schedule was quite different for each curve. The importance of defining his client's objectives in terms of the value of repeated exposure was thus dramatized by sensitivity analysis on the model.

The model tries to give the users a sensible, consistent, comprehensive structure for media planning. The model leads people to look at many issues and ferret out information on all of them.

It also helps sort out relevant material from the deluge of data which planners invariably encounter.

Preparing problems for MEDIAC is often time consuming. Collecting the input data can be a time-consuming undertaking. A substantial amount of data manipulation is often needed because the data sources are frequently incomparable, the numbers are in different units, or the data must be extrapolated. We have made some headway into computerization of the data manipulation problem and consider it a fruitful area for development. Once the planner has done a few problems he usually finds short-cuts in input generation which save considerable amounts of time. Obtaining input for an average problem by an experienced MEDIAC user has taken from one to three man-days.

Users have been quite pleased with the results they have achieved with MEDIAC. Improvements in the objective functions, as defined by the users, have ranged from about 5% to 25% relative to previous schedules. Some MEDIAC generated schedules have looked much like previous ones; others have been quite different. In cases that have looked different, it has been possible to find out what data or phenomena have caused the change. In almost all cases the media planner has preferred the new schedule.

REFERENCES

- 1 J D C LITTLE L M LODISH
A media selection calculus
Sloan School of Management Working Paper 304-68 January 1968 To appear in Operations Research

Development and use of computer models for analysis of long range planning decisions at Southern Pacific Company

by E. P. ANDERSON

Southern Pacific Company
San Francisco, California

and

ROBERT K. McAFEE

Strong, Wishart & Associates
San Francisco, California

and

ALAN SEELINFREUND

Stanford University
Stanford, California

INTRODUCTION

Long range planning staff studies at Southern Pacific Company during the last three years have had as their primary objective the identification and analysis of major future decisions. These future decisions deal both with strategic, corporate problems and with operational investment problems; both types of problems are related and usually must be considered together. Some of these investment decisions will be made in the near future; the timing for others is as much as several years away.

Foreknowledge of decisions which are likely to be made even in the relatively distant future is important in view of the long planning lead times and long economic lives found in most railroad capital investment proposals. If today's decisions act to eliminate the most desirable future options, then the cost of not having identified and not having studied the likely future decisions can be great.

This sort of long range planning includes what has been termed "strategic planning." Some companies limit long range planning to a process that is mainly an extension of annual budget-making. In those companies, the esoteric task of strategic planning may be per-

formed entirely by the chief executive, with little or no staff participation. However, Southern Pacific's experience with complex strategic decision problems has tended to prove the value of increased participation by staff, department heads, and sometimes by outside consulting personnel.

The resulting studies of important factors in planning decisions have utilized a series of computer models. It is the purpose of this paper to discuss our approach to the analysis of planning problems, giving particular emphasis to a description of the computer models and their use. The paper begins with a description of planning problems that typically have been encountered. The second part of the paper describes the two most important computer models used: the operating cost model and the investment analysis model. The final part of the paper discusses the implementation and use of the models.

Description of the planning problem

Decision structure

From a conceptual viewpoint a typical planning problem may be divided into two parts. The first part

includes those problem elements which are largely subject to control: the "decision alternatives." The second part includes those elements over which little or no control can be exercised and which have an uncertain effect on the final outcome. For a given decision alternative and a given effect by an element of uncertainty, there is a unique conditional outcome or end state. The aggregate of values of all possible conditional end states in a planning problem is determined by all possible combinations of decision alternatives and effects of uncertainties.

The value for each conditional end state is deterministic, since it represents the answer to a series of "what if" questions. What if a particular decision alternative is adopted? What if each uncertainty element assumes a particular value? Calculation of values for conditional end states can be quite laborious in a planning problem of average complexity and, for this reason, a computer program is indicated. As this paper will later describe, the operating cost model and the investment analysis model provide the means for calculating the values of conditional end states and combining those values to form a distribution of outcomes.

The nature of the planning problem

We are concerned mainly with questions of when and where, as traffic increases, to modify railroad fixed plant. For example, railroads can add to fixed plant capacity by construction of second and additional main tracks parallel to existing single track lines. Centrally-directed traffic control systems which permit closer control over the movement of trains across the railroad system may be installed. Sidings, which are auxiliary tracks next to the main track that permit trains to meet and pass on single-track lines, may be required as traffic increases or train operating schedules are speeded up. Also, as the volume of traffic increases and functions of terminals change, there will be proposals to add to or modify fixed plant in rail yards and terminals.

The customary benefits used to justify fixed plant improvements and the acquisition of alternative routes have probably understated the value such investments have in allowing improved service to be offered more efficiently, due to the difficulty of translating such benefits into dollar terms. Indeed, labor savings and a few other easily-identified sources of savings have generally been the only benefits specifically attributed to most proposals. Our thesis is that by undertaking projects to make the fixed plant more efficient the railroad operation can be run at a lower overall operating cost. The reduction in operating cost results from economies in assigning motive power to trains. The more efficient motive power assignment is

made possible by reduced train delays in the more efficient system.

Of the many factors of uncertainty to be considered in the planning problem, the uncertainty of future transportation demand is often most significant. Not only is the volume of traffic to be transported uncertain, there is also uncertainty about the service requirements for this future traffic. Another important element of uncertainty is the outcome of efforts to merge with or acquire other railroad properties, which can have a marked effect on the desirability of an investment program. The uncertainty in future transportation demand is expressed in the form of probability estimates of the future rate of traffic growth, after studies and discussions with knowledgeable individuals. The uncertain outcomes of merger efforts are likewise incorporated as probability estimates, after interviewing lawyers and others involved.

A specific investment example

An important plant improvement problem studied is the proposal to extend double track over the Beaumont hill. Southern Pacific enters the Los Angeles Basin from the east through San Geronio Pass, west of Palm Springs, California. Transcontinental trains moving over this route originate and terminate as far east as St. Louis and New Orleans on Southern Pacific system lines and Chicago on the Rock Island. Nearly 40 trains climb the steep grades on both sides of the pass each day. Differences in uphill train speeds add to the problem of efficiently moving this large volume of traffic.

Between Colton, at the foot of the western slope, and Beaumont, at the top of the hill, there are two parallel main tracks for 20 of the 24 miles. On the eastern slope, from Beaumont down to Indio, the line is single-tracked for the entire distance of 46 miles with nine locations for meeting and passing trains.

Traffic over the Beaumont hill has grown rapidly in recent years, with the result that there has been an apparent need for increased main track capacity to maintain scheduled movement of trains into and out of the Los Angeles Basin. To provide this capacity, plans for extending existing double track lines to Indio on the east and Colton on the west have been proposed. Investment costs for these additions to fixed plant have been estimated.

The easily-identifiable direct cash benefits to be derived from double-tracking the Beaumont hill are small. In the corporate-wide competition for scarce capital funds, this project has therefore ranked relatively low. Management recognized that the proposal could produce important reductions in train delays

incurred in the meeting and passing of trains. However, there was no agreement as to the dollar value of eliminating these delays. The mathematical models to which we now turn were constructed to determine this value, and to provide a means for comparison with other investment alternatives.

Operating cost and investment analysis models

To evaluate the benefit accruing from an investment proposal like double tracking the Beaumont hill, an operating cost model was constructed to calculate the operating cost reduction that could result, at each of a number of assumed traffic levels, if trains were required to meet the same overall point-to-point schedules without incurring delays on the Beaumont hill. In the absence of these delays, less locomotive horsepower is needed to meet the same schedules; consequently an operating cost reduction could be realized by trading fixed plant investment dollars for savings in locomotive operating and ownership costs. Moreover, the cost model provides general guidelines for actually realizing these locomotive savings by identifying the reduction in locomotive horsepower requirements for each train operating over the Beaumont hill. The cost savings and probabilistic information concerning future traffic levels are input to an investment analysis program which calculates the probability distribution of net present value and of the return on investment for the Beaumont hill proposal.

The purpose of this section is to describe the formulation of the two mathematical models and the computational approaches to their solution.

The operating cost model

The operating cost model is an optimizing model whose primary purpose is to allocate locomotive horsepower in the lowest-cost manner yet still meet the train schedules demanded by railroad customers. The optimization is required since a proper comparison of the operating costs of two alternatives requires that each alternative be evaluated assuming its most efficient operation. Otherwise, it is possible to attribute relative benefits to a proposal which are in fact the result of operational improvements rather than the investment itself.

The railroad system under analysis is represented by a network. The nodes of the network correspond to terminals where changes in power assignment can be made. Links of the network represent sections of track called districts. Each district has associated costs and running times as functions of direction of travel and horsepower per ton of train weight. Also, there is an associated average delay time experienced by a train

forced into a siding as a consequence of interference with another train. This delay is a function of the type of train movement control, the length and location of sidings, and whether or not some or all of the district is doubled-tracked.

Trains are introduced into the system by defining a route through the network as a sequence of district couples (i, j) denoting movement from node i to node j . The route for the k^{th} train is denoted by the sequence of couples $J(k)$; e.g., if train 5 travels from node 3 to node 7 and then to node 5, $J(5) = [(3,7), (7,5)]$. Associated with each train is a total allowable schedule time for completing its route denoted by s_k for the k^{th} train, a departure time from the first node on the route denoted by d_k , and a total payload weight w_k . The schedule time is the total allowable elapsed time from the first node to the last. Departure time is measured on a 24-hour clock, as it is assumed that each train runs every day.

Since the objective is to find the optimal horsepower allocation for each train in each district, define the variables

- h_{ijk} as the horsepower per ton allocated to train k in its run over district (i, j) and
- v_{ijk} as the number of meets train k experiences in district (i, j) due to conflict with superior trains.

Let the functions

- $C_{ij}(\cdot)$ relate the cost incurred in district (i, j) to the horsepower per ton assigned to a given train in that district and
- $R_{ij}(\cdot)$ relate running time in district (i, j) to the horsepower per ton assigned to a given train in that district,

and define the constants

- m_{ij} the average delay time per meet in district (i, j) and
- f_{ijk} the fixed terminal delay train k experiences in node i before departing to traverse district (i, j) .

The independent or decision variables are the horsepower assignments $\{ h_{ijk} \}$, while the variables $\{ v_{ijk} \}$ representing the number of meets experienced by each train are dependent variables which are complex functions of horsepower assignments, functions and constants. In order to calculate optimal horsepower assignments, mathematical relationships must be derived to describe feasible values for the independent

variables and the resultant operating costs. A feasible assignment is one in which all trains traverse their routes in the scheduled times $\{s_k\}$, and the net power flow at any node is zero. The time it takes a train to complete a route is the sum of running time, fixed delay at terminals, and delays due to conflicts with other trains in districts on the route. We first describe the derivation of the running time and cost relationships, then turn to meet-delay calculations, and finally integrate the factors into a descriptive mathematical model and computational procedure.

Cost and run time functions

The operating cost model is an extension of Southern Pacific's formula for calculating direct and allocated costs of freight service. The formula includes estimates of direct costs (such as wages and fuel) which can be identified directly with particular trains. To these direct costs are added indirect costs, allocated on the basis of work done in moving a ton or carload of freight, or a train. While locomotive costs are the only category subject to optimization in the operating cost model, other costs (such as wages) are also considered in the model since certain operating alternatives may affect these costs. A detailed breakdown of locomotive costs is given by Figure 1.

The costs based on speed and load hauled are difficult to arrive at but crucial to the analysis and are calculated by applying unit costs to simulated fuel consumption data obtained from a computer simulation program.

developed by Canadian National Railway.* This program simulates the movement of a train with a given locomotive consist over a stretch of railroad, subject to specified speed limits. Elapsed time and fuel consumption of trains between successive points on the railroad are calculated by successive integration of the acceleration-velocity-displacement equations. Repeated runs of the simulation program were made at different levels of locomotive horsepower per ton of train weight to obtain function values of fuel consumption per ton of train weight vs. horsepower per ton, and of run time required to traverse the district *without* meet and other delays vs. horsepower per ton of train weight. The district running time vs. horsepower per ton is the function $R_{ij}(\cdot)$ and the fuel consumption per ton vs. horsepower per ton relationship is crucial to the calculation of $C_{ij}(\cdot)$, the cost per ton vs. horsepower per ton function.

The run times for intervals between sidings in a district were input to a short computer program which calculated the range of possible meet delays and the average delay for trains meeting each other within each district represented by the constants m_{ij} . The Canadian National simulation computer program thus was the important first step since it essentially provided the description of the railroad in terms of functions and constants to the operating cost model.

Delays due to meets

In order to calculate delays due to meets, we numbered the trains in order of decreasing priority and assumed that in a meet situation only the lower priority train suffers delay. For a specific train k and associated power allocation $\{h_{ijk}\}$, the time a_{ijk} that train k enters link (i, j) in its route is calculated as

$$a_{ijk} = S_k + \sum [R_{ij}(h_{ijk}) + f_{ijk} + v_{ijk} m_{ij}]$$

where the summation on the right is over all districts preceding (i, j) in the sequence $J(k)$ and addition is performed modulus 24 hours. Train k will experience a meet delay in district (i, j) if there is some superior train \bar{k} (i.e., $\bar{k} < k$) that travels district (j, i) and that enters the district while train k is in the district or vice versa. Formally, train k is delayed in district (i, j) whenever

$$a_{ijk} < a_{j\bar{k}} \leq a_{i\bar{k}} + R_{ij}(h_{ijk})$$

OR

$$a_{j\bar{k}} < a_{ijk} \leq a_{j\bar{k}} + R_{ji}(h_{j\bar{k}})$$

*Canadian National Railway, *Train Performance Calculator for the IBM-7070.*

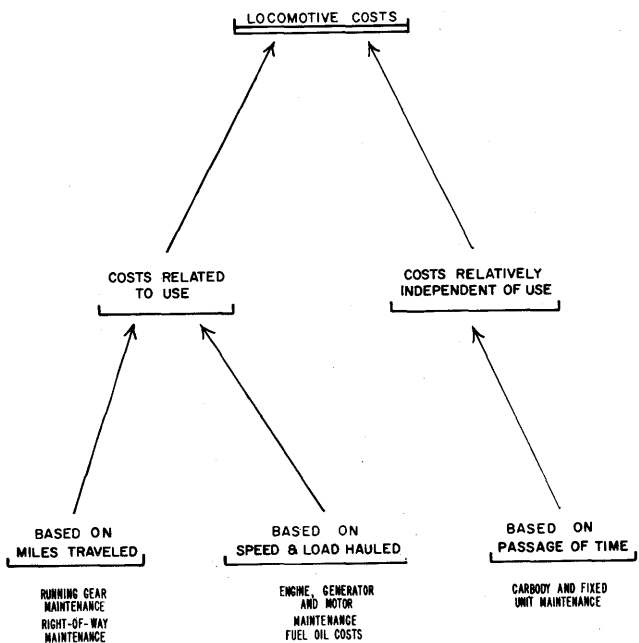


FIGURE 1

for some $\bar{k} < \hat{k}$, where appropriate care must be exercised when interpreting the inequalities for those trains spending portions of two consecutive days in a particular district. If the horsepower assignments $\{h_{ijk}\}$ are known for all trains, the above computations can be performed to determine the number of individual meets $\{v_{ijk}\}$ by starting with the first district traversed by train 2 (train 1 experiences no meet delay) and computing a_{ij2} and v_{ij2} for all $(i, j) \in J(2)$ in sequential order. The difficulty is that these horsepower assignments are the independent variables to be determined; consequently their optimal values are not known initially. This necessitates the recursive procedure described below under Method of Computation.

Horsepower assignment calculation

Assume that the number of meets $\{v_{ijk}\}$ encountered by each train in each district is known for the optimal horsepower assignments $\{h_{ijk}\}$. If K trains are to be run, the solution to the following mathematical programming problem determines the desired optimal horsepower values:

$$\text{Minimize } \sum_{k=1}^K \sum_{(i,j) \in J(k)} w_k C_{ij}(h_{ijk})$$

subject to:

- 1) $\sum_{(i,j) \in J(k)} [R_{ij}(h_{ijk}) + f_{ijk} + v_{ijk}m_{ij}] \leq S_k$
for $k = 1, \dots, K$
- 2) $\sum W_k h_{ijk} - \sum W_k h_{jik} = 0$
for each node i in the network
- 3) $h_{ijk} \geq 0$ for $k = 1, \dots, K$ and all $(i, j) \in J(k)$

where the first summation in equation group (2) is over all trains which depart node i and the second summation is over all trains which enter node i at some point on their route.

The objective function expresses the desire to find a minimum cost solution. Equation group (1) represents the constraint that all trains complete their routes within the scheduled times; equation group (2) requires that the net flow of horsepower into each node be zero; and group (3) restricts horsepower assignments to non-negative levels. Since the functions $R_{ij}(\cdot)$ are convex the feasible region represented by (1), (2), and (3) is convex. The problem of minimizing the convex function $C_{ij}(\cdot)$ subject to a convex constraint set is formally a problem in non-linear programming with separable

functions under the assumption that the values of the meet-delay variables $\{v_{ijk}\}$ are known. The computational approach and the scheme devised to circumvent the dependence of the meet delay times upon the actual horsepower assignment are discussed next.

Method of computation

Since the cost and running time functions are initially calculated from the train performance program as piecewise linear functions, the bounded variable method of linear programming is immediately suggested. Using horsepower increments of one half horsepower per ton with 10 increments describing each district, a problem with 75 trains and 40 districts results in an LP matrix with 93 equations and 1,674 unknowns. The LP code selected was MPS-360 supplied by IBM.

Since the size of the problems considered is large, an input pre-processor was required to prepare the data for MPS. Input to the pre-processor is the running time and fuel consumption data from the Canadian National train performance program, a network description, unit costs, train schedules, routes, and delay times. The operating cost function $C_{ij}(\cdot)$ is calculated and the following three output files are created: the first specifies the minimization problem in a form suitable for direct input to MPS, the second file contains running time information required in the meet delay calculation, and the third file contains data required by a post-processor for report generation. Final output of the system is a series of reports giving the optimal train horsepower assignments and cost per day to operate the specified system.

As we have seen, if the values of the optimal meet delay variables v_{ijk} are known, the horsepower optimization is a straightforward problem in mathematical programming. However, the optimal values of the meet delay variables are not known *a priori* which leads us to a recursive procedure in which a set of values is assumed for these variables; a horsepower optimization run using these values; and then a meet-delay calculation performed using the current optimal horsepower values to calculate new values for the meet delay variables. The process is then repeated unless the values of the meet delay variables are unaltered at which point an output report is produced.

The computations are initiated with the meet delay variables v_{ijk} set equal to zero. Optimization of this system by MPS proceeds rather quickly since the running time constraints are easily satisfied. The optimal horsepower values $\{h_{ijk}\}$ are then transmitted to a subroutine that performs the meet delay calculations, thereby determining new meet variables $\{v_{ijk}\}$. If the new values of the $\{v_{ijk}\}$ are the same as the old

ones, a consistent schedule has been found and the post-processor is called to write the output reports. Otherwise, the new values are transmitted back to MPS and a new optimization takes place with the calculations starting from the old optimal basis. This calculation is rapid since the old basis is still nearly optimal.

From a theoretical standpoint, there can be no guarantee that this procedure will terminate (cycling can occur) or that, upon termination, an optimal horsepower assignment will be found. In fact, very simple examples can be created that exhibit either of the above difficulties. The essential difficulty is two-fold; a moderate change in the $\{h_{ijk}\}$ may cause one or more meets to shift from one district to another, or, by powering the trains in an entirely different manner some meets could be shifted to locations where the delay time per meet m_{ij} is lower. Both of these problems arise only if the optimal solution is very sensitive to the times that the trains are departed from their original terminals. Since the model is being used for long range planning purposes, and since random problems influence daily operations, this type of sensitivity would be particularly objectionable. Fortunately, experiments with realistic data indicate that this sensitivity does not occur. In practice an upper limit is placed on the number of cycles allowed, and every solution is carefully examined to check on the reasonableness of the running times and the resulting congestion in each district.

The investment analysis program

The operating cost savings from alternative investment strategies become the input to the investment analysis model. In Section III we discuss our approach for comparing and ranking alternatives; here we briefly outline the structure of the investment analysis model.

To illustrate, again consider the Beaumont hill double track proposal. The operating cost model provides the operating cost incurred per day for various traffic levels with and without double track assumed on the Beaumont hill. These costs, together with the initial investment cost of the project and probability estimates on possible patterns of future traffic growth, are inputs to the investment analysis model which calculates cost savings per year for the input traffic points and then interpolates between these points to determine yearly cost savings for each of the assumed future traffic patterns.

The annual cost savings for each specified traffic pattern and the original investment cost are then used to calculate the investment's net present value for several discount rates. In this problem, a rate of return

on the investment is also calculated. Finally, a probability distribution based on the estimates of the likelihood of traffic patterns is calculated for net present value and rate of return.

Implementation

Even in the best environment, it is difficult to implement and gain management acceptance for a complex long range planning model such as described here. In essence, management is asked to make or approve detailed estimates of model inputs such as traffic growth forecasts, merger probabilities, and track improvement costs. The model then determines the "preferred" alternative through involved computer programs that management, for the most part, understands only in a general way. Because of this inherent problem, it is important to implement the models in a manner favorable to gaining management acceptance and use in decision making.

To this end, we found the following steps to be effective:

- A. Use an iterative approach to model development.
- B. Present the output in terms that management is accustomed to and/or easily understands.
- C. Sharply limit the alternatives evaluated by the model.

The following paragraphs discuss each of these points in greater detail.

Iterative approach to model formulation

The development of an interim model at an early date proved to be worthwhile—even though this model did not contain all the refinements required to make it fully useful in determining investment policies. The initial model's results and shortcomings were reviewed in detail with management, who in turn participated in refining its decision rules, parameters and constraints. This management involvement not only produced a more realistic model but also increased their understanding of how the model specifically handled the evaluation of various alternatives.

A second advantage of the iterative approach was the ability to gain insight into the relative importance of various constraints and parameters at an earlier date. For example, it was discovered that the initial operating cost model understated the indicated benefits from making various fixed plant improvements by not imposing the obvious practical constraint (equation group (2)) of forcing locomotive horsepower into and out of terminals to balance over a specified period. Benefits were understated because the initial model

permitted locomotive horsepower imbalances which were found to be much higher for the existing plant than for the improved plant, yet failed to attribute the reduced imbalance to the improvement. Later versions of the operating cost model include the locomotive horsepower balancing constraint, and we no longer had to explain the considerable manufacturing or scrapping of locomotives at terminals indicated by the earlier model.

Presentation of output in management understandable terms

The objective in formulating output was to display results in terms familiar to top management and still fully present both relative returns and relative risk of various investment alternatives. Two important questions had to be resolved. First, should the results be expressed in terms of discounted cash flow rate of return or as net present value at some specified interest or discount rate. Second, should utility (or preference) functions of management's attitude toward risk be incorporated in the model. These questions have been raised and discussed in recent literature although some doubt remains regarding their resolution.* Our discussion here is presented for the purpose of relating the general concepts of investment criteria and risk analysis to the particular environment in which this work was performed.

Net present value vs. return on investment

There were advantages to ranking the investment alternatives by their return on investment as calculated by the discounted cash flow method, including:

Southern Pacific management had been accustomed to examining investments in terms of return on investment.

It would not be necessary for management to decide upon an opportunity cost of capital.

On the other hand, there were disadvantages:

The magnitude of our investment alternatives varied widely and, as is well known, a small investment with a high rate of return may not be as attractive to management as a larger investment with a smaller, but still acceptable, rate of return.

Some of the investment alternatives require an initial outlay in the first year and additional net investments several years later. With this type

of investment pattern it is often impossible to calculate a single rate of return.

These disadvantages can be overcome by ranking the alternatives according to their net present value. However, in order to calculate the net present value of an alternative it is necessary to assume an interest rate at which to discount future cash flows. In our evaluations, we view the future benefits of proposed investments as being subject to reinvestment in other proposals that could not be undertaken without the cash flow from the investment currently under consideration. The rate at which benefits are realized from reinvestment of benefits from the initial investment becomes the discount rate, commonly called the opportunity cost of capital, used to evaluate the present value of an investment. The difficulty with this approach is our imprecise knowledge of the future opportunities and their associated returns.

The above dilemma is, as usual, resolved by compromise. We present the results both in terms of return on investment (when possible) and net present value for a range of plausible interest rates, which allows management to view both criteria.

Incorporation of relative risk

As reported in the literature previously cited, we found there was danger in ranking alternatives simply on the basis of expected rates of return or expected net present values. Since expected values are the weighted averages of all the conditional end state values, possibility of a large conditional loss or gain may be obscured. Yet, this is precisely the information management requires to evaluate the relative risk of the alternative. For example, assume that the expected rate of return for Investment A is 30 percent and for Investment B is 25 percent. However, depending upon the rate of growth of traffic volume there is a 30 percent chance in Investment A and only a one percent chance in Investment B of obtaining a negative rate of return. Depending upon management's attitude towards risk, the investment with the higher expected value may or may not be the best decision.

One standard method for weighing these relative risks is to develop a management utility function. A utility (or preference) function presents the differing values of money for varying profit or loss positions and could be mathematically incorporated in the investment analysis model.** While this approach is theoretically sound, we found it impractical to conduct the required interviews with Southern Pacific top management in

*The approach suggested by Hertz³ is most similar to the one recommended here. See also Grayson,¹ Schlaifer⁴ and Solomon⁵ for further discussion of these questions.

**See John S. Hammond.²

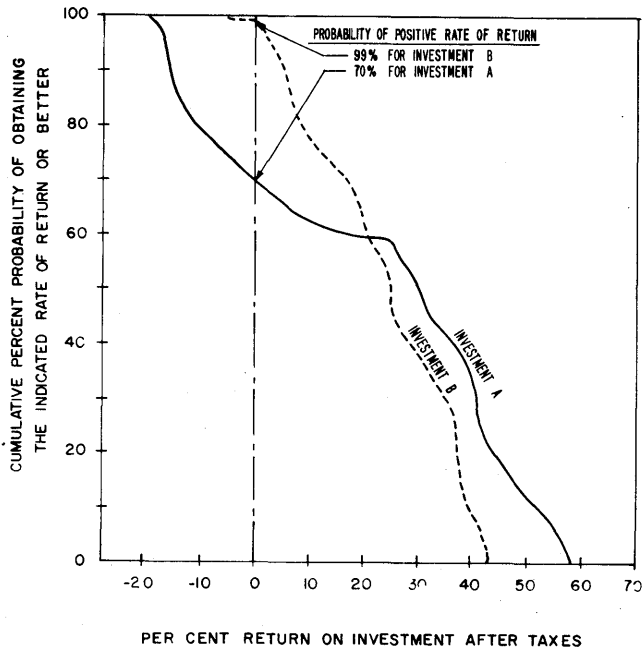


FIGURE 2

order to express their attitudes toward risk in precise mathematical terms. Instead, we display the results of our analysis by plotting cumulative probabilities of obtaining at least a given return or net present value. Figure 2 demonstrates how the rates of return for the hypothetical Investment A and B discussed above would be displayed. Managements' subjective judgment is again called into play to choose the set of investments with Probability-Return Profiles consistent with their preferences. The interviewing required to arrive at a decision in this manner is far less demanding than that required to construct an abstract utility function.

Limiting alternatives

The straightforward approach to determining which investment or set of investments is superior is to calculate incremental Probability-Return Profiles for each proposed improvement compared to the existing fixed plant, and to all other investments, based upon uncertainties in traffic growth and merger outcomes. If this approach had been followed the number of profiles to be calculated is $2^N - 1$ where N is the number of alternate fixed plant improvements. For example, over 32,000 profiles are required to fully evaluate all combinations of a set of fifteen alternative improvements. This would not only take a substantial block of computer time, but the proliferation of output would make a poor impression on management concerning the use-

fulness of computer modeling in long range planning and decision making.

In order to substantially reduce the number of investment comparisons, initial runs of the operating cost model were undertaken for the existing facility and for each of the alternative modifications to that facility for various traffic levels. The probability distributions of the incremental benefit (in terms of both rate of return and net present value) due to each proposal taken by itself were then obtained from a run of the investment analysis model. From this data an initial ranking was made under the assumption that only one project could be selected. At this point a large number of the proposed projects could be eliminated from further consideration since they were not judged desirable when compared with the existing plant and would be even less desirable when evaluated in conjunction with projects of higher initial rank.

After this initial pruning, evaluation of the investment alternatives proceeds iteratively comparing combinations of the initially attractive investments that are believed compatible from knowledge of the railroad's operation. Utilizing this approach in evaluating 15 alternative investments, the total number of comparisons was reduced to a manageable size of approximately 30 alternatives.

SUMMARY

We feel that a detailed analysis of long range investment alternatives is often indicated because of the large sums at stake and the risk inherent in building for an uncertain future environment. In many cases the future benefits derived from current investments are complex functions of a system's operation. One method for ascertaining benefits from this type of investment is to represent the system by a mathematical model which is then optimized for the possible spectrum of future environments. The model should be constructed and refined in a series of stages interspersed with meetings with management. These meetings are essential to achieve the desired rapport between decision maker and model builder. Results of the investigation should be presented in a concise form with emphasis on the relative risk of the various proposals. The computer is an essential partner in an investigation of this type as it is required to optimize the system model and to perform the myriad of computations required to account for future uncertainty and the combinatorial nature of the investment problem.

ACKNOWLEDGMENTS

We especially want to acknowledge the important and sizable contributions of Miss Beth P. Humphrey, who

programmed the operating cost model, Mr. Ralph S. Dippner, who programmed the investment analysis model, and Mr. Roy Carlson. All are members of Southern Pacific Company's Analytic Services Group.

Professor Ronald Howard of Stanford University suggested the original concept of the cost function, where costs are viewed not only as a function of traffic volumes, but also of the time commitments for each traffic flow.

REFERENCES

1 C J GRAYSON

Decisions under uncertainty: drilling decisions by oil and gas operators

Division of Research Harvard University Graduate School of Business Administration Boston 1960

2 J S HAMMOND

Better decisions with preference theory

Harvard Business Review November-December 1967 p 123

3 D B HERTZ

Risk analysis in capital investment

Harvard Business Review January-February 1964 p 95

4 R SCHLAIFER

Probability and statistics for business decisions

McGraw-Hill New York 1966

5 E SOLOMON

The management of corporate capital

The Free Press Glencoe Ill 1959

A computational model of verbal understanding*

by ROBERT F. SIMMONS, JOHN F. BURGER

and ROBERT M. SCHWARCZ

System Development Corporation
Santa Monica, California

INTRODUCTION AND BACKGROUND

The long-term goal for computational linguistics is to increase our understanding of linguistic and conceptual structures and to formally describe them so that computers can deal effectively with natural languages in such applications as question answering, stylistic and content analysis, essay writing, automated translation, etc. The eventual realization of this goal requires not only a satisfactory model of linguistic structures, but also models for verbal understanding and verbal meaning. In this paper we outline a theory and a model of verbal understanding and describe Protosynthex III, an experimental implementation of the model in the form of a general-purpose language processing system. The effectiveness of the model in representing the process of verbal understanding is demonstrated in terms of Protosynthex III's capability to disambiguate English sentences, to answer a range of English questions and to derive and generate meaning-preserving paraphrases.

Background

Computational linguistics is fortunately a field in which there is no dearth of state-of-the-art surveys. Over the last three years, Bobrow, Fraser and Quillian,¹ Kuno² and Simmons³ have independently reviewed recent relevant literature in structural linguistics, semantics, psycholinguistics and computer language processing. A critical survey is even now in press by Salton⁴ to cover most recent trends. A survey of question-answering systems by Simmons⁵ describes the earlier developments in that area.

*Much of the work reported in this paper was supported in part by United States Air Force, Air Force Systems Command, System Engineering Group under Contract Number F33615-67-C-1986, toward the development of a Natural Language Computer Aided Instructional System.

Several very recent lines of research by Quillian,^{6,7} Colby,⁸ Bohnert,⁹ Abelson¹⁰, Green and Raphael¹¹ and Simmons, *et al.*¹² have introduced ideas of deep logical and/or conceptual structures to represent understanding of phrases and sentences from natural language. Theoretical papers by Katz,¹³ Woods,¹⁴ and Schwarcz¹⁵ and experimental work by Kellogg^{16,17} have advanced our understanding of how to accomplish various forms of semantic analysis. Recent papers by Kay^{18,19} have been of great value in explicating and generalizing computational methods for syntactic analysis with particular reference to various forms of transformations.

These surveys and recent lines of research lead to the conclusions that the field of computational linguistics is a very active one, developing computational techniques at a rate that keeps pace with the advances in structural linguistic theory. Unfortunately, excepting for the Abelson and Colby models and cognitively oriented works by Miller *et al.*,²⁰ Deese²¹ and Reitman,²² there appears still to be a significant lack of psychological theory of verbal understanding to guide computational experimentation.

A representation of deep conceptual structure

Such operations as semantic analysis, question answering, paraphrase and mechanical translation each require the explication or transformation of concepts that are signaled or communicated by sentences in natural language. The concepts being communicated via language are not the words nor the phrases nor any other explicit structure of a discourse. Instead, what is being communicated is some set of relations among cognitive structures (i.e., ideas) that are held in common between a speaker and a hearer of the language. The linguistic notion of deep syntactic structure is a partial recognition of this fact, but for computers to demonstrate "verbal understanding" and manipulate "verbal mean-

ings," an even deeper level of conceptual structure must be represented. This deep conceptual structure serves as a partial model of verbal cognition, i.e., of how a human understands and generates meanings communicated by language. The effectiveness of a model of verbal understanding can be evaluated in terms of how well it supports such criterial operations as disambiguation, question answering, paraphrase, verbal analogies, etc. Whether the model truly represents the operations that humans actually use is another question and one to be studied by psychological experiment.

We thus define *verbal understanding* as the capability of a system to disambiguate, paraphrase, translate and answer questions in and from natural language expressions. *Verbal meaning* is defined as the set of interrelations in the model among linguistic, semantic and conceptual elements that provides this competence.

Our general model of understanding derives from a theory of structure proposed by Allport²³ in the context of psychological theories of perception. Our models also owe a conceptual debt to such widely varying sources as Chomsky's²⁴ theory of deep syntactic structure, Quillian's⁷ semantic nets and most recently, to Fillmore^{25,26} who proposes a significant variation to the Chomsky deep structure.

The primitive elements of our general model are *concepts* and *relations*. A concept is defined either as a primitive object in the system or as a concept-relation-concept (C-R-C) triple. In the model of verbal understanding, a concept that is a primitive object corresponds to a meaning or word sense for a word. But even these "primitives" can be defined as a structure of C-R-C triples that can be transformed to a verbal definition. A relation can also be either a primitive object or a C-R-C triple. Ideally, all relations should be primitive and well-defined by a set of properties such as transitivity, reflexivity, etc. Since each property corresponds to a rule of deductive inference, well-defined relations are most useful in making the inferences required for answering questions or solving verbal problems. Any relation, primitive or complex, can be defined in extension by the set of pairs of events that it connects. However, unless the relation is definable intensionally by a set of deductive properties, its use in inference procedures is generally limited to the substitution of equivalent alternate forms of expression.

Meaning in this system (as in Quillian's) is defined as the complete set of relations that link a concept to other concepts. Two concepts are *exactly equivalent* in meaning *only if* they have exactly the same set of relational connections to exactly the same set of concepts. From this definition it is obvious that no two nodes of the concept structure are likely to have precisely the same meaning. A concept is *equivalent* in meaning to another

if there exists a transformation rule with one concept as its left half and the other as its right. The degree of similarity of two concepts can be measured in terms of the number of relations to other concepts that they share in common. Two English statements are equivalent in meaning either if their cognitive representation in concept structure is identical, or if one can be transformed to the other by a set of meaning preserving transformations (i.e., inference rules) in the system.

English sentences can be mapped onto the deep conceptual structure of this model of verbal understanding by considering prepositions, conjunctions and verbs as relational terms, and nouns, adjectives and adverbs as conceptual objects. Thus, a sentence such as "The angry pitcher struck the careless batter" can be expressed in the following set of relational triples:

A. (((pitcher MOD angry)TMOD the) struck
((batter MOD careless) TMOD the))

As it stands, this is simply a form of syntactic diagramming of the sentence (where MOD and TMOD are modificational relations). However, by using the semantic analysis procedure to be described in a later section, the selection of word sense meanings is made explicit as follows (SUP means "has as a semantic superclass"):

B. (((pitcher SUP player) MOD (angry SUP emotion) TMOD the)

(struck SUP hit)

((batter SUP player) MOD (careless SUP attitude) TMOD the)).

The particular sense of "pitcher" is the one that is "a kind of player"; the sense of "strike" is "to hit" and the sense for "batter" is "player". The complex element (struck SUP hit) is the relational term for the larger triple

((pitcher, etc.) (struck SUP hit) (batter, etc.)).

When the triple structure B is embedded in the conceptual model, it can be roughly represented by the graph of Figure 1.

The result of embedding the sentence in the conceptual structure is to make explicit many aspects of verbal meaning that were implicit in the selection and ordering of words in the English sentence. Without any analysis or context the example sentence would answer only the question "Is it true that the angry pitcher struck the careless batter?" With such a relational analysis and embedding in the conceptual structure a whole range of new questions can be answered—for example:

Is the pitcher a person?
 Is a batter a baseball player?
 Did a baseball player hit a person?
 Do persons have attitudes?
 etc.

However, Figure 1 is only an approximate representation of the actual conceptual structure. The subscripts on each word in Figure 1 represent the word sense and concept selection appropriate to the sentence. In the actual structure a concept number occurs for each word on the graph. Each unique sense of meaning for a word corresponds to exactly one concept number; but each concept number may map onto more than one word sense *and* onto a defining structure of concepts. For example, the words "young" and "youthful" share a sense meaning in common, viz., "having the characteristics of youth." In each case this sense meaning corresponds to a concept number, say C72. C72 might be defined by the structure (C72 EQUIV (CO C42 C55)), which translates into "C72 is equivalent to something having youth."

What the conceptual structure does is to allow word meanings to be represented by a single conceptual object but, at the same time, to allow a conceptual object to be expressible in many different verbal forms. The conceptual level is necessary for paraphrase and translation operations. For example, the English expressions "old man" and "ancient" and the French word "vieux" can all be expressions of a single concept which we will label C37. The structure (man_c MOD old_c)—where the subscripted "c" means the concept number—is a defining term for C37 which is one of the word senses for "ancient" and for the French word "vieux." When the semantic analysis system produces (man_c MOD old_c) it tests to discover whether the triple can be expressed, as in this case it can, as a single concept. In the generation system that concept, C37, can be expressed by any of its mappings onto word senses and thus onto words.

Assuming for the moment our assertion that the

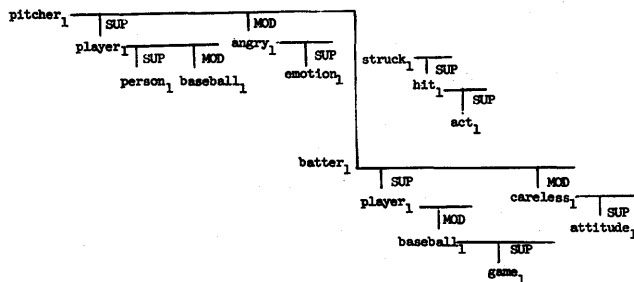


Figure 1—A graph of conceptual structure

model we have described does support the criterial operations of verbal understanding, the important question is: By what means can we transform English sentences into such a conceptual structure? The section immediately following, describes a method of syntactic and semantic analysis that accomplishes the transformation. A later section describes experiments to test the system's capabilities for disambiguation, paraphrase and question answering.

Analytic method

The method of analysis requires a lexicon, a grammar that includes transformations, a set of semantic event forms (SEFs), and a modified Cocke algorithm to actually carry out the analysis. In brief, the method finds immediate constituents of the surface structure of the sentence, transforms these into the form of deep conceptual triples and tests each such triple for semantic well-formedness. The resulting analysis is a bracketed structure of triples with each element marked for its selection of word sense meaning or concept. All analyses that are allowed by the grammar and SEFs are produced. A person operating the system is given the opportunity to select any one or several interpretations to be stored in the conceptual model.

The lexicon

The lexicon is composed of word and concept entries. With each English word entry, a set of word sense meanings is associated; each word sense, in turn, is associated with a syntactic class, a set of syntactic features, a chain of semantic word classes and a concept. For each concept entry, there may be a pointer to one or more word senses that may be used to express that concept verbally and an equivalence relation to one or more concept structures that represent its meaning. Some concepts, however, are not expressible as single word senses and are only verbally expressible by deriving the word senses for a concept structure to which they are equivalent. In addition to these elements, each concept entry has pointers to its tokens in the data structures where it has been used.

The semantic word classes that characterize each word are a chain of concepts that are in a linguistic superset relation. To explain by example, the word "pitcher" is characterized by two word senses and thus two different chains of semantic classes as follows:

- pitcher ... N, player, person, mammal
- ... N, container, physical object, object

The first superset chain (or SUP-chain) means that

“pitcher is a kind of player is a kind of person is a kind of mammal.” This is usually expressed as “pitcher SUP person...” Actually in place of the words for semantic classes, the lexicon contains concept numbers that usually refer to particular word senses. A more complete example of dictionary structure is presented in Figure 2.

A concept is created for the system for each new word sense and for each occasion when an equivalence relation occurs. Since every word sense can be defined by a dictionary definition that can be substituted in contexts where the word in that sense is used, it follows that every word sense concept is in an equivalence relation to some other concept structure that expresses its meaning. In the actual system, not every concept need be so defined, although the power of the system for verbal understanding obviously increases with the number of concepts that are defined.

The concept entry for the second sense of “strike” in Figure 2, i.e., concept number 55, would appear as follows:

Concept	Word	Senses	Meaning	Used/in
C55	208	(CO C89 C251)	G42, G45 . . . etc.	

This example shows that C55 may be expressed verbally by sense 208 that corresponds to the singular, present tense, verb “strike.” By looking up CO, C89 and C251, it can be discovered that the meaning of C55 can also be expressed by the words “to stop work.” The list of G-prefixed numbers in the Used/in column are simply pointers to data structures in which the concept C55 has been used to make factual statements.

The aim of this form of lexical structure is to distinguish clearly between linguistic and conceptual information. Syntactic classes and features* are defined as those elements which are required by the grammar and are clearly linguistic in nature. Semantic classes are expressed as concepts and are in a borderline area between the linguistic structure and the deep conceptual structure. Semantic classes are elements of the semantic event forms, but are also concepts that can occur anywhere in the deep conceptual structure.

The grammar

For discovering immediate syntactic constituents for a sentence and transforming them directly into the conceptual structure, we use a form of rule that combines phrase structure rewrite rules with a transformation.

*Although the lexical and conceptual structures provide for treatment of tense and agreement based on features, the analysis, generation and question-answering algorithms do not yet use this information.

	SENSE NBR	SYNTACTIC CLASS	SYNTACTIC FEATURES	SEMANTIC CLASSES	CONCEPT NBR
BOYCOTT	41	N	SING	123, 200	53
	42	V	PL, PR	123, 200	53
DISCOVERY	55	N	SING	98	67
DISCOVER	56	V	PL, PR	98	67
FIND	34	N	SING	---	98
	35	V	PL, PR	---	98
REBEL	150	V	PL, PR	---	123
	151	N	SING	---	124
STRIKE	207	N	SING	67, 98	100
	208	V	PL, PR	53, 123	55

ABBREVIATIONS: SING = Singular
PL = Plural
PR = Present Tense

Figure 2—A fragment of lexical structure

The form of this grammar can be understood by a simple example.

(a) $\text{adj} + \text{noun} \rightarrow (\text{B MOD A}) \text{NP} = (\text{NP} (\text{noun MOD adj}))$

The phrase structure component states that an adjective followed by a noun can be rewritten as a *Noun Phrase*. The transformation requires that the Bth or second element of the left side be written first followed by the term MOD followed by the Ath or first element.

A more complex example to account for a certain type of discontinuity is illustrated below.

(b) $\text{adv} + \text{S} \rightarrow (\text{BA} (\text{BB MOD A}) \text{BC}) \text{S}$
 $= (\text{S} (\text{Subject} (\text{verb phrase MOD adverb}) \text{object}))$

The transformation of (b) states that the BAth element of the left side is to be written first. The Bth element is S; S always breaks down into a triple whose Ath element is a noun phrase, whose Bth element is a verb phrase, and whose Cth element is an object noun phrase or an explicit null symbol. Thus the BAth element is the Ath element of the Bth element, or the subject of the S term. Similarly the BBth element is the verb phrase and the BCth element is whatever is in the object position.

A simple grammar to account for the sentence “the angry pitcher hit the careless batter” is presented in (c) below:

(c) $\text{adj} + \text{noun} \rightarrow (\text{B MOD A}) \text{NP}$
 $\text{art} + \text{NP} \rightarrow (\text{B TMOD A}) \text{NP}$
 $\text{verb} + \text{NP} \rightarrow (\text{D A B}) \text{VP}$
 $\text{NP} + \text{VP} \rightarrow (\text{A BB BC}) \text{S}$

The string of syntactic word classes corresponding to

these words is as follows:

art + adj + noun + verb + art + adj + noun

The analysis that results from this grammar is as follows:

(S (((noun MOD adj) TMOD art) verb ((noun MOD adj) TMOD art)))

In a previous paper [Simmons and Burger²⁷] we showed that this type of structure can be obtained by applying transformations to the elements of a phrase structure analysis of a sentence. That is precisely what the combined phrase structure and transformation rules of this type of grammar accomplish as each constituent of the sentence is discovered.

There is no theoretical limit to the depths to which the transformational notation can refer; strings such as ABBBCAB can be written to refer to the first, second, or third element of the nth level of the depth of structure. Certain elements of the transformations such as MOD, D (an explicit dummy marker) and the brackets are taken literally; only combinations of the terms A, B, and C refer to the structure of the left-hand side. The elements in a rule can be semantic classes on which the transformation can operate, and the resulting constituent can be a composition function of the semantic classes. For example, rules might be written to analyze the phrases "park bench" and "wooden bench" as follows:

place + furniture → (B LOC A) furniture-LOC
 material + furniture →(B TYPE A)
 furniture-TYPE

Compositions such as furniture-LOC imply a controlled combination of the SUP-chains for the two elements in a manner such as that described by Katz¹³ or used by Kellogg.¹⁶

Rules of this kind would eliminate the set of SEFs and the separate check for semantic acceptability. The disadvantage would be an enormous increase in the number of rules. Consequently, we have so far preferred to keep separate the syntactic and semantic components of the system.

Semantic event forms

As each constituent is discovered and transformed according to the grammar, the result of the transformation is tested for semantic well-formedness. The SUP-chain of semantic classes and a set of semantic event form (SEF) triples whose elements are semantic class terms are required for making this test. By considering

our example sentence again, the elements and method of this test can be explained.

A. The angry pitcher hit the careless batter. When "pitcher" was looked up in the lexicon, two word senses were discovered and both of these were nouns; for "angry" there was only one. Thus, two constituents of the form "adj + noun" were discovered to represent "angry pitcher." The SUP-chain of semantic classes that represented each sense of the words was then called into use to form the following pair of complex triples:

pitcher	MOD	angry
player		emotion
person		feeling
animal		sense
mammal		

and

pitcher	MOD	angry
container		emotion
physical obj		feeling
object		sense

Thus a complex triple is one whose elements are SUP-chains of the elements in a simple triple. From the total set of SEFs, the possibly relevant ones are those which contain one or more elements that are included in any of the complex triples of the sentence. This subset of SEFs include among others the following:

(ANIMAL MOD EMOTION)
 (PERSON MOD ATTITUDE)
 (PHYSOBJ MOD QUALITY)
 (PERSON HIT PERSON)
 (OBJECT HIT PERSON)
 (PERSON BOYCOTT ORGANIZATION)
 ETC.

The test for semantic well-formedness is to discover whether any triple of elements, selected one from each SUP-in a complex triple, corresponds to an SEF. In the present example, the combination (animal MOD emotion) from the first complex triple does correspond to an SEF in the list. No combination of elements from the second complex triple corresponds to an SEF, so the sense of "pitcher" as a "container" does not apply to constituent (N MOD Adj) for that sense and it is rejected. For the acceptable sense, "pitcher" as "person," the constituent is kept and elsewhere it is stored as ((pitcher SUP player) MOD (angry SUP emotion)). In subsequent constituents using this complex constituent, the SUP-chain of semantic classes for the head element

“pitcher SUP person” is used to stand for the entire constituent.

The result of these semantic tests is to reject many syntactic constituents that would otherwise lead to multiple interpretations of the sentence. For example, if we consider the number of common meanings for “pitcher,” “struck,” and “batter” to be respectively 2, 3, and 2 there would be 12 possible interpretations of the sentence. By the use of the three SEFs (ANIMAL MOD EMOTION) (PERSON MOD ATTITUDE) and (PERSON HIT PERSON) only the one interpretation presented below survives the analysis process.

```
((pitcher . person) MOD (angry . emotion))
      (struck. hit)
((batter  person) MOD (careless . attitude)))
```

The dot pairs are used for conciseness in representing (concept SUP concept).

It appears to us that an SEF is an abstraction of some element of lexical information that should (in a more sophisticated system) be directly a part of the lexicon. It appears to be an abstraction expressed in terms of semantic classes of the set of features that characterize a word's combinatorial possibilities in ordinary usage in the language. For example, the SEF (ANIMAL MOD SENSE) indicates the relationship expressed by linguists in terms of a restriction on sensory verbs and adjectives to co-occurrence with subjects marked by the feature “+ animate.” We believe, for the present state of computational linguistics, we can better represent such linguistic data in the form of acceptable combinations of semantic classes for words—i.e., SEFs—and later, from the useful SEFs, work out underlying features.

We have no answer to the question of how many SEFs would be required to cover a large subset of English. A related device, the semantic message forms [Wilks²⁸] are based on approximately fifty semantic classes and believed by CLRU* researchers to allow sufficient combinations to account for all English forms. We are currently tending toward the belief that although the separate SEF set provides adequate machinery for relatively small subsets of English, this information must eventually become an integral part of the lexicon to avoid very large space and time requirements in semantic analysis of large sets of English.

Selecting the level at which to write an SEF is hardly more easily dealt with. Considering each SEF as a rule of semantic combination, the task is very much like that of preparing a grammar. One attempts to obtain the

number of SEF rules that will distinguish acceptable and unacceptable combinations of word senses. The elements of each rule are selected at the highest level of semantic abstraction that will successfully distinguish all word senses that are in a superset relation to—i.e., subclasses of—those elements. Thus, in coining the SEF (ANIMAL MOD EMOTION) we are stating our understanding that the nature of these concepts is such that anything that is an emotion is restricted to modifying only those things that are animals. Similarly in (PERSON BOYCOTT ORGANIZATION) we restrict the concepts that are kinds of “boycott” to co-occurrence with things that are persons as subjects and things that are organizations to receive the action.

In favor of the SEF approach, we have found them simple to build and use and of the same functional utility as the semantic markers and selection restrictions of Katz's¹³ current semantic theory. Something approaching the function of his projection rules can be seen in our use of the semantic class of the head of a construction to stand for the semantic classes of the whole. However, we claim only that the SEF approach is a first approximation to expressing some parts of the semantic information that should be an integral part of a lexical entry for a word sense.

The analysis algorithm

After several experiments in producing various forms of recognition algorithms, we finally concluded that the Cocke algorithm was superior in respect to conciseness, completeness and efficiency of computation. This algorithm has been presented in ALGOL and described in detail by Kay.¹⁹ Our modifications have been only to add more tests on each constituent for agreement and semantic well-formedness and to introduce transformations into the operation of the grammar.

The essential operation of the algorithm is to test—exhaustively, but efficiently—each adjacent pair of elements in a sentence structure to discover if they form constituents acceptable to the grammar. If they do, the pair of constituents are rewritten according to the grammar rule. The process continues until all elements of the sentence are encompassed by at least one single constituent usually named S. All interpretations acceptable to the grammar are so formed.

Results

The complete language processing system that has been described has been programmed as Protosynthex III in LISP 1.5 for the SDC Q-32 time-shared computer. The semantic analysis system has also been programmed in JOVIAL and used to prepare Example 4

*Cambridge Language Research Unit, Cambridge, England.

and Figure 4. It includes the capability to syntactically and semantically analyze single sentences into the formal language of the conceptual structure. From the resulting conceptual structure, the system is able to answer a range of English questions using logical inference procedures based on properties associated with the well-defined relations. It is also able to paraphrase by finding equivalence relations among concepts and to generate English sentences in accordance with a generation grammar. In this paper a limited set of examples of these operations will be presented; additional computer printouts of examples have been collected as a special supplement that is available on request from the authors.

Syntactic and semantic analysis

The grammar reproduced in Figure 3 has proved suf-

ficient to account for the analysis of the sentence in the following paragraph about physiological psychology of the eye:

The eye is the organ of sight. The retina is the light sensitive surface of the eye. Cones and rods are special sensors in the retina. Cones and rods react to light. When we see anything, we see light reflected from the objects we look at. Reflected light passes through the lens and falls on the retina of the eye. Seeing an object actually means seeing the reaction of our retina.

The sentences comprising this paragraph were selected to represent a range of fairly difficult structures including various kinds of embeddings.

Example 1, below, shows the manner of inputting the sentence and dictionary data into the system.

```

ANALYSIS MODE //

READY--
THE EYE IS THE ORGAN OF SIGHT . . BODYPRT FUNCTION VISION ,
  ORGAN EQUIV BODYPRT .
  (SUPS- THE EYE IS THE ORGAN OF SIGHT)
  DDET ORGAN EQUIV DDET BODYPRT FUNCTION VISION
  (SUPS- VISION FUNCTION BODYPRT DDET ORGAN DDET)
  COGACT ASSOC OBJECT *TOP BODYPRT *TOP
  (SUPS- BODYPRT OBJECT ASSOC COGACT)
  OBJECT *TOP *TOP ACT
  (SJPS- ACT OBJECT)
  DO *TOP
  (SJPS- DO)
  *TOP
  (WCS- (SIGHT . VISION)
    (OF . FUNCTION)
    (ORGAN . BODYPRT)
    (THE . DDET) (IS . EQJIV) (EYE . ORGAN) (THE . DDET))
  NP PREP NP DART VBE NP DART
1
P
(((EYE . ORGAN) TMOD (THE . DDET))
  (EQJIV . PRIMIT)
  (((ORGAN . BODYPRT) (OF . FUNCTION) (SIGHT . VISION))
    TMOD (THE . DDET)))

```

Example 1.

A sentence is typed in followed by a period. Optionally a set of supersets for each word of the sentence can then be input followed by a period. Following this second period, SEF triples can be given to the system as

was done in Example 1. The third period—i.e., the one following the SEFs—is taken by the system to mean completion of input. At that point the system looks up each word in the dictionary to obtain superset classes

```

((QPP QBEAJ) [BA BB (BC AB AC)] S)
((DVBE ADJ) (AC MOD B) QBEAJ)
((VBEG NP) [BA BB BC] NP)
((NPN DVBE) [(AA EQUIV BC) AB (AC EQUIV BC)] S)
((VDO S) [BA BB BC] S)
((VDO SI) [BA BB BC] S)
((QND S) [BA BB AA] S)
((QND SI) [BA BB AA] S)
((QN DVBE) [BC EQUIV A] S)
((QN VDO) (A B D) QND)
((QAVDO S) [BA (BB MOD AA) BC] S)
((QAV VDO) (A B D) QAVDO)
((ADJ PP) (A BB BC) ADJ)
((ADJ NP) (B MOD A) NP)
((VCOMP PRIVCOMP) (A BB BC) CONJVCOMP)
((CONJ NP) (D A B) NCOMP)
((CONJ VCOMP) (D A B) PRIVCOMP)
((PREP QN) (D A B) QPP)
((PREP DNP) (D A B) PP)
((PREP NP) (D A B) PP)
((RELABV CONJS) [BA A BC] S)
((S CONJSPT) [A BB BC] CONJS)
((COM SI) (D A B) CONJSPT)
((COM S) (D A B) CONJSPT)
((VED NP) (B SMOD [*OBJECT A B]) NP)
((VED PP) (A BB BC) VCOMPED)
((AKI NP) (B TMOD A) NP)
((NP NCOMP) (A BB BC) NNP)
((NP V) [A B ****] S)
((NP VCOMP) [A B ****] S)
((NP PRED) [A BB BC] S)
((NP VPREP) [A B ****] SPREP)
((NP SPREP) (A SMOD [BA (EBA BBB A) BC]) NP)
((NP CONJVCOMP) [[A BA ****] BB [A BC ****]] CONJS)
((NP PP) (A BB BC) NP)
((NP VCOMPED) (A SMOD [*OBJECT B A]) NP)
((NP CONJSPT) [A BB BC] CONJS)
((V PP) (A BB BC) VCOMP)
((V DNP) (D A B) PRED)
((V NP) (D A B) PRED)
((V PREP) (A B D) VPREP)
((DNP DVBE) [A EQUIV BC] S)
((DNP PRED) [A BB BC] S)
((VBE RELPR) (D A B) VBEG)
((VBE DNP) (D A B) VBEG)
((VBE NP) (D A B) PRED)
((IAKI NP) (B TMOD A) DNP)
((SEQUIV **) (D A B) H*SEW)
((** H*SEW) [A BB BC] S)
NIL

```

Figure 3—Recognition grammar rules

and syntactic word classes. If it does find these, it requests SUPS for each word that is not in the dictionary.

It reiterates this request for semantic word classes until each word has developed a SUP-chain that terminates with the symbol, *TOP. It then asks for the syntactic word-classes with the request "WCS-" identifying the word sense by using the dot pair (word . superclass). When all these data are present—either already in the dictionary or having been input with the sentence—the system computes its semantic analysis using the grammar and SEFs available to it. In Example 1, the bracketed structure shows the syntactic analysis and the selection of word senses for each word in the sentence. This example shows that the system correctly transformed "is" in the context "The NP is the NP" into the well-defined relation "EQUIV." The relation "TMOD" is used by the system to alert it to the presence of an article. If the article is definite it refers to a particular or already existing token of data; if the article is indefinite or absent it is understood to represent any token or instance of its concept.

The most complex sentence of the paragraph is presented as Example 2, below. The analysis of this sentence shows four embedded sentences each of which is surrounded by square brackets. The first of these, "We see light" is in an IMPLY relation to the remainder. The expression "... light reflected from ..." gives rise to a noun phrase that is modified by the sentence "*object reflected ... light," where "*object" stands for "something." The phrase "... from objects we look at" gives rise to the structure (object SMOD [we (look at objects) ****]), a noun modified by an intransitive sentence that uses that noun as the object of a preposition. By following syntactic word class pairs through the grammar of Figure 3, the interested reader can observe the application of relatively simple transformations to compute these structures.

Examples of the analysis of question structures are shown in Example 3, below. In some cases the question

word is deleted while in others the question is transformed to declarative structure.

```

READY--
DO WE SEE ANYTHING ?
1
P
[(WE . PERSON) (SEE . COGACT) (ANYTHING . OBJ)]
1

READY--
WHAT DO WE SEE ?
1
P
[(WE . PERSON) (SEE . COGACT) (WHAT . OBJ)]
1
EXIT

WHEN WE SEE ANYTHING , WHAT DO WE SEE ?
(SUPS- WHEN ANYTHING ,)
RELAVB OBJ
COMMA
(SUPS- COMMA RELAVB)
DO DO
(WCS- (, . COMMA) (ANYTHING . OBJ) (WHEN . RELAVB))
COM NP RELAVB
1

P
[[ (WE . PERSON) (SEE . COGACT) (ANYTHING . OBJ) ]
 (WHEN . RELAVB)
 [(WE . PERSON) (SEE . COGACT) (WHAT . OBJ) ] ]
1
READY--
HOW DOES THE OBJECT REFLECT LIGHT ?
(SUPS- HOW DOES REFLECT)
MANNER DO THPO"
MANNER DO THROWBACK
(SUPS- MANNER)
VQJAL
(WCS- (REFLECT . THROWBACK) (OBJECT . OBJ) (DOES . DO) (HOW MANNER))
V NP WDO QAV
1

P
[ ((OBJECT . OBJ) TMOD (THE . DET))
 ((REFLECT . THROWBACK) MOD (HOW . MANNER))
 (LIGHT . RADIATION) ]
1

```

Example 3.

Disambiguation

An example deriving from Katz¹⁸ was chosen to illustrate the system's ability to select correct word senses from a potentially ambiguous situation. The example frame, "The man *hit* the colorful ball" is varied by substituting "gave," "attended" and "hit" in the verb slot. The relevant dictionary, grammar and SEF entries

are presented in Figure 4. Since the dictionary provides two senses each for "colorful," "ball" and "gave," in the worst case the frame using "gave" might provide eight interpretations (as it in fact did without SEF checking). With the use of the relevant SEFs, the system provided the interpretations shown in Example 4. The two interpretations for "...gave a colorful ball" are

expected in that SEFs are allowed for "person present object" and "person present event." In the remaining

cases of "hit" and "attend" only one interpretation was obtained.

ANALYSIS

ANALYSIS MODE

THE MAN GAVE A COLORFUL BALL.

2 INTERPRETATIONS

1
 [(MAN TMOD THE) <GAVE - PRESENT>
 ((<BALL - SPHERE> MOD <COLORFUL - MULTICOLORED>) TMOD A)]

NEXT/FINISHED/RULES/SEF'S? NEXT

2
 [(MAN TMOD THE) <GAVE - PRESENT>
 ((<BALL - DANCE> MOD <COLORFUL - GAY>) TMOD A)]

NEXT/FINISHED/RULES/SEF'S? FINISHED

THE MAN ATTENDED A COLORFUL BALL.

1 INTERPRETATION

1
 [(MAN TMOD THE) ATTENDED
 ((<BALL - DANCE> MOD <COLORFUL - GAY>) TMOD A)]

NEXT/FINISHED/RULES/SEF'S? FINISHED

THE MAN HIT A COLORFUL BALL.

1 INTERPRETATION

1
 [(MAN TMOD THE) HIT
 ((<BALL - SPHERE> MOD <(OLORFUL - MULTICOLORED>) TMOD A)]

Example 4.

Disambiguation Example

Answering questions

Our approach to answering questions in this system is described briefly in Simmons and Silberman.²⁹ A more detailed description of the question-answering system and experiments with it is in preparation [Schwarcz *et al.*, 1968]. Briefly, the system attempts a direct match with the concept structure of each triple resulting from the semantic analysis of a question. Failing to find a direct match, it generalizes each element of a question

triple to include all of its equivalences and subclass elements. Thus a question triple with the element "bird" would generalize to include "condors, robins, bluebirds, etc." This approach failing, the system uses more complicated inferences based on combinations of relations into compound and complex relational products.

We believe the approach is very general and approximately equivalent to that taken in the General Problem Solver [Newell *et al.*³⁰]. A top-down generator is used

```

LOOKUP THE MAN GAVE ATTENDED HIT A COLORFUL BALL.
THE: FUNCTION WORD, 1 SENSE:
  1 ART      DET
MAN: FUNCTION WORD, 1 SENSE:
  1 NP      PERSON, ANIMAL, OBJ
GAVE: FUNCTION WORD, 2 SENSES:
  1 V      PRESENT, OFFER, ACT
  2 V      TRANSFER, MOVE, ACT
ATTENDED: FUNCTION WORD, 1 SENSE:
  1 V      GOTO, MOVE, ACT
HIT: FUNCTION WORD, 1 SENSE:
  1 V      CONTACT, ACT
A: FUNCTION WORD, 1 SENSE:
  1 ART      DET
COLORFUL: FUNCTION WORD, 2 SENSES:
  1 ADJ      MULTICOLORED, BRIGHT, QUAL
  2 ADJ      GAY, LIVELY, QUAL
BALL: FUNCTION WORD, 2 SENSES:
  1 NP      SPHERE, OBJ
  2 NP      DANCE, EVENT

```

PRINT RULES.

```

ADJ NP (B MOD A) NP
ART NP (B TMOD A) NP
V NP (D A B) PRED
NP PRED (A BB B C) S

```

OK.

SEFS.

```

(PERSON PRESENT EVENT)
(PERSON PRESENT OBJ)
(PERSON GOTO EVENT)
(PERSON CONTACT OBJ)
(EVENT MOD GAY)
(OBJ MOD MULTICOLORED)

```

Figure 4—Dictionary, grammar, and SEF entries for disambiguation example

to transform question triples according to possible inference rules and various heuristics are followed to minimize tree search among the resulting vast set of possibilities.

For example we asked questions of the following sentences:

The eye is the organ of sight. The retina is the light-sensitive surface of the eye. Cones and rods are the special sensors in the retina. Cones and rods react to light. These sentences were analyzed with the grammar of Figure 3 and appropriate SEFs to limit the interpretations to one per sentence. Questions were asked and the system answered as in the following examples:

- Example 5. What is the eye?
Eye is organ of sight.
Organ of sight.
- Example 6. What is the function of the eye?
Sight.
- Example 7. What is the surface of the eye?
Surface of eye be light-sensitive surface.
Surface of eye be retina.
Light-sensitive surface of eye be retina.

Example 8. To what is the retina sensitive?
Light-sensitive to light surface.

Example 9. What are the sensors in the eye?
Sensors in eye be sensors.
Sensors in eye be cones.
Sensors in eye be rods.

Example 10. Are there rods in the eye?
Sensors in eye.

Example 11. Does the eye contain cones?
Retina inverse-of-in sensors.
Sensors in eye be sensors.
Sensors in eye be cones.
Sensors in eye be rods.

Example 12. How does the eye react to light?
Cones and rods in retina in eye react to light.
Rods react to light.
Cones and rods react to light.

Example 5 is a result of direct lookup. The correspondence of "function" to "sight" in Example 6 results from the SUP-chain "sight-cogact-function" showing that sight or any other cognitive act is a kind of a function. In this example also, the structure (eye EQUIV (organ ASSOC sight)) implies (eye ASSOC sight) by right-collapsibility* of the "EQUIV" relation. The relation "ASSOC" is defined as symmetric and thus the question transforms to (eye ASSOC function) which is answered by (eye ASSOC sight).

Example 7 is essentially a direct lookup that is successful because of the symmetric property of EQUIV that allows the reversal of the clauses. The answer to Example 8 depends on an additional fact given to the system, "light-sensitive means sensitive to light." With this added information the question which was analyzed to the following structure:

(retina MOD (sensitive TO what))

is directly answered by the structure:

retina EQUIV (surface MOD (sensitive TO light)).

Example 9 and Example 10 require a chain of inference depending on the property transitive attached to (in contained-in). Thus, "sensors contained-in retina," and "retina contained-in eye" imply "sensors contained-in eye." In Example 11 a similar logic applies with the addition of the information that "contained-in inverse contained."

*The property right-collapsible is defined for R1 as follows:
(X1 R1 (X2 R2 X3)) IMPLIES (X1 R2 X3).

Example 12 shows one method for treating simple "how" questions. By analyzing the question into the statement "eye reacts to light" the system naturally returns relevant material, which is one main requirement of such questions. It should be noticed, incidentally, that the transitivity of the contained-in relation is used in this example.

The question-answering system has two important weaknesses. First, we do not yet formally distinguish between the requested operation (i.e., count, list, name, etc.) and the data-identifying portions of the question. This lack partially accounts for the second weakness—a certain degree of vagueness in the generated answers, as can be seen in Examples 8 and 10, where appropriate answers would have been "light" and "yes" respectively. Syntactic and semantic inadequacies in the generation of answers will be discussed in the following section.

Syntactic generation and lexical paraphrase

Our primary emphases in developing a theory of verbal understanding have been to account for the recognition of verbal meanings as communicated by sentences and to demonstrate understanding by the model's ability to answer English questions. Other measures of understanding include the capabilities for syntactic and lexical paraphrase and for the generation of new sentences that are in controllable relations to the data that have been stored as a consequence of understanding meanings from sentences that have been analyzed. It was our hypothesis that the structure for recognition and question answering would prove largely sufficient for generation and paraphrase. In the main this hypothesis was supported and we added the generation grammar and the special machinery for paraphrase to the model in short order. However, it is apparent that generation and paraphrase from deep conceptual structures require more theoretical explanation than we are prepared to deal with in this paper. Particularly required is an outline of correspondences with and contradictions of current generative linguistic theories. At this point, having only scratched the surface in experimenting with the generation area, we will present a brief discussion of our method and save detailed treatment for a later paper.

Generating English phrases or sentences from the conceptual structure is accomplished by the use of transformational phrase structure rules similar to those used in the analysis phase. Since the structure is composed of nested triples, these rules have the form of a three-element left half which is transformed to a structure or a string as a right half. Example rules for genera-

ting "the angry pitcher struck the careless batter" are shown below:

(NP MOD ADJ) (B A) NP
 (NP TMOD ART) (B A) NP
 (NP V NP) (A B C) S

The notation conventions are identical to those used in recognition rules. The generation algorithm, given a complex triple of concepts, first discovers for each concept its mapping onto word-sense and associated syntactic class and associates with each triple of concepts a triple of word-classes. Then, beginning with the most deeply nested triple and working outward, it looks up each triple of syntactic word-classes in the set of generation rules, and if found rewrites that triple by the phrase structure name and applies the transformation to the associated concept triple. This process is iterated until all elements in the nested structure have been accounted for (whether or not they result in the terminal symbol S). If a given triple can be rewritten in more than one way, the algorithm applies all rules, generating several syntactic paraphrases of the same structure. Thus, for the triple (NP MOD ADJ) the two strings "NP that is ADJ" and "ADJ NP" might result. At the end of the process the concepts are transformed into print images.

Lexical paraphrase is accomplished by allowing the free substitution of concepts that are in an equivalence relation. These concepts may map onto words or phrases. Thus in the examples presented below, "eye" is conceptually equivalent to "eyeball," and "organ of sight" is equivalent to "sensor for vision." An equivalence class is established by the statement "X SEQUIV Y" which is semantically analyzed like any other sentence except that SEQUIV is an operator used to construct an equivalence between concepts in the system.

The following examples illustrate both syntactic and lexical paraphrases accomplished by the system:

- Example 13. The eye is the organ of sight.
 The eye is the sensor for vision.
 The eyeball is the sensor for vision.
 The eyeball is the organ of sight.
- Example 14. The retina is the light-sensitive surface in the eye.
 The retina is the surface that is light-sensitive in the eyeball.
 The retina is the sensitive to light surface in the eye.
 The retina is the light-sensitive surface in the eyeball.

Example 15. Light falls on sensitive retina.
 Radiance falls on retina that is sensitive.
 Radiance falls on sensitive retina.
 Light falls on surface of eye that is retinal.
 Light falls on retinal surface of eye.
 Radiance falls on retinal surface of cranial orb.

No stylistic controls have so far been established to select either the generation transformation or the lexical item where several choices have been offered, and in expressing answers to questions no method has yet been developed for selecting a "best" answer. Such controls offer an entire field of study such as that currently in progress by Klein.* Our general procedure is also undeveloped with respect to choice of articles and the various forms of agreement in tense, number, etc. In respect to such syntactic features, the system makes provision for recording them, but we have not yet used them in any of our recognition, question answering or generation experiments.

Discussion and conclusions

In the preceding sections we have described a theory and a model of verbal understanding that is based on a formalization of conceptual structures sufficient to represent a wide range of the verbal meanings that are expressed in English sentences. The model includes a linguistic component that is composed of a lexicon and syntactic and semantic systems. The formal conceptual structure includes inference rules, a limited quantificational capability and a logical structure of relations that are definable by properties for use in inference procedures. These features of the model support a range of question answering and verbal problem solving capabilities.

The model has so far been limited to representing single sentence meanings, although the conceptual structure naturally embeds fragmentary meanings in their most relevant contexts. We do not believe, however, that a theory of sentence meanings is broad enough to encompass the communications mediated by natural languages. Related work in our laboratory by Olney²¹ has investigated anaphoric and discourse analysis to a degree that is sufficient to show us that complete understanding of a sentence can only be modeled in the context of its discourse structure. This line of research has also provided several workable approaches to finding antecedents for pronouns and other kinds of anaphoric structures. An important next step in the development

of the model will be to incorporate this line of thought and experimentation and so expand from a model of sentence understanding to one that represents understanding of larger discourse structures.

In the last section of this paper, we have briefly shown and discussed examples of the system's capability to produce syntactic and semantic analyses of sentences and questions; to select appropriate word senses according to context; to answer questions; to generate English sentences and to produce meaning-preserving paraphrases. We believe (but have not shown) that only minor modifications are required for the system to deal with a wide range of verbal analogy problems and to accomplish sentence-for-sentence translation. We claim that these results support the theory of verbal understanding outlined in an earlier section and demonstrate that this theory is adequate as a first approximation to account for how natural language sentences can communicate verbal meanings from one person (or system) to another.

The model's implementation as a LISP 1.5 program leaves much to be desired. It is slow and cumbersome in its operation and sharply limited in storage capability, having in its final version 11,000 words of free space. Yet only in LISP could we have tried so many variations of our original ideas until we were able to formulate them in terms of consistent workable programs. So on the one hand the system owes its existence to the facility with which complex ideas can be expressed in LISP while on the other, since sequential computers so poorly fit the requirements of large associative networks that LISP is well-suited to handle, the system is core-bound and painfully slow.

"Slow" means concretely that a typical sentence requires 90 seconds of compute time to analyze while an equivalent question requiring no great amount of inference may compute for three to four minutes. When these compute times are translated to wait-times on the time-shared system, analyzing and answering a question may take from fifteen to thirty minutes. Experimenting with such a system is obviously only tolerable to the most devoted believers in the eventual value of computer language processing.

In consequence a JOVIAL version of Protosynthex III, also for the Q-32 time-shared system, has been designed and already partly programmed.* So far the semantic analysis and generation systems are operating. This version has access to eight million words of disc storage. Its computing time for sentence analysis is gratifyingly reduced to tenths of seconds and its wait times on the time-shared system are typically within the turnaround time of 5-15 seconds. It is our current estimation that question answering with relatively short

*Personal communication, S. Klein, University of Wisconsin, Computer Sciences Department.

*Detailed design and programming by William J. Schoene.

chains of inference will be vastly shortened with respect to the LISP version.

CONCLUSION

We believe that the present system, Protosynthex III, demonstrates beyond question that sophisticated natural language processing by computers is a realistic goal and one that has been partly achieved here—although so far only on a sentence-by-sentence basis. We believe we have shown that with an appropriate lexicon, syntactic and semantic systems, that a wide range of English sentences can be translated with relative ease into formal structures that support logical operations of deduction and inference. It is further apparent to us that when a question and an answering text have been translated into the formal concept structure, question answering fits into the theorem proving and general problem solving models.

A reasonable goal, today, is to construct limited systems that deal in a limited manner with limited bodies of text. Such systems, programmed, require syntactic and semantic information in the form of dictionary entries, recognition and generation grammars, semantic event forms, and properties and rules of inference to define relations. All of these materials must be generated by skilled human users of the language and the system. For a single article such as this one, thousands of lexical entries and hundreds of syntactic and semantic rules would have to be produced.

We believe that this and other papers have demonstrated that natural language processing by computers is rapidly approaching a developmental phase in which the application of significant amounts of time and money can lead to eminently practical results. Significant improvements in automated translation, data base query systems, information and text retrieval, stylistic and content analysis can all be expected in the near future providing support is forthcoming for these efforts. This support will not only be required for the computer programming costs but also in equal or greater measure for the ancillary linguistic effort to formalize appropriate subsets of natural language.

REFERENCES

- 1 D G BOBROW J B FRASER M R QUILLIAN
Automated language processing
In Cuadra C A (ed) Annual Review of Information Science and Technology Interscience New York 1967 Vol 2
- 2 S KUNO
Computer analysis of natural languages.
Presented at Symposium on Mathematical Aspects of Computer Science American Mathematical Society New York April 5-7 1966
- 3 R F SIMMONS
Automated language processing
In Cuadra C A (ed) Annual Review of Information Science and Technology (Interscience New York 1966) Vol 1
- 4 G SALTON
Automated language processing
In Cuadra C A (ed) Annual Review of Information Science and Technology (Interscience New York 1966) Vol 1
- 5 R F SIMMONS
Answering english questions by computer a survey
Comm ACM
8 1 (1965) pp 53-70.
- 6 R QUILLIAN
Word concepts a theory and simulation of some basic semantic capabilities
Paper 79 CIT Pittsburgh Pa April 5 1965
- 7 R QUILLIAN
Semantic memory
PhD Thesis Carnegie Institute of Technology Pittsburgh Pa February 1966
- 8 K N COLBY
Computer simulation of change in personal belief systems
Behavioral Science (1967 in press)
- 9 H G BOHNERT P O BECKER
Automated English-to-logic translation in a simplified model
IBM (Thomas J Watson Research Center Yorktown Heights New York March 1966) AFOSR 66-1727 (AD-637 227) 117 pp
- 10 R P ABELSON J D CANOLL
Computer simulation of individual belief systems
The American Behavioral Scientist
9 24-30 (May 1965)
- 11 C C GREEN B RAPHAEL
Research on intelligent question-answering system
Stanford Research Institute Scientific Report No 1 Menlo Park Calif May 1967
- 12 R F SIMMONS J F BURGER R E LONG
An approach toward answering English questions from text
Proceedings of the 1966 Fall Joint Computer Conference pp 357-363
- 13 J J KATZ
Recent issues in semantic theory
Foundations of Language 3 124-194 (1967)
- 14 W A WOODS JR
Semantic interpretation of English questions on structured data base
Mathematical Linguistics and Automatic Translation
Report NSF-17 The Computation Lab Harvard University
Cambridge Mass 1966 39 pp
- 15 R M SCHWARCZ
Steps toward a model of linguistic performance a preliminary sketch
The RAND Corporation Memorandum RM-5214-PR
Santa Monica Calif January 1967
- 16 C H KELLOGG
On-line translation of natural language questions into artificial language queries
SDC document SP-2827 April 28 1967 47 pp (a)
- 17 C H KELLOGG
CONVERSE—a system for the on-line description and retrieval of structured data using natural language
SDC document SP-2635 May 26 1967 pp Presented at IFIP/FID Conference on Mechanized Information Storage Retrieval and Dissemination, Rome Italy June 1967 (b)
- 18 M KAY
The tabular parser a parsing program for phrase structure and dependency

- The RAND Corporation Memorandum RM-4933-PR Santa Monica Calif July 1966
- 19 M KAY
Experiments with a powerful parser
The RAND Corporation Memorandum RM-5452-PR Santa Monica Calif October 1967
- 20 G A MILLER E GLANTER K H PRIBRAM
Plans and the structure behavior
Holt New York 1960
- 21 J DEESE
Some contributions of psycholinguistic studies to content analysis
Paper presented to National Conference on Content Analysis Philadelphia November 16-18 1967
- 22 W R REITMAN
Cognition and thought; an information processing approach
John Wiley and Sons New York 1965
- 23 F H ALLPORT
Theories of perception and concept of structure
John Wiley and Sons New York 1955
- 24 N CHOMSKY
Aspects of the theory of syntax
Cambridge Mass MIT Press 1965 251 pp (Massachusetts Institute of Technology Research Lab of Electronics Special Technical Report No 11)
- 25 C J FILLMORE
A proposal concerning English prepositions
In Dineen FP (ed) Report of the Seventeenth Annual Round Table Meeting on Linguistics and Language Studies (Georgetown University Press Washington DC 1966) pp 19-34
- 26 C J FILLMORE
The case for case
The Ohio State University Columbus Ohio 1967
- 27 R F SIMMONS J F BURGER
A semantic analyzer for English sentences
SDC document SP-2987 January 10 1968 45 pp
- 28 Y WILKS
Computable semantic derivations
SDC document SP-3017 January 15 1968 160 pp
- 29 R F SIMMONS H F SILBERMAN
A plan for research toward computeraided instruction with natural English
SDC document TM-3623 August 21 1967 40 pp
- 30 A NEWELL H A SIMON
GPS; a program that simulates human thought
In Feignbaum E and Feldman J (eds) Computers and Thought (McGraw Hill New York 1963)
- 31 J C OLNEY D L LONDE
An analysis of English discourse structure with particular attention to anaphoric relationships
SDC document SP-2769 November 10 1967 10 pp

Procedural semantics for a question-answering machine †

by W. A. WOODS

Harvard University
Cambridge, Massachusetts

INTRODUCTION

Structure of a question-answering system

Simmons¹ has presented a survey of some fifteen experimental question-answering and related systems which have been constructed since 1959. These systems take input questions in natural English (subject to varying constraints) and attempt to answer the questions on the basis of a body of information, called the *data base*, which is stored inside the computer. This process can be conceptually divided into three phases—syntactic analysis, semantic analysis, and retrieval, as illustrated schematically in Figure 1. The first phase consists of parsing the input sentence into a structure which explicitly represents the grammatical relationships among the words of the sentence. Using this information the second component constructs a representation of the semantic content or “meaning” of the sentence.* The remaining phase consists of procedures for either retrieving the answer directly from the data base, or else deducing the answer from information contained in the data base. The dotted lines in the figure represent the possible use of feedback from the later stages to aid in parsing and semantic interpretation.**

†This research has been supported in part by the National Science Foundation under grants GN-554 and GS-2301 and by the Air Force Cambridge Research Laboratories under grant F-19628-68-C-0029.

*The second component of the diagram of Figure 1 is not always distinguished from the other two. As Simmons says, “The phase of a question-answering program devoted to semantic analysis is often not clear-cut and usually merges into the syntactic analysis and matching phases.”

**The only question-answering system to actually make use of such feedback is the DEACON system,³ which performs all three phases in parallel, so that when the parsing is complete, the question has already been interpreted and answered. It seems likely that in the human question-answerer, some such feedback

The objective of the research described here has been to develop a uniform framework for performing the semantic interpretation of English sentences. It was motivated by the fact that, although there exists a variety of formal parsing algorithms for computing the syntactic structure of sentences, the problem of using this information to compute their semantic content remains obscure. A question-answering system provides an excellent vehicle for such a study, because it forces consideration of semantics from the point of view of setting up a correspondence between the structures of a sentence and objects in some model of the world (i.e., the contents of the data base).

The initial phases of this research consisted of an evaluation of the semantics of existing question-answering systems and the design of the semantic interpreter described here. A more detailed discussion of these topics may be found in the author’s Ph.D. thesis.³ Subsequently, the semantic interpreter has been implemented in LISP on the Harvard Time-Sharing System and tested on a variety of sentences.

Limited deductive systems

The systems reviewed by Simmons comprise a wide variety of approaches to the question-answering problem. The structures used for the data base range from natural language text (bolstered by indices, concordances, and synonym dictionaries) to various sorts of hierarchies and networks of lists and sets of attribute-value pairs. Moreover, the methods which are used in the

does take place and that it results in a significant reduction in the number of ambiguous parsings and semantic interpretations which are generated. On the other hand, it is possible to consider the operation of an experimental question-answering system without this feedback by systematically generating all ambiguous parsings of the sentence, and applying the semantic interpreter to each of them. This latter approach has been taken in this study.

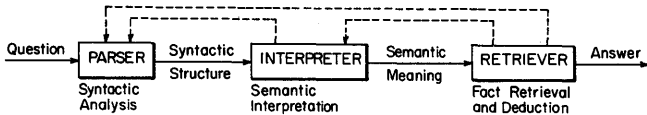


FIGURE 1—Basic organization of a question-answering system

three phases of processing are strongly influenced by the data structure chosen. In the extreme cases the syntactic analysis of English is done with a grammar whose constituents correspond not to grammatical categories of English but to structures in the data base. § Consequently, the techniques developed in these systems would not be expected to carry over to new types of data bases, and this indeed seems to be the case.

One common feature of existing question-answering systems is that they all deal with limited areas of discourse and limited subsets of English. An argument in favor of this approach is that by studying restricted models one discovers principles and techniques with wider applicability. In the area of fact retrieval and automatic deduction useful techniques do seem to be emerging—notably the use of list-structured data bases for rapidly locating data and performing simple types of deduction. However, the efficiency of these techniques seems to be directly related to the restricted nature of the problem area. Those systems which enjoy reasonable efficiency gain it by applying special purpose solutions that capitalize on special features of the restricted problem area. Examples of such systems are Bobrow's STUDENT,⁴ Raphael's SIR,⁵ and Lindsay's kinship relations program.⁶ Most of these systems gain their efficiency from special types of data structures and data base organization as well as special retrieval techniques.* On the other hand, attempts at completely general deductive systems, such as Black's Specific Question-Answerer⁷ have encountered excessively long search times as the size of the data base increases. It seems that for efficient processing, different sorts of data require different sorts of data structures.

Expanding the universe of discourse

A promising method for achieving reasonable efficiency in larger, less restricted universes of discourse is to provide the system with a variety of different types of data structures and special purpose deduction routines for different subdomains of the universe of discourse. Integrating a variety of special purpose routines

§This is true of the DEACON system for example.

*Bobrow explicitly points out the advantages of using "limited deductive models" that provide special facility for performing deductions in a restricted universe of discourse.

into a single system, however, requires a uniform syntactic and semantic framework. In general it is only after parsing and semantic interpretation have been carried out that such a system would be able to tell whether a sentence pertained to a given subdomain or not. Therefore, if the syntactic and semantic analyses were different for each subdomain, then the system would have to parse and interpret each sentence several times by different procedures in order to determine the appropriate subdomain. Moreover, there can be sentences that simultaneously deal with two or more subdomains, requiring a semantic framework in which phrases dealing with different subdomains can be combined.

Subsequent sections of this paper will describe a semantic interpretation procedure that can be used in an encyclopedic question-answering system hosting a multiplicity of special purpose deductive routines. This procedure will be illustrated by a semantic interpreter for a hypothetical question-answering system that answers questions about airline flight schedules using the *Official Airline Guide*⁸ as its data base. In this prototype system, a standard interface is imposed between the semantic interpreter and the retrieval component in order to achieve independence between the semantic interpretation procedure and the particular data structures and retrieval techniques employed in the data base. This interface takes the form of a formal query language that reflects the logical structure of English sentences rather than the structure of the data base. Unlike previous question-answering systems, the semantic interpretations of sentences in this system are procedures to be carried out rather than structures to be matched in the data base.

Semantics of question-answering systems**

Among the question-answering systems reviewed by Simmons, Bobrow's STUDENT provides the most insight into the semantic structure of a question-answering system. His "speaker's model of the world" contains the following components:

- (1) a set of objects $\{O_i\}$
- (2) a set of functions $\{F_i^x\}$
- (3) a set of relations $\{R_i^x\}$
- (4) a set of propositions $\{P_i\}$
- (5) a set of semantic deductive rules.

The semantic interpretation of an English statement is expressed in terms of these components as a set of rela-

**The notion of semantics presented here is quite different from the notion of semantics presented in Katz and Fodor's "Structure of a Semantic Theory." See the thesis for a comparison of the Katz and Fodor theory and that described here.

tions which are asserted to hold among specified objects. Functions carry ordered n-tuples of objects into objects (they need not be single valued). They are represented by linguistic forms such as "the number of ———," "the father of ———," "the sum of ——— and ———." The objects of the system are denoted by noun phrases—simple objects by simple nouns (e.g., "flights," "John," "Chicago," "3"), and functionally determined objects by composite nouns (e.g., "the number of flights," "the father of the boy," the sum of 3 and 4"). The relations are represented by verbs and their associated prepositional modifiers (e.g., "——— flies from ——— to ———," "——— hit ———," "——— equals ———"). The relations in the system correspond to the "concepts" which the system can "understand." The set of propositions $\{P_i\}$ are instances of the relations with particular objects filled in for the arguments. They correspond to the *beliefs* of the system, i.e., the relations which it knows to hold among particular objects (e.g., "AA-57 flies from Boston to Chicago," "the sum of 3 and 4 equals 7"). The semantic deductive rules are procedures for deducing new propositions or beliefs from the ones which are already held by the system. These components are essentially those of a formal system for the first-order functional calculus without the quantifiers and logical connectives.

Although Bobrow's system does not recognize quantifiers and recognizes the logical connectives only as separators that mark the boundaries of kernel sentences, question-answering systems in general will require all of these components as part of their semantic structure. Both quantifiers and logical connectives, however, can be viewed as predicates which take propositions as arguments and can be included in the above framework by simply relaxing the restriction that arguments of relations must be objects in the data base. Thus, we will permit arguments of predicates and functions to be filled by propositions as well as data base objects.† For example, "AND" can be represented as a predicate which takes two propositions as arguments and is true if both of its arguments are true. Likewise, the quantifier, "ALL" can be thought of as a predicate which takes one designator (a variable name) and one proposition (containing a free occurrence of the variable) as arguments. This quantifier ALL is true if the quantified proposition is true for every assignment to the variable.

There is one remaining component of the semantic structure which has been implicit in previous question-

answering systems but has not been explicitly recognized. This component is the set of commands that the system will carry out. In most systems there has been only one or at most two commands—"record X as data" and/or "answer X", and they have been left implicit rather than explicitly represented in the semantic interpretations of these systems. In our discussion, however, the semantic interpretations of sentences will explicitly represent commands to the system.

Semantic primitives

We will call the predicates, functions, and commands which a question-answering system "understands" the *semantic primitives* of the system, and we will say that a collection of semantic primitives is *adequate* for a given data base if, using them to ask questions, it is possible to reconstruct the data in the data base (i.e., everything in the data base is retrievable). This is the minimum requirement which a set of primitives must satisfy. Beyond this minimum requirement additional primitives may be added to improve efficiency or usefulness.

For unrestricted English, an adequate set of primitives would be large, but for a restricted area of discourse an adequate set of primitives may be quite small. An adequate set of primitives for the prototype system to interrogate the *Airline Guide* is listed in Table I.

It is assumed that the retrieval component contains a programmed subroutine or other operational procedure for each semantic primitive which defines the meaning of that primitive. Thus the predicate CONNECT, for example, will be defined by a programmed subroutine named CONNECT, which given three arguments X1, X2, and X3, determines the truth or falsity of the proposition CONNECT (X1, X2, X3). For the Airline Guide Flight Schedules Table, the operation of this subroutine would be to scan the table for the section which deals with flights to X3. It would then scan this section for the subsection which contains flights from X2 to X3, and finally it would scan this subsection to see if flight X1 is listed there. For a different organization of the data base, the program for CONNECT might be defined differently. For example, in another table of the Airline Guide, the Flight Itineraries Table, the computation would proceed by looking in the table under flight X1 to find the flight itinerary for flight X1 and then looking down that list for the place X2 and, if X2 is found, looking further to see if X3 occurs later in the list. Thus the definition of primitive concepts by programmed subroutines allows complete freedom from the particular organization of the data base. It even allows one to extend the system to concepts which are not explicitly contained in the data base

†Although such constructions are not handled by the present system, the decision to allow propositions as arguments of predicates paves the way for handling embedded sentences as in "Columbus thought that the world was round."

Primitive Commands.

TEST (D1) Test whether D1 is a true proposition or not.
LIST (X1) Print the name of the object X1.

Primitive Predicates.

CONNECT (X1, X2, X3)	Flight X1 goes from place X2 to place X3
DEPART (X1, X2)	Flight X1 leaves place X2
ARRIVE (X1, X2)	Flight X1 goes to place X2
DAY (X1, X2, X3)	Flight X1 leaves place X2 on day X3
IN (X1, X2)	Airport X1 is in city X2
SERVCLASS (X1, X2)	Flight X1 has service of class X2
MEALSERV (X1, X2)	Flight X1 has type X2 meal service
JET (X1)	Flight X1 is a jet
DAY (X1)	X1 is a day of the week (e.g. Monday)
TIME (X1)	X1 is a time (e.g. 4:00 p.m.)
FLIGHT (X1)	X1 is a flight (e.g. AA-57)
AIRLINE (X1)	X1 is an airline (e.g. American)
AIRPORT (X1)	X1 is an airport (e.g. JFK)
CITY (X1)	X1 is a city (e.g. Boston)
PLACE (X1)	X1 is an airport or a city
PLANE (X1)	X1 is a type of plane (e.g. DC-3)
MEAL (X1)	X1 is a type of meal service (e.g. breakfast)
CLASS (X1)	X1 is a class of service (e.g. first-class)
EQUAL (X1, X2)	X1 is the same as X2
GREATER (X1, X2)	X1 > X2 (Where X1 and X2 are times or numbers)
AND (S1, S2)	S1 and S2
OR (S1, S2)	S1 or S2
NOT (S1)	S1 is false
IFTHEN (S1, S2)	if S1 then S2

} (where S1 and S2 are propositions)

Primitive Functions.

TZ (X1)	the time zone of place X1
DTIME (X1, X2)	the departure time of flight X1 from place X2
ATIME (X1, X2)	the arrival time of flight X1 in place X2
NUMSTOPS (X1, X2, X3)	the number of stops of flight X1 between place X2 and place X3
OWNER (X1)	the airline which operates flight X1
EQUIP (X1)	the type of plane of flight X1
FARE (X1, X2, X3, X4)	the cost of an X3 type ticket from place X1 to place X2 with service of class X4 (e.g. the cost of a one-way ticket from Boston to Chicago with first-class service)

TABLE I—A set of semantic primitives for the flight schedules table

but can be deduced from it. For example, the Flight Schedules Table may not contain explicit information about connecting flights, but it is possible to define a subprogram which, given two flight names X1 and X2 and a city name X3, will determine whether flight X1 arrives in X3 and flight X2 leaves X3 and the arrival and departure times are such that X2 leaves within a certain specified tolerance (say at least $\frac{1}{2}$ hour and not more than 3 hours) after X1 arrives. This subprogram could define a primitive predicate *MAKESCONEX*(X1, X2, X3) meaning "Flight X1 makes connections with flight X2 at place X3."

The query language

Once a set of semantic primitives has been selected and an adequate query language has been defined, one can proceed to the design of the semantic interpreter

without concern for the data structures that will be used in the data base or the retrieval techniques that will be used. It suffices to assume that the retrieval component will contain procedures for determining the truth of any given predicate (given its arguments), for determining the value of a function (for given arguments), and for carrying out any of the primitive commands. Communication between the two components will take place in terms of the names of the primitive predicates, functions, and commands which the system understands.

The query language contains three basic types of construction—commands, propositions, and designators. At the top level, every expression in the query language will be a command that will in general contain propositions and/or designators as arguments. The prototype system contains two basic commands—*TEST* and *LIST*, where *TEST*(P1) is a command to determine

the truth-value of a proposition P1 and LIST(X1) is a command to print out the name of the object designated by a designator X1. A proposition consists of a predicate name followed by a list of its arguments enclosed in parentheses (e.g., DEPART(AA-57, BOSTON)), and a designator consists of a proper name (e.g., Boston), a variable name (e.g., X1), or a function name followed by a list of arguments (e.g., OWNER(AA-57)). Moreover, both propositions and commands may be quantified by quantifiers of the form:

(FOR QUANT X/CLASS : R(X) ; P(X))

where QUANT is a quantifier (EVERY, SOME, THE, etc.), X is the variable being quantified, CLASS is the name of a set over which quantification is to range, R(X) is a (possibly vacuous) further restriction on the range of quantification, and P(X) is the proposition or command being quantified. For example, (FOR EVERY X1/FLIGHT : DEPART(X1, BOSTON) ; LIST(X1)) is a quantified command which directs the retrieval component to print the name of every flight which leaves Boston. It is assumed that there are a relatively small number of sets that can take the place of CLASS in these quantifiers and that for each such set, the retrieval component contains a successor function which enumerates the members of that set.* More restricted ranges of quantification are obtained by imposing additional restrictions R(X). The reader is referred to the thesis for a discussion of the advantages of this formulation of the quantifiers.

In addition to the ordinary quantifiers SOME and EVERY and the quantifier THE, the prototype system contains two types of numerical quantifiers. The first type,

(FOR n MANY X/CLASS : R(X) ; P(X)),

is true if there are at least n members X in CLASS such that R(X) and P(X) are true. The second type allows the insertion of an arbitrary proposition to specify a condition on the number of objects—e.g.,

(FOR GREATER (N,30) MANY X/CLASS : R(X), P(X)).

Also there is a counting function, NUMBER (X/CLASS : R(X)), which returns as value the number of

*The successor function gives a pointer to the first member of the set when it is applied to a standard argument (e.g., 0). Subsequently, the successor function applied to a member of the set yields a pointer to the next member of the set. When applied to the last member, the function returns a recognizable end symbol (e.g., END).

objects X in CLASS for which R(X) is true. For a detailed description of the syntax of these expressions and their effects when carried out in a retrieval component, the reader is again referred to the thesis. Examples of query language expressions and their meanings are listed in Table II. The language is roughly equivalent to the first-order functional calculus, with somewhat more elaborate quantifiers and with the addition of commands, a concept which is not interpretable in systems of formal logic.

Semantic interpretation

Semantic rules

The task of the Semantic Interpreter is to “look at” an input phrase marker and to translate the phrase marker into a formal representation of its meaning in terms of the semantic primitives of the system. Both in human beings and in the machine this process must be finitely specifiable, even though the class of phrase markers to be interpreted and the class of interpretations to be produced are both infinite. Thus, the Interpreter must decompose the phrase marker into substructures which it can interpret directly, and it must compute the interpretation of larger structures as some

- (a) (FOR EVERY X1/FLIGHT: CONNECT(X1, BOSTON, CHICAGO); LIST (DTIME (X1, BOSTON)))
e.g. “At what times do flights leave Boston for Chicago?” or “List the departure time from Boston of every flight that goes from Boston to Chicago.”
- (b) TEST (CONNECT (AA-57, BOSTON, CHICAGO))
e.g. “Does AA-57 go from Boston to Chicago?”
- (c) LIST (NUMSTOPS (AA-57, BOSTON, CHICAGO))
e.g. “How many stops does AA-57 make between Boston and Chicago?”
- (d) LIST(NUMBER (X1/FLIGHT: CONNECT(X1, BOSTON, CHICAGO)))
e.g. “How many flights go from Boston to Chicago?”
- (e) TEST((FOR SOME X1/FLIGHT: CONNECT(X1, BOSTON, CHICAGO); EQUAL(OWNER(X1), AMERICAN)))
e.g. “Does American have a flight which goes from Boston to Chicago?”
- (f) TEST((FOR 30 MANY X1/FLIGHT: JET(X1); DEPART (X1, BOSTON)))
e.g. “Do 30 jet flights leave Boston?”
- (g) (FOR 5 MANY X1/FLIGHT: CONNECT(X1, BOSTON, CHICAGO); LIST (X1))
e.g. “Name 5 flights that go from Boston to Chicago.”
- (h) TEST ((FOR GREATER (N, 30) MANY X1/FLIGHT: JET (X1); DEPART (X1, BOSTON)))
e.g. “Do more than 30 jets leave Boston?”

TABLE II—Sample query language expressions.

sort of composite of the interpretations of its substructures.

In the prototype described here, the manner in which the semantic interpretation of a construction is built up from the semantic interpretations of its constituents is specified by a finite set of *semantic rules*. These rules are represented in a "pattern \Rightarrow action" format, where, the "pattern" is a description of a fragment of a phrase marker together with some conditions on the constituents of that fragment, and the "action" specifies the semantic interpretation of such a fragment in terms of the interpretations of its constituents. These rules have a kinship to the "pattern \Rightarrow substitution" rules of Post production systems and the rewrite rules of phrase-structure grammars, but the effect is somewhat different. The action in this case is not a substitution but an operation that attaches a semantic interpretation to the given fragment.

In the syntax tree of a sentence, there are basically two types of nodes that receive semantic interpretations—S nodes, corresponding to sentences and clauses, and NP nodes, corresponding to noun phrases. The semantic interpretations of the former are either propositions or commands, while the semantic interpretations of noun phrases are designators. Corresponding to these two types of nodes, there are two processors—the S-processor, for interpreting S nodes, and the NP-processor for interpreting NP nodes. Each of these processors attaches a semantic interpretation to the node of the syntax tree that it is interpreting. In addition, the NP-processor may also generate quantifiers that are to govern the sentence of which the NP is a constituent.

The S-processor

We may illustrate the operation of the S-processor independent of the NP-processor by considering sentences whose noun phrases are all proper nouns (which we assume to be directly interpretable). For example, consider the sentence:

AA-57 flies from Boston to Chicago.

A phrase marker for this sentence might look something like that of Figure 2.

Since verbs in English correspond roughly to predicates, and noun phrases are used to denote the arguments of the predicate, the verb in the structure diagram will be the primary factor in determining the primitive predicate (or composite of primitive predicates and functions) which represents the meaning of a sentence. For the example in Figure 2, the predicate will CONNECT as defined in Table 1. In addition to deter-

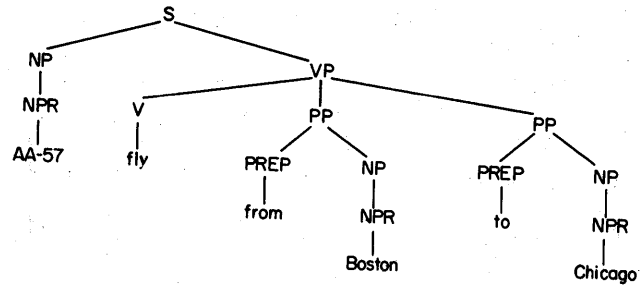


FIGURE 2—A sample phrase marker

mining the predicate, however, the S-processor must verify that all of the required arguments for the predicate are present in the structure diagram, that they meet appropriate conditions for the predicate to take them as arguments, and that they have the correct grammatical relationships with the verb. For example, for the predicate CONNECT it is necessary that the subject be a flight, and it is necessary that there be prepositional phrases whose objects are places representing origin (from) and destination (to). These tests are performed in order to rule out semantically bad interpretations and to choose the correct senses from among the various senses which a given verb may have. For example, the sentence:

Boston flies from AA-57 to Chicago

is semantically bad, while the verb "fly" in:

Can I fly from Boston to Chicago?

has a different sense from that of the sentence in Figure 2. Thus the S-processor must be able to ask questions about the constituents of a sentence (e.g., whether they are flights or places, etc.) and about the grammatical relationships among them (e.g., what is the subject, is there a prepositional phrase, what is the object of the prepositional phrase, etc.).

The grammatical relations among elements of a phrase marker are defined by partial tree structures which match subtrees of the phrase marker. A set of such partial structures is given in Figure 3. The numbers in parentheses in these partial trees are labels used to refer to the nodes to which they are connected. Thus, two nodes A and B in a phrase marker are in the subject-verb relation iff the partial tree G1 matches a subtree of the phrase marker in such a way that the noun phrase (1) matches node A and the verb (2) matches node B. Notice that this grammatical relation is defined with respect to the particular S node that matches the topmost node (or *root*) of the partial tree structure G1.

S1	1-(G1: FLIGHT ((1)) and ((2) = fly or (2) = depart or (2) = go) and 2-(G3: (1) = from and PLACE ((2))) and 3-(G3: (1) = to and PLACE ((2))) ⇒ CONNECT (1-1, 2-2, 3-2)	e.g.	"AA-57 flies from Boston to Chicago"	e.g.	"AA-57 gets into Chicago at 9:23 a.m."
S2	1-(G1: FLIGHT ((1)) and ((2) = leave or (2) = depart) and 2-(G3: (1) = from and PLACE ((2))) and 3-(G3: (1) = for and PLACE ((2))) ⇒ CONNECT (1-1, 2-2, 3-2)	e.g.	"AA-57 departs from Boston for Chicago"	e.g.	"AA-57 departs from Boston on Monday"
S3	1-(G1: FLIGHT ((1)) and (2) = leave) and 2-(G2: (1) = leave and PLACE ((2))) and 3-(G3: (1) = for and PLACE ((2))) ⇒ CONNECT (1-1, 2-2, 3-2)	e.g.	"AA-57 leaves Boston for Chicago"	e.g.	"AA-57 leaves Boston on Monday"
S4	1-(G1: FLIGHT ((1)) and ((2) = fly or (2) = leave or (2) = depart) and 2-(G3: (1) = from and PLACE ((2))) ⇒ DEPART (1-1, 2-2)	e.g.	"AA-57 departs from Boston"	e.g.	"JFK airport is in New York"
S5	1-(G1: FLIGHT ((1)) and (2) = leave) and 2-(G2: (1) = leave and PLACE ((2))) ⇒ DEPART (1-1, 2-2)	e.g.	"AA-57 leaves Boston"	S15	1-(G1: ((1) = they or (1) = people or (1) = one or (1) = I or (1) = you) and ((2) = have or (2) = get) and 2-(G2: ((1) = have or (1) = get) and MEAL ((2))) and 3-(G3: (1) = on and FLIGHT ((2))) ⇒ MEALSERV (3-2, 2-2)
S6	1-(G1: FLIGHT ((1)) and ((2) = fly or (2) = go) and 2-(G3: (1) = to and PLACE ((2))) ⇒ ARRIVE (1-1, 2-2)	e.g.	"AA-57 flies to Chicago"	e.g.	"You get breakfast on flight AA-57"
S7	1-(G1: FLIGHT ((1)) and (2) = arrive) and 2-(G3: ((1) = in or (1) = at) and PLACE ((2))) ⇒ ARRIVE (1-1, 2-2)	e.g.	"AA-57 arrives in Chicago"	S16	1-(G1: ((1) = they or (1) = you) and (2) = serve) and 2-(G2: (1) = serve and MEAL ((2))) and 3-(G3: (1) = on and FLIGHT ((2))) ⇒ MEALSERV (3-2, 2-2)
S8	1-(G1: FLIGHT ((1)) and ((2) = leave or (2) = depart) and 2-(G3: (1) = from and PLACE ((2))) and 3-(G3: (1) = at and TIME ((2))) ⇒ EQUAL (D'TIME (1-1, 2-2), 3-2)	e.g.	"AA-57 departs from Boston at 8:00 a.m."	e.g.	"They serve breakfast on flight AA-57"
S9	1-(G1: FLIGHT ((1)) and (2) = leave) and 2-(G2: (1) = leave and PLACE ((2))) and 3-(G3: (1) = at and TIME ((2))) ⇒ EQUAL (D'TIME (1-1, 2-2), 3-2)	e.g.	"AA-57 leaves Boston at 8:00 a.m."	S17	1-(G1: FLIGHT ((1)) and (2) = be) and 2-(G2: (1) = be and CLASS ((2))) ⇒ SERVCLASS (1-1, 2-2)
S10	1-(G1: FLIGHT ((1)) and (2) = arrive) and 2-(G3: ((1) = in or (1) = at) and PLACE ((2))) and 3-(G3: (1) = at and TIME ((2))) ⇒ EQUAL (A'TIME (1-1, 2-2), 3-2)	e.g.	"AA-57 arrives in Chicago at 9:20 a.m."	e.g.	"AA-57 is first-class"
S11	1-(G1: FLIGHT ((1)) and (2) = get) and 2-(G3: (1) = into and PLACE ((2))) and 3-(G3: (1) = at and TIME ((2))) ⇒ EQUAL (A'TIME (1-1, 2-2), 3-2)			S18	1-(G1: AIRLINE ((1)) and ((2) = have or (2) = own or (2) = operate or (2) = run) and 2-(G2: ((1) = have or (1) = own or (1) = operate or (1) = run) and FLIGHT ((2))) ⇒ EQUAL (OWNER (2-2), 1-1)
				e.g.	"American Airlines operates flight AA-57"
				S19	1-(G1: FLIGHT ((1)) and (2) = belong) and 2-(G3: (1) = to and AIRLINE ((2))) ⇒ EQUAL (OWNER (1-1), 2-2)
				e.g.	"AA-57 belongs to American Airlines"

TABLE III—A representative set of S-rules

CONNECT(AA-57, Boston, Chicago) AND DEPART (AA-57, Boston) AND ARRIVE(AA-57, Chicago).

This redundant information may simply be left alone since it does no harm other than to cut down on efficiency. In the implementation of the prototype, however, this redundancy is eliminated by an appropriate ordering of the semantic rules, and a procedure for

determining what rule to try next depending on the success or failure of the previous rule.

Notice that these rules are able to describe the close connections between verbs and the prepositions which they can take, as embodied in constructions such as "fly from X to Y", "leave from X for Y," "arrive in X." They are also capable of distinguishing the meanings of the preposition "at" in "arrive at Chicago" and "arrive at 4:00 p.m." Notice furthermore that although the number of semantic rules in a system may be quite large, only a small number of rules can apply for any given verb, so that for interpreting a given phrase marker we need actually consider only a small number of semantic rules. In the implementation of the prototype the dictionary contains entries for each verb and noun specifying the set of possible semantic rules that could apply to a construction with that word as its head. This entry also specifies the order in which the rules are to be tried, and this order may in general be conditional on the success or failure of previous rules, allowing considerable flexibility for optimizing the interpretation procedure.

Semantic features

Notice that in the semantic rules of Table III, the semantic conditions, FLIGHT((1)), Place ((2)), etc., are all single place predicates corresponding to membership in some class of objects—i.e., flights, airports, etc. Furthermore, the truth of such conditions can be deduced from the noun of the noun phrase, and the necessary information can be recorded in the dictionary entries for the nouns. That is, in the dictionary entry for a proper noun, we can record *semantic features* corresponding to the sets of which the named object is a member, e.g.:

AA-57/FLIGHT
Boston/CITY, PLACE
JFK/AIRPORT

In the dictionary entry for a common noun, we can record features corresponding to the sets of which the class denoted by the noun is a subset, e.g.:

flight/FLIGHT
plane/FLIGHT
city/CITY, PLACE
town/CITY, PLACE
place/PLACE
airport/AIRPORT, PLACE.

These set-membership conditions can then be tested by referring to the dictionary entry for the noun to see if the indicated set is listed there as a semantic feature. More complex semantic conditions could always be defined by programmed subroutines that the semantic interpreter may call (and this facility is available in the prototype), but there has not yet been any need for semantic conditions more complex than simple set-membership conditions.

The generation of quantifiers

When considering the semantic interpretation of sentences containing quantified nouns, one discovers that a recursive top-down method of analysis is preferable to a bottom to top analysis. This is due to the fact that the quantifier which results from the interpretation of a noun phrase must be attached to the S node which it governs. When processing from the bottom up, we encounter the quantified noun phrase before we know where the appropriate S node is. In the recursive process from the top down, however, we encounter the S node first, and while we are still "looking" at it, the S-processor calls the NP-processor for each NP that must be interpreted. The NP-processor, then, returns the appropriate quantifiers to the S-processor, which is still "looking" at the S node to which they are to be attached.

Specifically the S-processor begins a working string (which may actually be a tree instead of a string) that will ultimately contain the representation of the semantic meaning of the S node being processed. Initially this string consists only of a single symbol Δ . Each time the NP-processor is called to process a quantified noun, it tags the NP that it is processing with a variable name, and returns one of the quantifiers described earlier. This quantifier contains the symbol Δ indicating the position in the quantifier where the remaining semantic interpretation of the S node is to be inserted (e.g., (FOR SOME X1/FLIGHT:-; Δ)). The S-processor then replaces the symbol Δ in its working string by the quantifier which the NP-processor has returned (thus maintaining exactly one symbol Δ in the working string). When all of the NP's have been processed, the S-processor uses the S-rules to interpret the sentence in terms of the variable names which the NP-processor has tagged onto the NP's. It then replaces the symbol Δ in the working string with the interpretation specified by the S-rules, thus eliminating the symbol Δ from the working string, which now contains the complete analysis of the S node. Finally, the S-processor tags the S node with the interpretation which it has built up in its working string and exits.

As an example to illustrate the process just described,

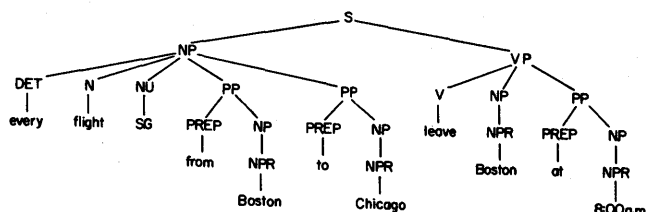


FIGURE 4—Phrase marker for a sentence containing a quantified noun phrase

before explaining the mechanism in detail, consider the sentence "Every flight from Boston to Chicago leaves Boston at 8:00 a.m.," which is diagrammed in Figure 4.* Looking at the topmost S, the S-processor starts a working string consisting only of the symbol Δ and then calls the NP-processor to process the NP "every flight from Boston to Chicago." The NP-processor creates a new variable X1 which has not been used before (e.g. by concatenating an integer to the right of the symbol X) and tags the NP with it. The NP-processor then (by mechanism to be discussed in the next section) returns the quantifier (FOR EVERY X1/FLIGHT: CONNECT(X1, Boston, Chicago); Δ) to the S-processor. The S-processor now substitutes this quantifier for the symbol Δ in its working string (giving a new working string which consists of just the quantifier (FOR EVERY X1/FLIGHT: CONNECT(X1, Boston, Chicago); Δ)). It then uses the S-rules (in particular rule S9 from Table III) to obtain the semantic interpretation, EQUAL (DTIME (X1, Boston), 8:00 a.m.), which it finally substitutes for the symbol Δ in the working string to give the final semantic interpretation of the sentence as: (FOR EVERY X1/FLIGHT: CONNECT(X1, Boston, Chicago); EQUAL (DTIME (X1, Boston), 8:00 a.m.)). This mechanism allows for the nesting of an arbitrary number of quantifiers (depending on the number of quantified NP's) before the S-processor finally interprets the sentence via the S-rules.

The NP-processor

Like the S-processor, the operation of the NP-processor is determined by a set of semantic rules in a "pattern \Rightarrow action" format. The "pattern" part of these rules is a set of templates just as for S-rules, except that the partial tree structures used in the templates are dominated by the node NP instead of S. Some partial tree structures for the NP-processor are

*In this diagram, NU is a syntactic category for the *number* of a noun phrase (SG for singular and PL for plural). NPR is the category for proper nouns. The other node names should be self-explanatory.

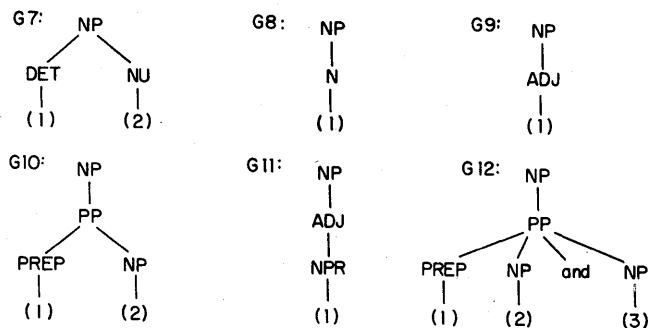


FIGURE 5—Partial tree structures for the NP-processor

given in Figure 5. The difference between G9 and G11 will become apparent later.

The process of interpreting a noun phrase consists of three parts: (1) determining the quantifier to be used, (2) determining the range of quantification, and (3) determining additional restrictive clauses on the range of quantification. The first part is governed primarily by the determiner of the noun phrase and is specified by a set of D-rules, a sample of which are shown in Table IV. The symbol Δ in these rules, as we mentioned before, marks the position where the semantic interpretation of the sentence will ultimately be inserted. Notice that this position is different in the quantifier that results from the interrogative determiners (rule D6) than it is for the others. The symbol ∇ marks the position where the

D1	1-(G7: ((1)=some or (1)=a or (1)=any) and (2)=SG) \Rightarrow (FOR SOME X/ ∇ ; Δ)
e.g.	"American has a flight from Boston to Chicago."
D2	1-(G7: ((1)=each or (1)=every) and (2)=SG) \Rightarrow (FOR EVERY X/ ∇ ; Δ)
e.g.	"Every flight from Boston to New York goes to J.F.K."
D3	1-(G7: (1)=no and (2)=SG) \Rightarrow NOT (FOR SOME X/ ∇ ; Δ)
e.g.	"No flight goes from Boston to Chicago."
D4	1-(G7: (1)=not every and (2)=SG) \Rightarrow NOT (FOR EVERY X/ ∇ ; Δ)
e.g.	"Not every flight from Boston goes to Chicago."
D5	1-(G7: ((1)=the or (1)=this or (1)=that) and (2)=SG) \Rightarrow (FOR THE X/ ∇ ; Δ)
e.g.	"American Airlines operates the flight which leaves Boston for Chicago at 9:00."
D6	1-(G7: ((1)=which or (1)=what) and (2)=SG) \Rightarrow (FOR THE X/ ∇ AND Δ ; LIST (X))
e.g.	"Which flight from Boston to Chicago leaves Boston at 8:00?"

TABLE IV—D-rules for the NP-processor

range of quantification and other restrictive clauses are to be inserted by the NP-processor. The symbol X stands for the variable name which the NP-processor will create. The NP-processor will tag the NP with this variable name and use it in place of X in the quantifier which it produces.

The second phase of the NP-processor is to determine the range of quantification. This is primarily determined by the noun of the noun phrase, and is specified by a set of N-rules. The right hand side of these rules specifies a successor function for the set over which quantification is to range and possibly some restrictions on the range. It indicates by the symbol ∇ the position where any additional restrictions are to be inserted. The NP-processor will replace the symbol ∇ in the quantifier (produced by the D-rules) with the right hand side of a matching N-rule (which will maintain exactly one symbol ∇ in the working string). A small sample of relatively straight forward N-rules is given in Table V. Notice the case of rule N2 where the noun specifies both a successor function and an additional restriction; it is this mechanism that allows us to have a small number of successor functions in the system and still use a considerably larger variety of nouns. Some more complicated N-rules will be described in the next section.

The third phase of the NP-processor attaches additional restrictive clauses onto the range of quantification. For relative clauses modifying the NP the NP-processor does this automatically by first taking the variable X which it has created and using it to tag the NP in the relative clause which dominates the relative pronoun, "which," "who," or "that". It then calls the S-processor to interpret the relative clause. The inter-

N1	1-(G8: (1) = flight or (1) = plane) ⇒ FLIGHT: ∇ e.g. "a flight"
N2	1-(G8: (1) = jet) ⇒ FLIGHT: JET (X) AND ∇ e.g. "a jet"
N3	1-(G8: (1) = airline or (1) = carrier) ⇒ AIRLINE: ∇ e.g. "an airline"
N4	1-(G8: (1) = city or (1) = town) ⇒ CITY: ∇ e.g. "a city"
N5	1-(G8: (1) = airport or (1) = place) ⇒ AIRPORT: ∇ e.g. "an airport"

TABLE V—Some sample N-rules for the NP-processor

pretations of adjectives and prepositional modifiers, however, are specified by a set of R-rules. The right-hand side of these rules consists of a proposition which is to be added as a restriction to the range of quantification. When the third phase of the NP-processor has determined all of the restrictions implied by R-rules and by relative clauses, it conjoins them and replaces the symbol ∇ in the working string with this conjunction. If there are no restrictions, it replaces the ∇ with a vacuous restriction (represented by a dash). A sample set of R-rules is given in Table VI. Notice the use of a noun phrase denoting an airline as an adjective in rule R8. This is the reason for the partial tree structure G11 for noun phrases used as adjectives as distinct from G9 for lexical adjectives.

R1	1-(G9: (1) = non-stop) and 2-(G10: (1) = from and PLACE ((2))) and 3-(G10: (1) = to and PLACE ((2))) ⇒ EQUAL (NUMSTOPS (X, 2-2, 3-2), 0) e.g. "a non-stop flight from Boston to Chicago"
R2	1-(G9: (1) = first-class) ⇒ SERVCLASS (X, first-class) e.g. "a first-class flight"
R3	1-(G9: (1) = jet) ⇒ JET (X) e.g. "a jet flight"
R4	1-(G9: (1) = propeller) ⇒ NOT (JET (X)) e.g. "a propeller flight"
R5	1-(G10: (1) = from and PLACE ((2))) and 2-(G10: (1) = to and PLACE ((2))) ⇒ CONNECT (X, 1-2, 2-2) e.g. "a flight from Boston to Chicago"
R6	1-(G10: (1) = from and PLACE ((2))) ⇒ DEPART (X, 1-2) e.g. "a flight from Boston"
R7	1-(G10: (1) = to and PLACE ((2))) ⇒ ARRIVE (X, 1-2) e.g. "a flight to Chicago"
R8	1-(G11: AIRLINE ((1))) and 2-(G8: (1) = flight or (1) = plane or (1) = jet) ⇒ EQUAL (OWNER (X), 1-1) e.g. "an American Airlines flight"
R9	1-(G9: (1) = morning or (1) = a.m.) and 2-(G8: (1) = flight or (1) = plane or (1) = jet) and 3-(G10: (1) = from and PLACE ((2))) ⇒ GREATER (1200, DTIME (X, 3-2)) e.g. "a morning flight from Boston"

TABLE VI—Some R-rules for the NP-processor

Functionally determined objects

As we mentioned previously, linguistic forms such as "the departure time of _____ from _____" correspond to functions which assign values (in this case a time) to each combination of arguments (in this case a flight and a place). That is, a noun phrase such as "the departure time of AA-57 from Boston" is a complex designator for an object that is functionally determined from the arguments "AA-57" and "Boston." Other such constructions are "owner of X1," "the father of the boy," "the captain of the ship," "the sum of x and y." In such constructions, the noun of the noun phrase is the name of the function, and the arguments are specified by prepositional phrases (usually with the preposition "of"). A few question-answering systems have recognized the functional nature of such constructions. Bobrow gives a general discussion of the correspondence between such linguistic forms and functions, and he uses such constructions for addition, subtraction, multiplication, etc., in his system. The DEACON system also, can handle a limited class of such functional noun phrases, namely those which denote the value of some attribute of an object. For example "the owner of the Enterprise" denotes "the Unites States," which is the value of the attribute "owner" associated with the object "Enterprise" in the data base. In both of the above systems, however, only single-valued functions are considered. In such cases the determiner of a functional noun phrase is always 'the,' and in both systems this determiner is either dropped or has no semantic effect.

Not all such functions are single-valued, however. Take for example, "a member of the crew," "a cousin of the duke," "an eigenvalue of the matrix," "an officer of the ship," "a root of the equation." In such cases the functional noun phrase may contain any of the determiners used to denote quantification over a set, e.g., "every officer of the ship," "no officer of the ship," "some officer of the ship," "which officer of the ship." We have already described a mechanism which handles these quantifiers given a successor function which enumerates the members of a set. We can handle them for functionally determined objects by defining a successor function which takes as additional arguments the arguments of the functional noun phrase. For example, a successor function OFFICER (x, y) could be defined to enumerate the officers of a ship x. Then, if the values of the pointers to the officers of x were a_1, a_2, \dots, a_n , we would have:

$$\begin{aligned} \text{OFFICER}(x; 0) &= a_1 \\ \text{OFFICER}(x, a_1) &= a_2 \\ &\vdots \\ \text{OFFICER}(x, a_n) &= \text{END}. \end{aligned}$$

With this mechanism, we can represent the range of quantification in the quantifiers by naming a successor function together with the fixed arguments of the functional noun phrase (the argument for the index of enumeration is always assumed and need not be represented). Thus the sentence:

Every officer of the Enterprise is ashore

can be represented as:

```
(FOR EVERY X1/OFFICER(ENTERPRISE):-;
  ASHORE(X1))
```

where ASHORE(X) is a predicate meaning X is ashore.

Once the above mechanism has been constructed for multiple-valued functions, single-valued functions are seen to fall naturally into the same scheme. In this case, the successor function will enumerate exactly one object before returning the value "END" and the retrieval mechanism for the quantifier "THE" will automatically verify whether the use of the determiner "the" is correct (i.e., it will verify that exactly one object is generated). This mechanism will handle the usage of the determiner "the" in all cases where the object involved is uniquely determined, even in the case of a "very small" ship which has only one officer (or in the case where additional modifiers appear in the functional noun phrase and restrict the range of quantification, as in "the redheaded officer of the ship"). This approach to quantified noun phrases allows full generality in handling both functional and non-functional noun phrases with the full range of possible determiners and modifiers.

Since, in the formulation of functional noun phrases described above, the function of the determiner and additional modifiers in the noun phrase is the same as for non-functional noun phrases, the same D-rules and R-rules will work for both functional and non-functional noun phrases. Furthermore, the N-rules for functional noun phrases will operate exactly as do the N-rules for non-functional noun phrases. A sample set of N-rules for functional noun phrases is given in Table VII. Notice that in rule N12, some of the arguments of the function are specified by adjectives as well as by prepositional phrases. Notice also that many of the function names are compound forms. Note especially that while the construction "number of stops" has a meaning to the user which is based on counting stops, the system has no data base objects corresponding to stops and does no counting. Instead, it looks up the number of stops in a table. Thus the phrase "number of stops" is effectively an idiom to the system and serves merely to name a function.

N6	1-(G8: (1) = departure time) and 2-(G10: (1) = of and FLIGHT ((2))) and 3-(G10: (1) = from and PLACE ((2))) ⇒ DTIME (2-2, 3-2): ∇ e.g. "the departure time of AA-57 from Boston"
N7	1-(G8: (1) = arrival time) and 2-(G10: (1) = of and FLIGHT ((2))) and 3-(G10: ((1) = in or (1) = at) and PLACE ((2))) ⇒ ATIME (2-2, 3-2): ∇ e.g. "The arrival time of AA-57 in Chicago"
N8	1-(G8: (1) = owner or (1) = operator) and 2-(G10: (1) = of and FLIGHT ((2))) ⇒ OWNER (2-2): ∇ e.g. "the operator of AA-57"
N9	1-(G8: (1) = time zone) and 2-(G10: (1) = of and PLACE ((2))) ⇒ TZ (2-2): ∇ e.g. "the time zone of Boston"
N10	1-(G8: (1) = number of stops) and 2-(G10: (1) = of and FLIGHT ((2))) and 3-(G12: (1) = between and PLACE ((2)) and PLACE ((3))) ⇒ NUMSTOPS (2-2, 3-2, 3-3): ∇ e.g. "the number of stops of AA-57 between Boston and Chicago"
N11	1-(G8: (1) = type of plane or (1) = kind of plane) and 2-(G10: (1) = of and FLIGHT ((2))) ⇒ EQUIP (2-2): ∇ e.g. "the type of plane of AA-57"
N12	1-(G8: (1) = fare) and 2-(G9: (1) = one-way or (1) = round-trip) and 3-(G9: (1) = first-class or (1) = coach or (1) = stand by) and 4-(G10: (1) = from and PLACE ((2))) and 5-(G10: (1) = to and PLACE ((2))) ⇒ FARE (4-2, 5-2, 2-1, 3-1): ∇ e.g. "one-way first-class fare from Boston to Chicago"

TABLE VII—Some N-rules for functional noun phrases.

"What" questions

One frequently asks for the value of a function for specified arguments by means of a "what" question as:

What is the departure time of AA-57 from Boston?

If it can be avoided, we do not want such questions to be answered by ranging over the universe of possible answers and choosing the correct one, but instead we want to determine the correct answer directly from the arguments of the function by means of the operational procedure which defines the function. Thus the interpretation of this sentence should be:

(FOR THE X1/DTIME(AA-57, Boston):-;
LIST(X1))

rather than:

(FOR THE X1/TIME: EQUAL (X1, DTIME(AA-57,
Boston)); LIST(X1)).

The procedure that we have just described will generate the quantifier:

(FOR THE X1/DTIME(AA-57, Boston):-; Δ)

which we require. All that remains is to interpret the sentence itself as the command LIST. This is specified by following semantic rule:

S23 1-(G1: (2) = be) and
2-(G2: (1) = be and (2) = what)
⇒ LIST (1-1) .

Thus, assuming that the deep structure of the sentence is something like that of Figure 6, then rule S23 will apply to the topmost S, but will first call for the semantic interpretation of the subject noun phrase. Rules D5 and N6 will then generate the quantifier, and finally, rule S23 will insert the command LIST(X1) to give the final interpretation

(FOR THE X1/DTIME(AA-57, Boston):-; LIST
(X1)).

CONCLUSION

Summary

In the preceding sections, we have presented a proposal for a uniform method of performing the semantic interpretation of English questions for a question-answering system. We have presented an operational definition of semantic meaning in terms of programmed subroutines and have outlined a formal query language for representing the semantic interpretation of questions and commands. We have concentrated on the prob-

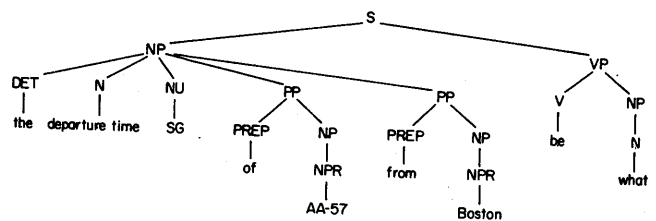


FIGURE 6—Phrase marker for a "what" question

lem of translating from a syntax tree that represents the syntactic structure of the input question into an expression in the query language which represents the "meaning" of the question, and we have developed a procedure for performing this translation by means of a set of semantic rules. While the techniques described are not completely general for unrestricted English usage, they are applicable in a wide variety of situations in which the computer is called upon to "understand" English input and take some appropriate action.

In addition to recognizing the predicate which corresponds to a given sentence, the proposed system handles universal and existential quantifiers, numerical quantifiers, interrogative determiners, and the determiner "the." It handles relative clauses, adjectival and prepositional modifiers in noun phrases, and functionally determined noun phrases. It also takes into proper consideration whether a noun phrase is singular or plural. This combination of features has not been attained in any existing question-answering system although some have appeared in isolation in various systems. To my knowledge, the handling of the determiner "the," the treatment of relative clauses, and the distinction between singular and plural nouns are new, as is the uniform framework in which these techniques are integrated. Moreover, the techniques presented here are general and easily extended to other situations in which there is a well-defined data base.

Implementation

A sample implementation of the prototype system for the flight schedules problem has been designed to interpret the deep structures assigned by a small transformational grammar containing 40 transformations. Although these phrase markers could in principle be mechanically assigned,* they are currently produced by hand, since the central concern of this research is the design of the semantic interpreter. The semantic interpreter itself has been programmed in LISP and tested on a variety of sentences including those listed in Table VIII. The interpreter is driven by a set of 36 S-rules,

*The grammar conforms sufficiently to the conditions imposed by Petrick¹⁰ that it could be recognized by a slight modification of Petrick's transformational recognition procedure.

1. a. AA-57 flies from Boston to Chicago.
b. CONNECT (AA-57, BOSTON, CHICAGO)
2. a. Doesn't American operate flight AA-57?
b. TEST (EQUAL (OWNER (AA-57), AMERICAN))
3. a. Isn't AA-57 an American Airlines flight?
b. TEST ((FOR SOME X1/FLIGHT:
EQUAL (OWNER (X1), AMERICAN AIRLINES);
- EQUAL (AA-57, X1)))
4. a. Does American have a flight which goes from Boston to Chicago?
b. TEST ((FOR SOME X1/FLIGHT:
CONNECT (X1, BOSTON, CHICAGO);
EQUAL (OWNER (X1), AMERICAN)))
5. a. Does American have a flight which doesn't go from Boston to Chicago?
b. TEST ((FOR SOME X1/FLIGHT:
NOT (CONNECT (X1, BOSTON, CHICAGO));
EQUAL (OWNER (X1), AMERICAN)))
6. a. What American Airlines flight goes from Boston to Chicago?
b. (FOR THE X1/FLIGHT:
EQUAL (OWNER (X1), AMERICAN AIRLINES)
AND CONNECT (X1, BOSTON, CHICAGO);
LIST (X1))
7. a. What American Airlines flights go from Boston to Chicago?
b. (FOR EVERY X1/FLIGHT:
EQUAL (OWNER (X1), AMERICAN AIRLINES)
AND CONNECT (X1, BOSTON, CHICAGO);
LIST (X1))
8. a. What is the departure time of AA-57 from Boston?
b. (FOR THE X1/DTIME (AA-57, BOSTON):-;
LIST (X1))
9. a. What is the departure time from Boston of every American Airlines flight that goes from Boston to Chicago?
b. (FOR EVERY X2/FLIGHT:
EQUAL (OWNER (X2), AMERICAN AIRLINES)
AND CONNECT (X2, BOSTON, CHICAGO);
(FOR THE X1/DTIME (X2, BOSTON):-;
LIST (X1)))
10. a. What American Airlines flights arrive in Chicago from Boston before 1:00 p.m.?
b. (FOR EVERY X1/FLIGHT:
EQUAL (OWNER (X1), AMERICAN AIRLINES)
AND ARRIVE (X1, CHICAGO) AND
GREATER (1:00 p.m., ATIME (X1, CHICAGO));
LIST (X1))
11. a. Are all flights from Boston to Chicago American Airlines flights?
b. TEST ((FOR EVERY X1/FLIGHT:
CONNECT (X1, BOSTON, CHICAGO);
(FOR SOME X2/FLIGHT:
EQUAL (OWNER (X2), AMERICAN AIRLINES);
EQUAL (X1, X2))))
12. a. How many flights that go from Boston to Chicago does American Airlines operate?
b. LIST (NUMBER (X1/FLIGHT:
CONNECT (X1, BOSTON, CHICAGO) AND
EQUAL (OWNER (X1), AMERICAN AIRLINES)))
13. a. How many airlines have more than 3 flights that go from Boston to Chicago?
b. LIST (NUMBER (X1/AIRLINE:
(FOR GREATER (N,3) MANY X2/FLIGHT:
CONNECT (X2, BOSTON, CHICAGO);
EQUAL (OWNER (X2), X1))))
14. a. What is the number of flights from Boston to Chicago?
b. (FOR THE X1/NUMBER (X2/FLIGHT:
CONNECT (X2, BOSTON, CHICAGO)):-;
LIST (X1))

TABLE VIII—Sample test sentences and their semantic interpretations.

16 D-rules, 17 N-rules, and 11 R-rules. Again, since the primary concern was the design of the semantic interpreter, no retrieval component for the flight schedules problem has been implemented. To do so, however, would be a straightforward problem of writing LISP functions for each of the semantic primitives and for the quantifiers. The semantic interpretations produced by the interpreter are already in the form of LISP S-expressions** and would simply be evaluated in the retrieval component by the LISP function evaluator.

Extendability

The query language in the proposed scheme constitutes a standard interface between the semantic interpreter and the retrieval component, thus freeing the semantic interpretation algorithm from dependence on the particular structure of the data base. It also frees the interpreter from a distinction between answers that are actually stored in the data base and answers which must be computed from the data base. The query language is thus easily extended by allowing new predicates, functions, and commands to be added as actual coded subroutines which the retrieval component may call. In addition, the semantic interpreter is a general routine, driven by a set of semantic rules, thus completing the facility for extendability. Any concept that can be made explicit for the machine by means of a programmed subroutine can be added to the retrieval component, and new semantic rules can be added to the semantic interpreter to specify the ways in which the concept is expressed in English. If new kinds of syntactic structures in English are involved, then new rules may also have to be added to the parser in order to recognize the new constructions.

**The LISP S-expression notation is a trivial modification of the notation used in this paper, differing mainly in that the function name is included in the list of arguments rather than outside—e.g. (TEST X1) instead of TEST(X1).

BIBLIOGRAPHY

- 1 R F SIMMONS
Answering English questions by computer: a survey
Communications of the ACM Vol 8 No 1 Jan 1965 pages 53-70
- 2 Deacon Project
Phrase-structure oriented targeting query language
RM65TMP-64 TEMPO General Electric Co Santa Barbara California Sept 1965
- 3 W A WOODS
Semantics for a question-answering system
PhD thesis Division of Engineering and Applied Physics
Harvard University Cambridge Massachusetts Aug 1967
Also Report NSF-19 Harvard Computation Laboratory
- 4 D G BOBROW
A question-answering system for high school algebra word problems
AFIPS Conference Proceedings Vol 26 1964 Fall Joint Computer Conference pages 591-614
- 5 B RAPHAEL
A computer program which 'understands'
AFIPS Conference Proceedings Vol 26 1964 Fall Joint Computer Conference pages 577-589
- 6 R K LINDSAY
Inferential memory as a basis of machines which understand natural language
In Feigenbaum E A and Feldman J (eds) Computers and Thought McGraw-Hill New York 1963 pages 217-233
- 7 F S BLACK
A deductive question-answering system
Unpublished PhD thesis Division of Engineering and Applied Physics Harvard University Cambridge Massachusetts June 1964
- 8 *Official Airline Guide*
Quick Reference North American Edition Standard reference of the Air Traffic Conference of America March 1 1966
- 9 J J KATZ J A FODOR
The structure of a semantic theory
In Katz J J and Fodor J A (eds)
The Structure of Language: Readings in the philosophy of language
(Prentice-Hall Englewood Cliffs New Jersey 1964 pages 479-518
- 10 S R PETRICK
A recognition procedure for transformational grammars
Unpublished PhD thesis Department of Modern Languages
Massachusetts Institute of Technology Cambridge Massachusetts May 1965

A natural language compiler for on-line data management

by CHARLES H. KELLOGG

System Development Corporation
Santa Monica, California

INTRODUCTION

During the past few years there has been a rapid advance in the technology of time-sharing systems and software to permit quick access to large files of structured data. This has led to a growing interest in communicating with computer files directly in a natural language such as English. The natural language systems described in the literature are largely small-scale research vehicles dealing with small data bases of restricted subject scope. Giuliano (1965), among others, has questioned the generalization of these systems to wider universes of discourse. Developments in this area have been reviewed by Simmons (1966), and by Bobrow, Fraser and Quillan (1967). In contrast, the work in on-line data management has been more concerned with the efficient organization of structured data to allow for quick access and maintenance of large volumes of formatted information [see the reviews by Kellogg (1967), Climsonson (1966), and Minker and Sable (1967)].

At SDC, we have been concerned for some time with the design and implementation of a prototype system that combines the advantages of the two approaches: the natural language system's facility for accepting ordinary English, and the on-line data management system's ability to manipulate large quantities of data of varying subject matter. We have been particularly concerned with striking a reasonable compromise between the difficulties of allowing completely free use of ordinary English and the restrictions inherent in existing artificial languages for data base description and querying.

This paper will discuss a number of the properties that seem necessary and desirable for an on-

line English subset. We will describe the implementation of an English subset and the construction of a compiler that translates facts and requests for facts, expressed in ordinary English grammatical patterns, into procedures in an intermediate language (IL)—procedures that are sufficiently explicit to be directly accepted and processed by a data management machine. The use of computer-to-user feedback to ameliorate some of the confining restrictions of the English subset will be illustrated, as will the means whereby the user can extend the vocabulary and scope of the English subset to permit the description and interrogation of data bases of varying structure and content.

We will show how semantic structures can be precisely represented as procedures and how, from this point of view, a natural language compiler constitutes a device for transforming natural language syntactic structures into underlying semantic structures. Along the way we will draw upon a number of examples illustrating the on-line use of the most recent version of the prototype system.

Research methodology

The components of an experimental system (called CONVERSE) for investigating the feasibility of natural language compilers and for constructing and testing such compilers are shown in Figure 1. The programs and associated files are implemented on the IBM Q-32 Time-Sharing System at SDC.

The META-LISPX meta (or syntax-directed) compiler developed by Book and Schorre (1965) has been used to implement, test and modify the natural language compiler and data management

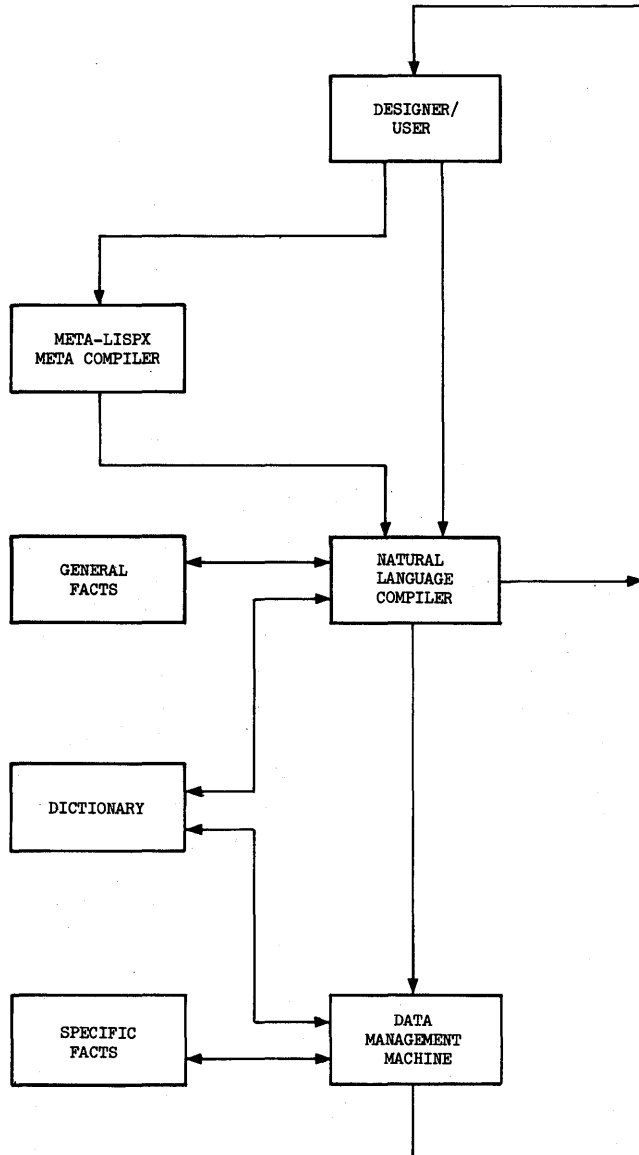


FIGURE 1—The CONVERSE system

machine. META-LISPX is implemented within a version of the LISP 1.5 List Processing System. It accepts a description of an object compiler expressed as a series of phrase structure rewrite rules. These META rules are associated with procedures, written in LISPX, that represent the actions to be taken once a substring or constituent of an input sentence has been recognized. Since these actions can take the form of any procedure expressible in the LISP list processing language, they can be of any desired degree of generality or complexity. In particular these procedures implement a process of semantic interpretation—a process that is controlled or directed by the recogni-

tion of syntactically well formed constituents. The object natural language compiler produced by the META compilation process contains a top-to-bottom left-to-right syntactic recognition algorithm which is driven by the encoded form of the phrase structure rules input in the META-LISPX language. As illustrated in Figure 1, a data management machine has also been produced using META-LISPX. These object programs are easily modified or extended by incremental recompilation.

In use, the natural language compiler first constructs and then uses a file of general facts to control the compilation process. It also updates and utilizes information in the dictionary that it shares with the data management machine. Procedures produced by the compiler drive the machine to store, modify, or search a file of specific facts.

The difficulty in using a meta- or syntax-directed compiler to implement a subset of natural language has been recognized by Cheatham and Warsall (1962). They point out that, in the case of translating a programming language input string into machine language, the formation rules of the programming language have semantic significance. Consequently, given a parse of the input string the semantic interpretation of this string is relatively straightforward. They go on to discuss the possibility of phrasing formation rules for a subpart of English in a form acceptable to a syntax-directed compiler: "however in contrast to the case of algebraic compilers an application of one of these rules does not in general correspond to a semantically meaningful act." Their point is well taken. As we shall see the recognition algorithm built into a syntax-directed compiler must be supplemented by a rather considerable amount of semantic interpretation apparatus in order to deal with the complexities (such as ambiguity resolution) of even a small subset natural language.

There are six basic actions which we require from a natural language compiler:

1. An anomalous input sentence, one lying outside of the English subset, must cause the compiler to construct appropriate feedback messages and send this information to the user. This information serves two important purposes. First, it will make the user aware of the current limitations of the Eng-

- lish subset, and secondly, it should enable him to take constructive steps to enrich the English subset. The feedback should give the user information as to how and why the English subset was exceeded.
2. Sentences which are syntactically or semantically ambiguous must be recognized as such and, where possible, the compiler should resolve this ambiguity. For example, semantic information can often be used to resolve cases of syntactic ambiguity and vice versa. When this is impossible the ambiguous interpretations must be displayed to the user in the intermediate language format.
 3. A declarative sentence specifying general information must cause the natural language compiler to update its own store of such data and create appropriate dictionary entries for newly recognized words.
 4. A declarative sentence specifying specific information must be translated into explicit intermediate language file storage procedures.
 5. A declarative sentence specifying a definitional extension of the English subset vocabulary must lead to appropriate actions to extend the range of interpretations of existing words in the dictionary, or to add newly defined words to the dictionary.
 6. The sixth basic action is to recognize interrogative sentences and to translate the requests implicit in these sentences into explicit file searching procedures.

Early experimental work on the CONVERSE system is reported in Kellogg (1967a, 1967b). At first (1967a) we developed an English subset only for questions and we used the QUUP artificial query language for the LUCID data management system as the intermediate language. Further, we translated questions into QUUP queries for only a single data base. Thus this initial experimental system was open to the same critical question asked of many experimental natural language systems, namely, "is it generalizable to other data bases of significantly different structure and/or content?"

Our next step was to allow some degree of generalization to data bases of different structure or content. This approach was based on a "computer interrogation of the user technique" (1967b). The

user was led through a series of steps designed to obtain a description of his data base from the answers he provided to the computer displayed questions.

The body of this paper is concerned with the more general framework already alluded to, viz., one in which a user can input interrogative and declarative sentences in any order to query a data base, to modify the content of a data base, or to extend the range of semantic structures admissible with respect to that data base.

Properties of the source language and the target language

The function of the (English) source language is to provide an on-line user with the ability to communicate his data and requests in the ordinary grammatical patterns or syntactic structures of English. The function of the target (intermediate) language is to make explicit the semantic structure of the English sentence as a series of file storage and search procedures. Syntactic and semantic structures are characterized by syntactic and semantic categories and by the allowed associations or relations among these categories. These characterizations may be represented formally by the use of rewrite or formation rules.

Figure 2 illustrates some of the categories used in structuring the English subset and intermediate languages. Such traditional categories as *noun phrase*, *verb phrase*, *adjectival phrase*, *prepositional phrase* and the lower level categories: *noun*, *adjective*, *preposition*, *verb* and *determiner* are used to formally specify the grammar of our English subset. The underlying, or deep semantic structures that we recognize comprise such semantic categories as *procedure*, *proposition*, *term*, *predicate*, *operator*, *function*, *relation*, *set*, and *property*. Lower level semantic categories correspond to concepts such as: *DEPART*, *SMOGGY*, and *STATE*. The members of the lowest level syntactic categories are the words in the dictionary, while the members of the lowest level semantic categories are seen to be groupings of ordered N-tuples of semantic values in the data base.

The heart of the difficulty in dealing with natural language for the computer is not only the great complexity of the syntactic structures which may arise in a natural language, but even more so, the complexity of the associations and relations which may obtain between members of syntactic categories and members of semantic categories.

In particular the associations between English subset words and the semantic values in a data base may be many-to-many and complex. The task of a natural language compiler is to recursively decompose sentences into their structural components and then to somehow use this information to cause the recursive composition of semantic categories into higher level semantic structures. The difficulty is one of establishing the correct relationships between syntactic and semantic categories, and sorting these out in an appropriate way to produce semantic structures that represent the facts or requests implicit in an English sentence.

Our semantic categories largely correspond to just those basic elements used in modern mathematics and symbolic logic to state facts about the actual or some possible world. Namely, the notions of object, classes or sets of distinct kinds of objects, properties and relations that may range over objects, and functions, which when applied to objects evaluate to other objects. Experience in mathematics and logic has shown that such a basic set of elements are sufficient, along with a few other devices, for the formulation of scientific

theories. Bohnert (1966) has gone farther and expressed the view that such elementary elements (as they are allowed for in the predicate calculus) constitute a reasonable framework for representing the "deep" structure of natural languages.

We are now in a position to draw a precise distinction between data base structure and data base content. Data base semantic content correlates with Carnap's (1956) notion of extensional meaning. Similarly, data base semantic structure corresponds to his notion of intensional meaning. For example, "smoggy" (the semantic category) designates a property of certain things. This is the intension of the concept "smoggy." On the other hand, the extension of "smoggy" comprises the list of objects in the data base that possess this property. The intensional or structural aspect of meaning will be represented in terms of *general facts* constituting a *data base description*. The *data base content* is the extensional aspect of meaning and it is represented as a series of *specific facts*. Thus a change in the data base structure is taken to mean a change in data base description or equivalently a change in the intensional meaning of the universe of discourse, while a

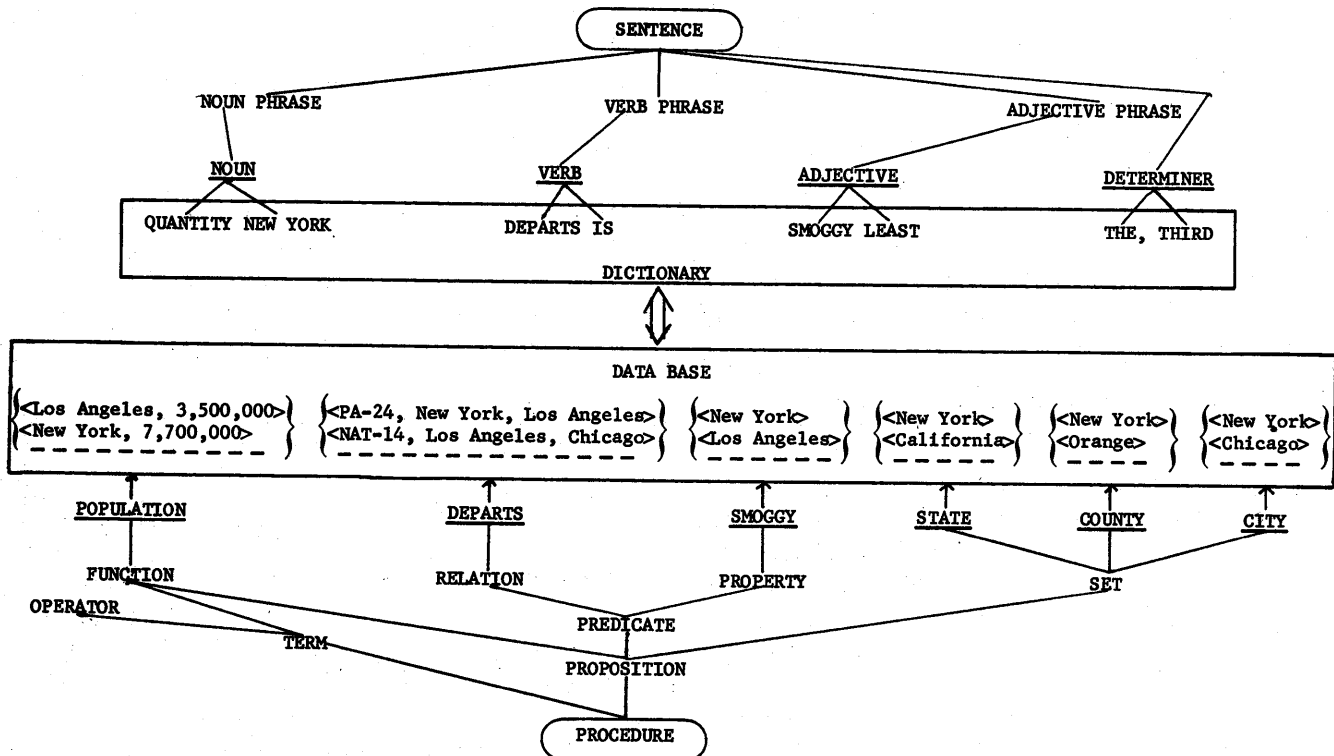


FIGURE 2—Syntactic and semantic categories

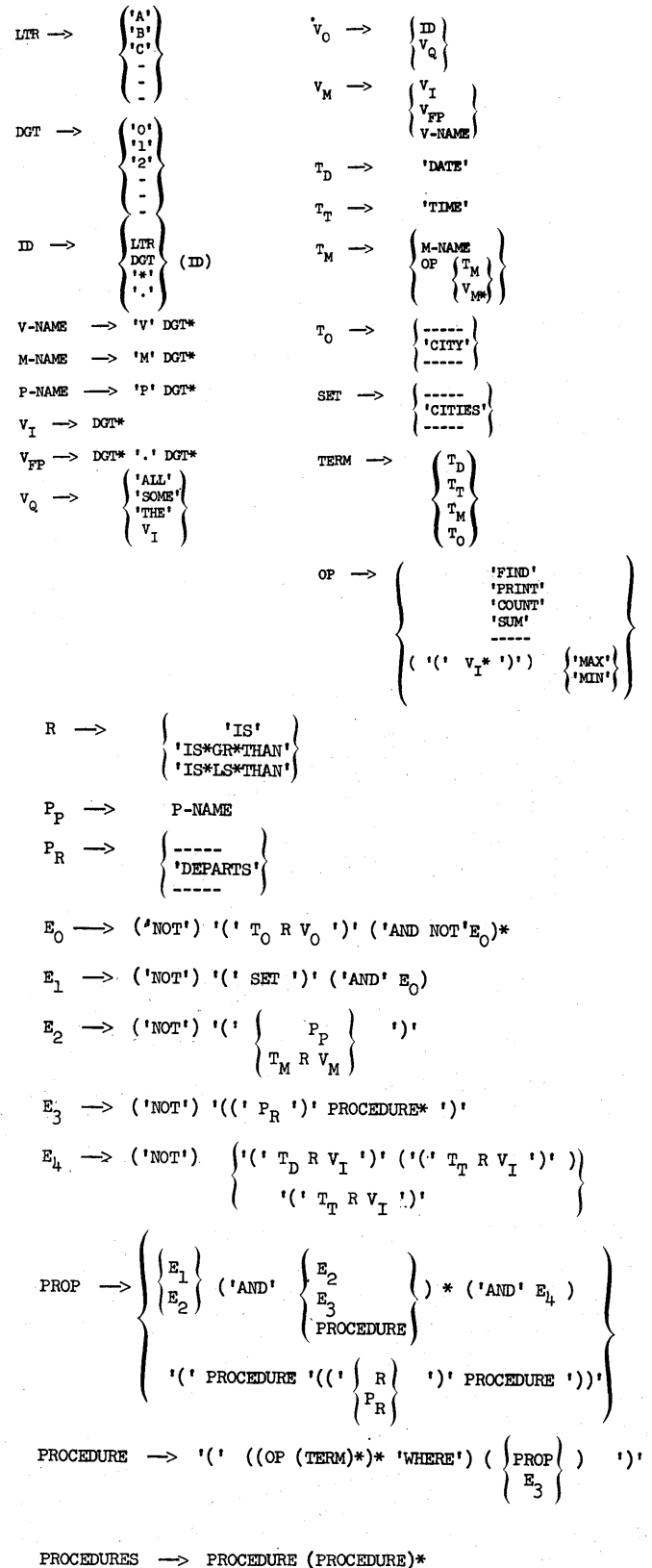
change in data base content constitutes a change in the extensional meaning of the universe of discourse.

A rough correspondence between some of the syntactic categories and semantic categories can be given as follows: In general, common nouns may have many-to-many associations with sets. For example, there may be a number of common nouns, each of which designates the same kind of objects in the data base. On the other hand, one common noun may designate several kinds of objects included in the data base. Again several proper nouns may specify the same object in the data base or one proper noun (such as "New York") may refer to objects in several different sets. Similarly, there may be a many-to-many association between the transitive verbs and some of the prepositions in the English subset, and the various relations specified for the data base. Many-to-many relations may also obtain between noun phrases and functions and between adjectives and properties.

The present grammar for the English subset consists of approximately 100 rules of the phrase structure type and a few syntactic transformation rules. The grammar admits a reasonably wide class of declarative and interrogative sentences but it has been kept relatively small by implementation of just those rules that allow us to recognize the syntactic clues essential for driving the semantic interpretation phase of the natural language compiler. The word classes include the determiners, adjectives, proper, common, and abstract nouns, adverbs, verbs, prepositions, question words, relative pronouns, and conjunctions. Several forms of relative and dependent clauses, and preposition, noun, verb, adjectival and adverbial phrases are recognized. The example given later in the paper will provide a reasonably good idea of the range of acceptable sentences.

A formal specification of the rules for the intermediate language is shown in Figure 3. In designing this language, we have been guided by our earlier experience in translating questions into queries admissible to the LUCID data management system and by three principal design objectives:

1. The language must be procedural. It should be comparable to procedure-oriented programming languages with regard to implementation and independence from particu-



Note: Terminal symbols appear in quotes. Brackets indicate a choice of elements. Parentheses indicate optional elements. An asterisk indicates an element may be repeated.

FIGURE 3—Formation rules for the intermediate language

lar machine configurations or detailed information structure considerations.

2. It should be a powerful and expressive language for stating both storage and search procedures. In this regard, we have been influenced by the structure of the predicate calculus. The intermediate language permits the composition of functions, the nesting of relations, the embedding of procedures within procedures, and quantification over sets.
3. The language should be relatively easy to read and understand.

The basic components are:

- . letters,
- . digits,
- . identifiers (the words of the intermediate language),
- . value-names,
- . measure-names (or function-names),
- . property-names,
- . forms of values to represent date, time, integer, and floating point numbers,
- . terms that specify the proper names of objects and values of measure functions,
- . operators that range over terms,
- . several kinds of predicates, and
- . a series of *elements* ($E_0 - E_4$) used to assert conditions on sets, objects, and relations.

The first form of element (E_0), for example, allows for such expressions as "City is Los Angeles," and "not City is Los Angeles and not City is San Francisco." We can quantify over sets by expressions such as "City is all," "City is some," "City is the," which must be interpreted, respectively, as: each member of the set of objects must meet the conditions specified; and exactly one object must meet the conditions specified.* Membership in a set may also be qualified with respect to a cardinal number. Rule E_1 associates the membership restriction element E_0 with the name of a particular set. Rule E_2 introduces the names of properties and expressions such as, "Population is larger than 500,000," that restrict a **measure function** to particular values for an object. E_3 serves to introduce the name of a relation and a specification of the codomain of the relation in terms of one or more procedures. An E_4 element specifies time, date, or both.

The next class of components are the proposi-

tions (PROP). They are the principal elements of procedures, as they specify the conditions which must be met for the storage or retrieval of information. Each proposition specifies a set, a property, or a function with restricted range. The subset of objects so specified may be further restricted by the conjunction of additional elements or procedures. The procedure imposes operations upon specific facts that have been conditionally specified by a proposition.

Relations among procedures are permitted in a proposition. For example, the question "Is the population of Los Angeles greater than the population of New York?" is represented in IL as

```
((FIND M1 WHERE ((CITIES) AND
(CITY IS LOS ANGELES))) IS*GR*
THAN
```

```
(FIND M1 WHERE ((CITIES) AND
(CITY IS NEW YORK))))
```

where M_1 is an M-name standing for the measure function "population."

When a proposition is evaluated, it returns a "true" or "false" value and an *F-name* for each fact satisfying the proposition. Similarly, a procedure returns a list of F-names and operation-derived values.

The procedures statement is the highest level semantic structure in the intermediate language. Each procedure is understood to be in an inclusive "or" relationship with the others. This statement type allows us to represent the meaning of a sentence in a form similar to the disjunctive normal form of symbolic logic. Within a procedure or proposition the only connectives allowed are conjunction and denial. Disjunction occurs only between procedures. This restriction simplifies the problem of representing the semantic interpretation of a sentence and also simplifies the logic of the data management machine. There is no substantial loss of expressive power, since it has been shown that any proposition expressible in symbolic logic can be represented as a proposition in disjunctive normal form.

The data description process

Data description is a process of communicating

*The facilities to allow for quantification have not yet been added to the natural language compiler.

information about the semantic structure of a data base to the computer. This process requires the identification of general factual information, the addition of new words to the dictionary, and the addition of semantic information to dictionary entries. These processes (adding to the store of general facts and adding to the vocabulary) correspond, respectively, to the processes of axiomatic extension and definitional extension in formal systems. It is the former process which is most critical, as it is the only means of broadening the universe of discourse to include new semantic categories. The process of data description in natural language must involve a method for deriving semantic categories from the input of declarative sentences. This can be accomplished by allowing a user to introduce new words and phrases into declarative English sentences in proper defining contexts. Some of the basic declarative sentence forms which are suitable for this process are illustrated in Figure 4.

Sentence 1 illustrates one of the simplest but most important statements that one can make in describing a data base: the identification of the kinds of objects of interest. A broad universe of discourse will admit many types or sets of objects. Otherwise a universe of discourse is narrow. The natural language compiler will accept statements such as Sentence 1, will assign the syntactic category "common noun" to the term "city," and will construct a record to contain general information about city-type objects.

Sentences 2 through 8 illustrate elementary sentence patterns that may be used to give depth as well as breadth to a characterization of a universe of discourse. Sentences 2 and 4, for example,

- (1) A city is an object.
- (2) A city has a number of local government employees.
- (3) Cities have families.
- (4) A city may be smoggy.
- (5) A city is a place.
- (6) A city is located in a state.
- (7) An English grade is A, B, C, D, or F.
- (8) Distribution factors include wholesale trade and retail trade.
- (9) Define population as the quantity of people.
- (10) Los Angeles is a city.

Note: Doubly underlined words must be NEW, singly underlined words may be NEW.

FIGURE 4—Elementary data description sentence patterns

illustrate the introduction of noun and adjectival phrases to designate descriptive (measure) functions and properties, respectively. Sentences 3 and 5, on the other hand, establish relationships between objects and between sets of objects. Sentence 3 is interpreted to mean that any member of the city set may stand in a possession relationship to one or more members of the family set. Similarly, Sentence 5 leads to a set inclusion relationship between cities and places.

Sentence 6 illustrates a most important elementary sentence pattern, one that allows us to introduce transitive verbs in such a way that the range of subjects and objects may be identified. In this simple example, the designated relation (located) can easily be associated with its domain (cities) and codomain (states). In Kellogg (1967b) we show how steps can be taken during data description to identify the converse of a relation and the logical properties (e.g., transitivity, symmetry, etc.) of relations introduced in this manner. Non-numerical values can be assigned to measure functions designated by noun phrases as shown in Sentence 7. Patterns such as Sentence 8 allows us to specify generic-specific relations among measure functions.

The meaning of new words or phrases may be defined in terms of well-formed English subset words, phrases, or sentences, as illustrated by Sentence 9. Similarly, existing words and phrases may have their meanings changed. In either case, the semantic information associated with the definiens is assigned to the definiendum. The last sentence, comparable in form to Sentence 1, illustrates the introduction of proper names for objects.

The property that a natural language compiler must have in order to accept and process sentences such as those shown in Figure 4 is the property of *semantic reflexivity*. By this property we mean the ability of the compiler to accept English sentences that lead to actions increasing the range of specific facts that may be expressed in the English subset. We see that in order for a compiler to be semantically reflexive, it must be able to identify, store, and access information about semantic categories.

In a recent book, Lenneberg (1967, p. 574) concludes that: "The perception and production of language may be reduced on all levels to categorization processes . . ." and "words label categorization processes." Of interest here is the fact that

many of the elementary categorization processes that he goes on to discuss correspond rather closely to the semantic categories that we have found it necessary to introduce and employ in order to describe data bases in natural language.

The use of CONVERSE to describe the structure and specify the content of a portion of a data base of census information is illustrated in Figure 5. The data base derives from a standard census publication, "County and City Data Book: 1960." This data base is typical of many large formatted data bases, and it is of a size (about 600,000 facts) to require several hundred data-descriptive elements (primarily measure functions) to properly characterize its content. It contains information concerning a large number of social and economic indicators for the several political subdivisions of the United States. Sentence 1 in Figure 5A specifies political subdivisions as a principal object of interest. In this case, since we specified a singular noun form, the computer asks for the input of the plural form. (The program does not as yet contain procedures for dealing with inflectional or derivational suffixes.) We see next how a number of the important characteristics of political subdivisions may be characterized and expressed in a natural way through the use of noun phrases to

describe, at varying levels of generality, some of the significant features of political subdivisions. The compiler associates each noun phrase with a corresponding measure function, and each measure function in turn is assigned a symbolic proper name beginning with the letter M (an *M-name*). Sentences 2 through 13 lead to the construction of a hierarchy of measure functions. This hierarchy is stored in an explicit form that may be accessed by the compiler or by the user in furthering the data description process or in asking questions. The user knows that his input sentence has been accepted when the compiler responds as in Figure 5A with "IN," a list of fact names, or a list of M-names.

Data description may proceed in as verbose or concise a style as the user desires. The use of definitional sentences (such as Sentences 5 and 9, for example) and M-names permits concise statement of a data base description. The 13 sentences in Figure 5A produce a hierarchy of 33 descriptive measure functions (see Appendix). Each of these functions expresses a general fact about political subdivisions. It should be understood that once a measure function (or any semantic category) is specified, further definitional processes may result in a number of linkages between English words and phrases and the semantic category.

The designation of an English subset phrase or sentence is always determined from a complex process of syntactic analysis and semantic interpretation, based on the syntactic and semantic information assigned to the individual words used in the phrase or sentence. This information may have been assigned either as part of the given primitive vocabulary of English subset function words and verbs or as the result of the input data description sentences.

- (1) A POLITICAL SUBDIVISION IS AN OBJECT.
PLURAL?
POLITICAL SUBDIVISIONS.
(F3)IN
- (2) A POLITICAL SUBDIVISION HAS STATISTICAL INDICATORS THAT INCLUDE
PRODUCTION FACTORS, DISTRIBUTION FACTORS,
POPULATION FACTORS, AND LOCAL GOVERNMENT FACTORS.
(M3 M4 M5 M6)IN
(M2)IN
- (3) POPULATION FACTORS INCLUDE DISTRIBUTION OF PEOPLE BY AGE, DISTRIBUTION
OF PEOPLE BY INCOME, DISTRIBUTION OF PEOPLE BY EDUCATION, POPULATION.
(M7 M8 M9 M10)IN
- (4) DISTRIBUTION OF PEOPLE BY AGE INCLUDES MEDIAN AGE OF
PEOPLE, PERCENTAGE
OF PEOPLE UNDER 5 YEARS OF AGE, PERCENTAGE OF PEOPLE OVER 21 YEARS OF
AGE, AND PERCENTAGE OF PEOPLE OVER 65 YEARS OF AGE.
(M11 M12 M13 M14)IN
DEFINE POP AS POPULATION.
- (5) IN
DEFINE PP AS PERCENTAGE OF PEOPLE.
IN
- (6) M5 INCLUDES MEDIAN FAMILY INCOME,
PERCENTAGE OF FAMILY INCOME UNDER \$3,000, PERCENTAGE OF
FAMILY INCOME OVER \$10,000, AGGREGATE FAMILY INCOME IN MILLIONS OF
DOLLARS.
(M15 M16 M17 M18)IN
- (7) M9 INCLUDES PP OVER 25 WITH LESS THAN 5 YEARS OF SCHOOLING,
PP OVER 25 WITH MORE THAN 12 YEARS OF SCHOOLING, MEDIAN NUMBER OF
SCHOOL YEARS COMPLETED.
(M19 M20 M21)IN
- (8) M4 INCLUDES RETAIL TRADE AND WHOLESALE TRADE.
(M22 M23)IN
DEFINE RT AS RETAIL TRADE.
- (9) IN
DEFINE WT AS WHOLESALE TRADE.
IN
- (10) RT INCLUDES NUMBER OF RT ESTABLISHMENTS, NUMBER OF RT
EMPLOYEES, RT YEARLY PAYROLL IN THOUSANDS OF DOLLARS.
(M24 M25 M26)IN
- (11) WT INCLUDES NUMBER OF WT ESTABLISHMENTS, NUMBER OF WT
EMPLOYEES, WT YEARLY PAYROLL IN THOUSANDS OF DOLLARS.
(M27 M28 M29)IN
- (12) M3 INCLUDES NUMBER OF MANUFACTURING ESTABLISHMENTS,
NUMBER OF MANUFACTURING EMPLOYEES, MANUFACTURING PAYROLL
IN THOUSANDS OF DOLLARS.
(M30 M31 M32)IN
- (13) M6 INCLUDES NUMBER OF LOCAL GOVERNMENT EMPLOYEES,
LOCAL GOVERNMENT PAYROLL IN THOUSANDS OF DOLLARS.
(M33 M34)IN

FIGURE 5A—Census data description-I

- (1) POLITICAL SUBDIVISIONS MAY BE SMOGGY, WARM OR COLD, EASTERN OR WESTERN,
LARGE OR SMALL.
(P1 P2 P3 P4 P5 P6 P7)IN
- (2) CITY, COUNTY, STATE, PLACE ARE OBJECTS.
PLURAL?
CITIES, COUNTIES, STATES, PLACES.
(F4)IN
(F5)IN
(F6)IN
(F7)IN
- (3) CITIES, COUNTIES, AND STATES ARE POLITICAL SUBDIVISIONS AND PLACES.
IN
- (4) CITIES HAVE CLIMATE FACTORS THAT INCLUDE TEMPERATURE FACTORS,
MEAN ANNUAL PRECIPITATION, AND MEAN ANNUAL HOURLY WIND VELOCITY.
(M36 M37 M38)IN
(M35)IN
- (5) DEFINE TEMP AS TEMPERATURE.
IN
- (6) TEMP FACTORS INCLUDE MEAN JAN TEMP, MEAN JULY TEMP, HIGHEST TEMP, AND
LOWEST TEMP.
(M39 M40 M41 M42)IN
- (7) COUNTIES AND STATES HAVE AGRICULTURAL FACTORS THAT INCLUDE
FARM LAND IN THOUSANDS OF ACRES, NUMBER OF FARMS.
(M44 M45)IN
(M43)IN

FIGURE 5B—Census data description-II

Several additional forms of descriptive sentences are shown in Figure 5B. Sentence 1 illustrates the introduction of adjectives. These expressions map into properties that are true or false for particular objects in the universe of discourse. The semantic structure of the data base is then extended to include additional forms of objects—cities, counties, states, and places. A compound sentence is used to express an equivalence relationship between political subdivisions and places and a set inclusion relationship between these classes and the classes of cities, counties, and states. Each general fact that applies to political subdivisions and places now applies also to cities, counties, and states. In addition, cities and counties are further distinguished in Sentences 3 through 7.

Figure 5C illustrates several of the sentence forms available for the specification of the proper names of objects. Just as the first step in describing a data base is to name one or more kinds of objects of interest, the first step in specifying specific factual information is to assign proper names to one or more objects. We note that proper names do not have to designate uniquely. The ambiguity of reference for the proper names, Los Angeles, Santa Barbara, and New York, is easily resolved later on, wherever they are used in appropriate contexts in English subset sentences.

The English subset has now been extended to the point where a wide variety of facts about cities, counties, states, and places may be introduced and specified through the use of additional declarative sentences. Several of the admissible syntactic patterns and variations in style are illustrated in Figure 5D. This figure clearly indicates how specific facts may be expressed in either

- (1) ORLANDO IS A CITY.
(F8)IN
- (2) LOS ANGELES AND SANTA BARBARA ARE CITIES AND COUNTIES .
(F9)IN
(F10)IN
(F11)IN
(F12)IN
- (3) NEW YORK IS A CITY,COUNTY, AND STATE.
(F13)IN
(F14)IN
(F15)IN
- (4) CALIFORNIA AND FLORIDA ARE STATES .
(F16)IN
(F17)IN
- (5) DEFINE SB AS SANTA BARBARA.
IN
- (6) DEFINE NY AS NEW YORK.
IN
- (7) DEFINE LA AS LOS ANGELES.
IN

FIGURE 5C—Proper names

- (1) THE POPULATION OF LOS ANGELES IS 2,479,015.
AMBIGUOUS INPUT:((COUNTIES)
AND (COUNTY IS LOS*ANGELES) AND (M10 IS 2479015))
(CITIES) AND (CITY IS LOS*ANGELES) AND (M10 IS 2479015)))
THE POPULATION OF THE CITY OF LOS ANGELES IS 2,479,015.
- (2) IN
- (3) THE MEDIAN FAMILY INCOME IN LA CITY IS \$6,896.
IN
- (4) M12 FOR LA CITY IS 10.1,AND IT IS SMOGGY,WARM,WESTERN, AND LARGE.
IN
- (5) LA CITY HAS AN M13 OF 66.2, M14 OF 10.1, M11 OF 33.2, M16 OF 14.4,
M17 OF 25.1, M18 OF 6,505, M19 OF 5.9, M20 OF 53.4, M21 OF 12.1,
M30 OF 8,149, M31 OF 288,546, M32 OF 1,535,900, M24 OF 25,913,
M25 OF 150,220, M26 OF 496,168, M27 OF 6,993, M28 OF 80,405, M29 OF
422,370, M33 OF 34,467, AND AN M34 OF 18,640.
IN
- (6) SB CITY HAS 58,768 M10, 8.6 M12, 67.8 M13, 16.0 M14, 36.8 M11,
\$6,477 M15, 15.4 M16, 21.1 M17, 146 M18, 12.2 M21, 5.9 M19, 54.5 M20,
102 M30, 2,257 M31, 11,024 M32, 794 M24, 15,983 M26, 4,766 M25,
108 M27, 3,367 M29, 771 M28, 542 M33, 252 M34, P2,P5,AND P7.
IN
- (7) NY CITY HAS 7,781,984 M10, 8.8 M12, 68.7 M13, 10.2 M14, 35.1 M11,
\$6,091 M15, 15.2 M16, 18.5 M17, 17,946 M18, 10.1 M21, 10.5 M19, 37.4
M20,
35,544 M30, 895,838 M31, 4,030,354 M32, 89,663 M24, 1,273,665 M26,
417,343 M25, 28,956 M27, 1,655,907 M29, 296,305 M28, 246,629 M33,
115,363 M34, P1,P3,P4,AND P6.
IN
- (8) ORLANDO HAS 88,135 M10, 10.1 M12, 64.7 M13, 11.9M14, 32.1 M11,
\$5,037 M15, 25.5 M16, 14.8 M17, 173 M18, 12.1 M21, 7.1 M19, 51.5 M20,
157 M30, 3,639 M31, 14,608 M32, 1,373 M24, 26,504 M26, 9,645 M25,
319 M27, 14,679 M29, 3,719 M28, 1,561 M33, 506 M34, P2, P4,AND P7.
IN
- (9) NEW YORK STATE HAS 16,782,304 M10, 6,371 M15, P3,P4,AND P6.
IN
- (10) FLORIDA HAS 4,951,560 M10, 4,722 M15, P2, P4, AND P6.
IN
- (11) CALIFORNIA HAS 15,717,204 M10, 6,726 M15, P5, AND P6.
IN

FIGURE 5D—Specific information

a verbose style or in a highly concise manner reminiscent of the artificial languages used in current data management systems. We believe a natural language subset should admit the concise patterns of expression shown in Sentences 5 through 10, as well as sentences such as 1 through 4.

The compiler response to Sentence 1 illustrates a typical feedback message to the user. In this case an ambiguous input sentence is detected. Rephrasing the sentence as in (2) results in acceptable input.

- (1) A FLIGHT IS AN OBJECT.
PLURAL?
FLIGHTS.
(F18)IN
- (2) PA=25 IS A FLIGHT.
(F19)IN
- (3) FLIGHTS MAY BE JET OR PROP,ECONOMY CLASS OR FIRST CLASS,
NATIONAL OR PAN AM.
(P8 P9 P10 P11 P12 P13)IN
- (4) OHARE IS A PLACE.
(F20)IN
- (5) A FLIGHT DEPARTS FROM A PLACE FOR ANOTHER PLACE.
(F21)IN
- (6) A FLIGHT ARRIVES FROM A PLACE AT ANOTHER PLACE.
(F22)IN
- (7) DEFINE DEPART AS DEPARTS.
IN
- (8) DEFINE ARRIVE AS ARRIVES.
IN
- (9) WHICH PAN AM FLIGHTS THAT ARE ECONOMY CLASS DEPART FOR OHARE
FROM THE CITY OF LOS ANGELES?
<<((PRINT FLIGHT WHERE FLIGHTS AND (PAN AM)
AND (ECONOMY CLASS)
DEPARTS (CITY IS LOS*ANGELES) (PLACE IS OHARE))))>>>
- (10) DOES PA=25 DEPART FROM NEW YORK CITY FOR OHARE?
<<((FLIGHTS AND FLIGHT IS PA=25 DEPARTS (CITY IS NEW*YORK)
(PLACE IS OHARE))))>>>
- (11) WHICH FIRST CLASS JET NATIONAL FLIGHTS ARRIVE FROM LA CITY
AT OHARE?
<<((PRINT FLIGHT WHERE FLIGHTS ARRIVES (CITY IS LOS*ANGELES)
(PLACE IS OHARE) AND (FIRST CLASS) AND (JET) AND (NATIONAL))))>>>
- (12) HOW MANY NATIONAL FLIGHTS DEPART FROM ORLANDO FOR OHARE?
<<((COUNT FLIGHT WHERE FLIGHTS DEPARTS (CITY IS ORLANDO)
(PLACE IS OHARE) AND (NATIONAL))))>>>

FIGURE 6—Extension of the data description to include new types of objects and relations

Extending the data description to encompass new relationships

Figure 6 illustrates a user-computer dialogue in which the range of objects and relations is extended beyond that given earlier in Figure 5. In this case, we are extending the universe of discourse to encompass information similar to that found in airline flight tables.

A new class of objects, "flights," is introduced, and a member of that class, "PA-25," is specified. Several adjectival modifiers are specified and "O'Hare" is then defined as a place. Sentences 5 and 6 illustrate the introduction of new transitive verbs into the English subset—in this case, verbs designating departure and arrival relations. The input of these sentences results in the generation of two new general facts, F21 and F22, that specify the relation names, the domains of the relations, and their codomains. Next, two simple definitional statements are used to establish the equivalence of the singular and plural forms of the two verbs. With the input of this information, the compiler is now able to process and translate questions of the form shown into IL search procedures.

Notice that the expressions "National" and "Pan Am" designate properties which may be true or false for specific flights. This is the simplest means of distinguishing the different kinds of flights. We could have used sentences that would have led to a further increase in the semantic structure. For example, we could have typed in sentences indicating that "Pan Am" and "National" are "airlines" and then the fact that "airlines have flights." This is a simple illustration of an important principle: The level of detail of the data base description is variable and under the control of the person describing the data base.

In the census data base, for example, it is possible to define persons as objects and to add the fact that "cities have people." Then, instead of asking a question such as "What is the population of Los Angeles?" we could ask the question, "What is the number of people in the city of Los Angeles?" The computer would then be directed to count the number of "people" objects in the city.

Getting back to our departure and arrival relations, we can extend the scope of the codomain of an already defined relation such as "depart" to include the notion of departure time. By typing in "a flight departs at a time," a third codomain

argument would be added to the two existing arguments (from place, for place) and the general fact for "depart" would be changed accordingly. We could then ask questions such as "What flights depart from Los Angeles for O'Hare before 7:30 a.m.?" If we wish to ask a question such as "What is the departure time for PA-25?" we can definitionally extend the English subset by typing in "Define departure time as departs at a time." This would establish a nominal to designate the third argument of the codomain of the departure relation and thus the set of departure times.

Deriving semantic structures from syntactic structures

The process of compiling sentences into procedures consists of dictionary lookup, syntax recognition, and three interrelated stages of semantic interpretation: semantic resolution, element construction, and procedure construction. The syntactic recognition process controls the application of semantic interpretation rules in accordance with the grammatical relations that are recognized during syntactic analysis. Just as syntax recognition is a process of recursive decomposition of a sentence into its syntactic categories and words, semantic interpretation can be conceived of as the recursive composition of elementary semantic values and categories into semantic structures. The satisfactory implementation of semantic interpretation processes requires not only a set of semantic interpretation rules but a dictionary structure that is rich enough to represent the complex associations obtaining between English words and the semantic categories and values they designate. Specifically, provision must be made for an English term to take on a number of meanings with respect to the contents of a data base.

The principal information structure for accomplishing sentence-to-procedure translation is the component. Lexical components reside in the dictionary, while derived components are the result of the application of rules to lexical components or to other derived components.

- . A lexical component consists of: a word or idiom, syntactic and semantic categories, and a lexical interpretation list.
- . A derived component consists of: a syntactic structure, and a derived interpretation list.
- . An interpretation list consists of one or more interpretations.

- Each interpretation represents one specific meaning of a word or syntactic structure and consists of:
 - a feature list (F),
 - a selection restriction list (SELR), and sometimes:
 - an element list (E), and
 - a procedure list (P).

An entry in our dictionary consists of a lexical component and additional associated information such as pointers to general or specific facts. This information is stored in LISP as a property list associated with the entry word or idiom.

During dictionary lookup, the hash coding system built into LISP for recognizing primitive symbols (atoms) is used to locate the entries for words. At this time digit sequences are recognized and assigned interpretations as integer or floating point values, as appropriate. The original input string is rewritten to include recognized syntactic categories.

Syntax analysis proceeds in a top-to-bottom, left-to-right manner. List structures are used to represent recognized syntactic elements in derived components. In general, as a pair of syntactic elements is recognized, one or more semantic rules are applied to derive an interpretation list for that pair.

We have discussed the rationale behind the use of interpretation list structures elsewhere (see Kellogg, 1967a). Briefly, each interpretation is an individual meaning-bearing element that may be pointed to by any number of words or syntactic structures. It always consists of at least a list of semantic (and sometimes syntactic) *features* and a *selection restriction*. These constructs generally perform the same functions as the corresponding concepts in the theories of Katz (1966) and Chomsky (1965). By themselves, features and selection restrictions may represent the sense of an English word and the contexts or environments in which that sense can correctly occur. For example, the meanings of certain prepositions, adverbs, and determiners in the English subset are adequately represented in this manner. However, if an interpretation must also designate semantic categories and values, an element list is required. Finally, as elements are combined to form parts of a procedure, they must be moved to a procedure list.

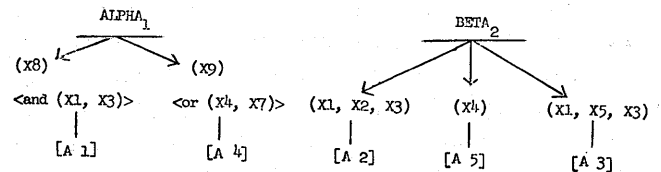
Since only a part of the burden of semantic in-

terpretation is placed on the use of semantic features (their use is restricted to indications of simple class membership, class inclusion, agreement, etc.), we have not found it difficult in practice to assign features during the data description process. Our experience in using interpretation lists confirms their value as a good solution to the problem of distributing semantic information between the dictionary and rules. Imposing a heavier semantic load on features can lead to a number of difficulties (see Bolinger, 1965), while dictionary entries of significantly simpler structure can result in the need for a very much larger set of semantic interpretation rules.

Semantic resolution rules apply to components that stand in a specific grammatical relationship to one another. Their function can be understood by considering the application of the following rule:

$SAT [F_2, SELR_1] \rightarrow (UNION [F_1, F_2], SELR_2, UNION [E_1, E_2])$

to the following components:



The rule applies if the word "alpha" grammatically modifies the word "beta" and the left half or pattern part of the rule evaluates to "true." These rules have a "pattern \rightarrow operation" format. The pattern part of a rule consists of predicates that return either true or false values. The right half consists of symbols and operations over symbols that construct a derived interpretation. In the above example, the first component has two interpretations, each having a list of features (enclosed in parentheses), a selection restriction (a Boolean function of features enclosed in angle brackets) and an element list (enclosed in square brackets). The "beta" component has three interpretations (selector restrictions are not shown in this case in order to simplify the example).

The rule applies to every possible combination of "alpha" and "beta" interpretations, and if the left half of the rule returned "true" in each case, then six derived interpretations would result. In this example, only three interpretations are allowed by the resolution rule.

The derived component is:

The semantic rule has resolved a possible sixfold ambiguity into a threefold ambiguity. The other three possible interpretations were rejected by the failure of the "SAT" predicate to match the selection restriction of an "alpha" interpretation (SELR₁) to the features list of a "beta" interpretation (F₂).

In general, resolution rules allow us to check for compatibility or agreement between interpretations and to map pairs of compatible interpretations into derived interpretations. Element lists, when they occur, are usually combined by a simple union operation as shown.

Two additional sets of rules must be applied to derived interpretations, in sequence, in order to construct well-formed semantic structures. Element construction rules combine the symbols in an element list into well-formed elements as specified by the formation rules E₀ - E₄ in Figure 3. Similarly, procedure construction rules move elements to the procedure list (P-list) in accordance with intermediate language formation rules, insuring the production of only well-formed procedures.

The feature, selection restriction, element, and procedure lists, in conjunction with the three kinds of semantic interpretation rules, constitute a combinatorial apparatus of considerable power and flexibility. Our experience so far indicates that such a system of information structures and rules is adequate for the processing of a fairly wide class of English sentences, and furthermore, that a translation procedure based on such structure and rules is reasonably efficient. (The version of the CONVERSE system in current use on the Q-32 Time-Sharing System at SDC typically requires from less than one second to as much as three seconds of computer processing time to effect sentence-to-procedure translation.)

An example of question-to-procedure translation

Figure 7 illustrates the syntactic structure that is assigned by the compiler for the question "Which Pan Am flights that are economy class de-

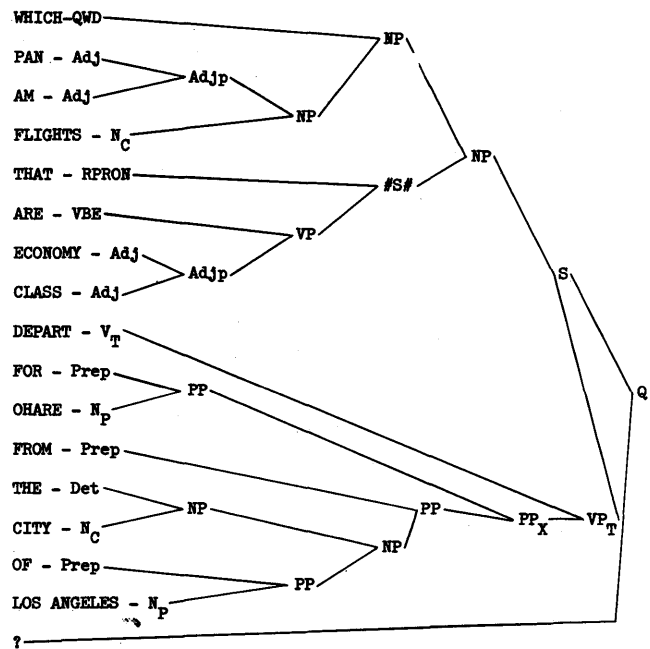


FIGURE 7—Compiler-assigned syntactic structure for a sample question

part for O'Hare from the City of Los Angeles?" Figures 8 and 9 illustrate the syntax rules for this question, the applicable resolution (CR), element (ER), and procedure (PR) rules, and the actions of semantic rules that result in the construction of a well-formed procedure.

Rules CR₁ and ER₁ are applied to each adjective phrase to test for the features "SETP" and "PTER." Where these conditions are met ("Pan

Syntax Rules	Semantic Rules	Resultant P-LIST
SR ₁ : Adj Adj → AdjP	CR ₁ , ER ₁	
SR ₂ : AdjP NC → NP	CR ₂ , PR ₂	(and P ₁)
SR ₃ : Qwd NP → NP	CR ₂	
SR ₄ : Rpron VP → #S#	CR ₂ , PR ₂	(and P ₂ and P ₁)
SR ₅ : NP #S# → NP	CR ₃ , PR ₁	((flights) and P ₂ and P ₁)
SR ₆ : Prep NP → PP	CR ₂	
SR ₇ : Det N _C → NP	CR ₂	
SR ₈ : VB AdjP → VP	CR ₂	
SR ₉ : PP PP* → PP _X	CR ₄	
SR ₁₀ : V _T PP _X → VP _T	CR ₅ , ER ₂ , PR ₃	((flights) and P ₂ and P ₁ (departs (City of Los Angeles) (place is O'hare)))
SR ₁₁ : NP VP _T → S	CR ₆ , PR ₄ , PR ₅ , PR ₆	Print flight where ((flights) and P ₂ and P ₁ (departs (City is Los Angeles) (place is O'hare)))
SR ₁₂ : S '!' → Q		

FIGURE 8—Syntactic and semantic rules applied to the sample question

$CR_1: \text{setp} \in F_1 \wedge \text{setp} \in F_2 \wedge \text{ptcr} \in \text{SELR}_2 \Rightarrow \text{SUBST}(\text{setpt}, \text{setp}, \text{UNION}(F_1, F_2)), \text{SELR}_2$
 $CR_2: \text{SAT}(F_2, \text{SELR}_1) \Rightarrow \text{UNION}(F_1, F_2), \text{SELR}_2$
 $CR_3: \text{SAT}(F_1, \text{SELR}_2) \Rightarrow \text{UNION}(F_1, F_2), \text{SELR}_1$
 $CR_4: \text{ADJOIN}(I_1, I_2, \dots, I_N)$
 $CR_5: \text{CDSAT}(I_2, \text{SELR}_1) \Rightarrow F_1, \text{SELR}_1$
 $CR_6: \text{DSAT}(I_1, \text{SELR}_2) \Rightarrow F_2, \text{SELR}_2$

$ER_1: \text{SINGLE}[E_1, E_2] \Rightarrow E_1$
 $ER_2: E_1 \wedge E_2 \Rightarrow \text{CON}[E_1, \text{CD}[E_2]]$

$PR_1: X \in \text{SET} \Rightarrow \text{SUBST}[E_1, X, E], \text{ADD}[X, P]$
 $PR_2: X \in P \Rightarrow \text{SUBST}[E_2, X, E], \text{ADD}[\text{and } X, P]$
 $PR_3: X \in E_3 \Rightarrow \text{SUBST}[E_3, X, E], \text{ADDR}[X, P]$
 $PR_4: X \in \text{OP}_V \wedge X \in \text{TERM} \Rightarrow \text{SUBST}[\text{PROP}, X, E], \text{ADD}[\text{where}]$
 $PR_5: X \in \text{TERM} \Rightarrow \text{SUBST}[\text{TERM}, X, E] \text{ ADD}[X, P]$
 $PR_6: X \in \text{OP} \Rightarrow \text{SUBST}[\text{PROCEDURE}, X, E] \text{ ADD}[X, P]$

FIGURE 9—Semantic resolution, element, and procedure rules

Am," "Economy Class") we substitute the feature "SETPT" for "SETP" in the derived feature list. This indicates that a phrase has been identified as designating a property and rule ER_1 is applied to test for the designation of a single unique property (specified by its P-name). A P-rule is first applied in conjunction with syntax rule SR_2 . Each P-rule has as its pattern part a set membership test. For rule PR_2 , a Pp element or P-name found in an element list is replaced by the name of the IL formation rule (E_2) that dominates it and the P-name is added to the P-list. This procedure allows us to keep track of the level of the semantic categories as well as to combine them, as necessary, in the P-list.

Rules $CR_{4,5}$ and 6 , ER_2 , and PR_3 are used to test the object and subject of the transitive verb and to construct a list of proper arguments for the designated relation. CR_4 combines the several interpretations of prepositional phrases into a single list. The CDSAT predicate in Rule CR_5 applies the selection restrictions of the verb "depart" to the interpretation list formed by Rule CR_4 . The selection restriction for this verb consists of a pointer to the general fact representing the relational meaning of the verb. In this case, the general fact has the form: (flight.X) (departs) (place.Y) (place.Z) and the result of the first application of CR_5 and ER_2 is the element list ((de-

parts), (), (place is O'Hare)), where the empty list stands for the missing argument: (place.Y). The second application of these rules (to the interpretation of "from the city of Los Angeles") results in the filling in of the missing argument. Finally, Rule PR_3 appends the relation name and its codomain arguments to the end of the P-list. The last steps in compilation include a test for subject-verb agreement and the addition of a "print" operation to the P-list.

Several properties of the semantic rules are of significance. We notice that with few exceptions (i.e., SETP, PTER) the rules are formulated independently of specific features. Were this not the case, we would run the risk of requiring new rules every time we described or modified the semantic structure of a universe of discourse. Secondly, only a small number of predicates and operations are required to formulate the rules (only a few more than shown here are required in the complete system). Thirdly, only a relatively modest number of semantic rules are required (the present system requires less than three times as many semantic rules as are shown in Figure 9).

Finally, though not illustrated here, the semantic rules always correctly distribute ambiguous or multiple requests to separate procedures.

The data management machine

The data management machine is at present implemented as a series of LISP procedures. Though only a modest number of facts can be handled within LISP, the information structures used are both simple and general enough to form the basis for a large-scale storage system.

Three kinds of facts are allowed: A *simple* fact expresses a relation among specific objects. For example, the list:

((departs), (flights.PA-25), (city.Los Angeles), (place.O'Hare), (time.1730))

represents the fact that flight PA-25 departs from Los Angeles for O'Hare at 5:30 p.m. Many requests can be answered by consulting *compound* facts, in which individual objects are characterized by a vector of elements such as:

((cities), (city.Los Angeles), P_1 , ($M_1.40-000$), (date.680427)).

This is an example of a compound fact that states

```
(1) WHAT IS?
<<<(PRINT)>>>
(2) WHAT IS POP?
<<<(PRINT (POPULATION))>>>
(3) WHAT ABOUT LA?
<<<(PRINT WHERE COUNTIES AND COUNTY IS LOS*ANGELES)
(PRINT WHERE CITIES AND CITY IS LOS*ANGELES)>>>
ANSWER1.
(COUNTIES ARE OBJECTS)(COUNTY IS LOS*ANGELES)

ANSWER2.
(CITIES ARE OBJECTS)
(CITY IS LOS*ANGELES)
(POPULATION IS 2479015)
(MEDIAN FAMILY INCOME IS 6896)
(PERCENTAGE OF PEOPLE UNDER 5 YEARS OF AGE IS 10.099999999)
(SMOGGY IS TRUE)
(WARM IS TRUE)
(WESTERN IS TRUE)
(LARGE IS TRUE)
(PERCENTAGE OF PEOPLE OVER 21 YEARS OF AGE IS 66.199999999)
(PERCENTAGE OF PEOPLE OVER 65 YEARS OF AGE IS 10.099999999)
(MEDIAN AGE OF PEOPLE IS 33.200000000)
(PERCENTAGE OF FAMILY INCOME UNDER 3000 IS 14.399999999)
(PERCENTAGE OF FAMILY INCOME OVER 10000 IS 25.100000000)
(AGGREGATE FAMILY INCOME IN MILLIONS OF DOLLARS IS 6505)
(PERCENTAGE OF PEOPLE OVER 25 WITH LESS*THAN 5 YEARS OF SCHOOLING IS
5.900000000)
(PERCENTAGE OF PEOPLE OVER 25 WITH MORE*THAN 12 YEARS OF SCHOOLING IS
53.399999999)
(MEDIAN NUMBER OF SCHOOL YEARS COMPLETED IS 12.099999999)
(NUMBER OF MANUFACTURING ESTABLISHMENTS IS 8149)
(NUMBER OF MANUFACTURING EMPLOYEES IS 288546)
(MANUFACTURING PAYROLL IN THOUSANDS OF DOLLARS IS 1535900)
(NUMBER OF RETAIL TRADE ESTABLISHMENTS IS 25913)
(NUMBER OF RETAIL TRADE EMPLOYEES IS 158226)
(RETAIL TRADE YEARLY PAYROLL IN THOUSANDS OF DOLLARS IS 496168)
(NUMBER OF WHOLESALE TRADE ESTABLISHMENTS IS 6993)
(NUMBER OF WHOLESALE TRADE EMPLOYEES IS 80405)
(WHOLESALE TRADE YEARLY PAYROLL IN THOUSANDS OF DOLLARS IS 422370)
(NUMBER OF LOCAL GOVERNMENT EMPLOYEES IS 34467)
(LOCAL GOVERNMENT PAYROLL IN THOUSANDS OF DOLLARS IS 18640)

(4) WHAT ARE THE CITIES?
<<<(PRINT CITY WHERE CITIES)>>>
ANSWER.
(CITY IS ORLANDO)
(CITY IS LOS*ANGELES)
(CITY IS SANTA*BARBARA)
(CITY IS NEW*YORK)
```

FIGURE 10—Elementary question patterns

that on the 27th of April 1968 property P_1 was true and function M_1 had a value of 40,000 for the city of Los Angeles. Only one date and/or time

```
(1) WHAT IS THE POP FOR LA AND NY CITIES?
<<<(PRINT CITY (POPULATION) WHERE CITIES AND CITY IS LOS*ANGELES)
(PRINT CITY (POPULATION) WHERE CITIES AND CITY IS NEW*YORK)>>>
(2) WHAT IS THE POP FOR WESTERN CITIES AND STATES?
<<<(PRINT CITY (POPULATION) WHERE CITIES AND (WESTERN))
(PRINT STATE (POPULATION) WHERE STATES AND (WESTERN))>>>
(3) WHAT ARE THE PRODUCTION FACTORS AND DISTRIBUTION FACTORS
FOR SMALL WESTERN CITIES?
<<<(PRINT CITY (NUMBER OF MANUFACTURING ESTABLISHMENTS)
(NUMBER OF MANUFACTURING EMPLOYEES)
(MANUFACTURING PAYROLL IN THOUSANDS OF DOLLARS)
(NUMBER OF RETAIL TRADE ESTABLISHMENTS)
(NUMBER OF RETAIL TRADE EMPLOYEES)
(RETAIL TRADE YEARLY PAYROLL IN THOUSANDS OF DOLLARS)
(NUMBER OF WHOLESALE TRADE ESTABLISHMENTS)
(NUMBER OF WHOLESALE TRADE EMPLOYEES)
(WHOLESALE TRADE YEARLY PAYROLL IN THOUSANDS OF DOLLARS)
WHERE CITIES AND (SMALL) AND (WESTERN))>>>
ANSWER.
(CITY IS SANTA*BARBARA)
(NUMBER OF MANUFACTURING ESTABLISHMENTS IS 102)
(NUMBER OF MANUFACTURING EMPLOYEES IS 2257)
(MANUFACTURING PAYROLL IN THOUSANDS OF DOLLARS IS 11024)
(NUMBER OF RETAIL TRADE ESTABLISHMENTS IS 794)
(NUMBER OF RETAIL TRADE EMPLOYEES IS 4766)
(RETAIL TRADE YEARLY PAYROLL IN THOUSANDS OF DOLLARS IS 15983)
(NUMBER OF WHOLESALE TRADE ESTABLISHMENTS IS 108)
(NUMBER OF WHOLESALE TRADE EMPLOYEES IS 771)
(WHOLESALE TRADE YEARLY PAYROLL IN THOUSANDS OF DOLLARS IS 3367)

(4) WHAT IS EASTERN CITY AND COUNTY M22 AND M23?
<<<(PRINT CITY (NUMBER OF RETAIL TRADE ESTABLISHMENTS)
(NUMBER OF RETAIL TRADE EMPLOYEES)
(RETAIL TRADE YEARLY PAYROLL IN THOUSANDS OF DOLLARS)
(NUMBER OF WHOLESALE TRADE ESTABLISHMENTS)
(NUMBER OF WHOLESALE TRADE EMPLOYEES)
(WHOLESALE TRADE YEARLY PAYROLL IN THOUSANDS OF DOLLARS)
WHERE CITIES AND (EASTERN))
(PRINT COUNTY (NUMBER OF RETAIL TRADE ESTABLISHMENTS)
(NUMBER OF RETAIL TRADE EMPLOYEES)
(RETAIL TRADE YEARLY PAYROLL IN THOUSANDS OF DOLLARS)
(NUMBER OF WHOLESALE TRADE ESTABLISHMENTS)
(NUMBER OF WHOLESALE TRADE EMPLOYEES)
(WHOLESALE TRADE YEARLY PAYROLL IN THOUSANDS OF DOLLARS)
WHERE COUNTIES AND (EASTERN))>>>
```

FIGURE 11—Compound requests-I

```
(1) HOW MANY CITIES HAVE A POPULATION THAT IS LARGER THAN 500,000 AND
A MEDIAN FAMILY INCOME THAT IS SMALLER THAN $6500 ?
<<<(COUNT CITY WHERE CITIES AND (POPULATION)
IS*GR*THAN 500000 AND (MEDIAN FAMILY INCOME) IS*LS*THAN 6500)>>>
ANSWER.
1
(2) WHAT IS THE POPULATION AND THE NUMBER OF LOCAL GOVERNMENT
EMPLOYEES FOR EACH CITY ?
<<<(PRINT CITY (POPULATION)
(NUMBER OF LOCAL GOVERNMENT EMPLOYEES) WHERE CITIES)>>>
BRIEF.
((CITY . ORLANDO) (M10 . 88135) (M33 . 1561))
((CITY . LOS*ANGELES) (M10 . 2479015) (M33 . 34467))
((CITY . SANTA*BARBARA) (M10 . 58768) (M33 . 542))
((CITY . NEW*YORK) (M10 . 7781984) (M33 . 246629))
(3) WHAT PLACES HAVE THE MOST AND LEAST POPULATION?
<<<(PRINT PLACE MAX (POPULATION) WHERE PLACES)
(PRINT PLACE MIN (POPULATION) WHERE PLACES)
(PRINT CITY MAX (POPULATION) WHERE CITIES)
(PRINT CITY MIN (POPULATION) WHERE CITIES)
(PRINT COUNTY MAX (POPULATION) WHERE COUNTIES)
(PRINT COUNTY MIN (POPULATION) WHERE COUNTIES)
(PRINT STATE MAX (POPULATION) WHERE STATES)
(PRINT STATE MIN (POPULATION) WHERE STATES)>>>
(4) WHAT IS THE HIGHEST AND LOWEST MEDIAN FAMILY INCOME FOR
THE EASTERN CITIES AND FOR THE WESTERN STATES ?
<<<(PRINT CITY MAX (MEDIAN FAMILY INCOME)
WHERE CITIES AND (EASTERN))
(PRINT CITY MIN (MEDIAN FAMILY INCOME) WHERE CITIES AND (EASTERN))
(PRINT STATE MAX (MEDIAN FAMILY INCOME) WHERE STATES AND (WESTERN))
(PRINT STATE MIN (MEDIAN FAMILY INCOME) WHERE STATES AND (WESTERN))>>>
(5) WHICH WESTERN CITY HAS THE LARGEST POPULATION?
<<<(PRINT CITY MAX (POPULATION) WHERE CITIES AND (WESTERN))>>>
(6) WHAT ARE THE TWO CITIES WITH THE SMALLEST POPULATION?
<<<(PRINT CITY 1 2 MIN (POPULATION) WHERE CITIES)>>>
(7) WHAT IS THE THIRD LARGEST WESTERN SMOGGY CITY IN POPULATION?
<<<(PRINT CITY 3 MAX (POPULATION)
WHERE CITIES AND (WESTERN) AND (SMOGGY))>>>
(8) WHAT ARE THE SECOND AND THIRD LARGEST WESTERN CITIES IN MEAN ANNUAL
PRECIPITATION?
<<<(PRINT CITY 2 MAX (MEAN ANNUAL PRECIPITATION)
WHERE CITIES AND (WESTERN))
(PRINT CITY 3 MAX (MEAN ANNUAL PRECIPITATION)
WHERE CITIES AND (WESTERN))>>>
```

FIGURE 12—Compound requests-II

value is allowed for each compound fact and a new compound fact is constructed for each description of an object at a different point in time. A compound fact becomes *complex* when the object it describes enters into a relationship with other objects. An example is:

((flights), (flight.PA-25), (departs, F_5 , F_{10})).

This fact links all of the information about flight PA-25 to those members of the "departs" relation that reference PA-25 as domain elements. In this case the members are the simple facts given the fact names: F_5 and F_{10} .

These three information structures permit us to describe and interrelate objects to virtually any desired degree of complexity.*

The data management machine offers a significant increase in the range of possible universes of discourse as contrasted with, for example, the BASEBALL system (Green, 1963) which was limited to the description of specific baseball games and the SIR system (Raphael, 1964), which was limited to the recognition of a small number of relationships.

*It is readily seen that simple and compound facts express atomic and molecular propositions respectively (see Travis, 1963). A complex fact represents a molecular proposition that associates the object descriptions that participate in the R-image for a specified object and relation.

FOR THE SMOGGY HIGH-INCOME CITIES WHAT IS THE AGE-INCOME VALUE-RANGE?

1. <<((PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF PEOPLE UNDER 5 YEARS OF AGE) IS*LS*THAN 10)
2. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF PEOPLE UNDER 5 YEARS OF AGE) IS*GR*THAN 10 AND (PERCENTAGE OF PEOPLE UNDER 5 YEARS OF AGE) IS*LS*THAN 15)
3. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF PEOPLE UNDER 5 YEARS OF AGE) IS*GR*THAN 15 AND (PERCENTAGE OF PEOPLE UNDER 5 YEARS OF AGE) IS*LS*THAN 20)
4. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF PEOPLE UNDER 5 YEARS OF AGE) IS*GR*THAN 20)
5. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF PEOPLE OVER 21 YEARS OF AGE) IS*LS*THAN 10)
6. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF PEOPLE OVER 21 YEARS OF AGE) IS*GR*THAN 10 AND (PERCENTAGE OF PEOPLE OVER 21 YEARS OF AGE) IS*LS*THAN 15)
7. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF PEOPLE OVER 21 YEARS OF AGE) IS*GR*THAN 15 AND (PERCENTAGE OF PEOPLE OVER 21 YEARS OF AGE) IS*LS*THAN 20)
8. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF PEOPLE OVER 21 YEARS OF AGE) IS*GR*THAN 20)
9. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF PEOPLE OVER 65 YEARS OF AGE) IS*LS*THAN 10)
10. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF PEOPLE OVER 65 YEARS OF AGE) IS*GR*THAN 10 AND (PERCENTAGE OF PEOPLE OVER 65 YEARS OF AGE) IS*LS*THAN 15)
11. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF PEOPLE OVER 65 YEARS OF AGE) IS*GR*THAN 15 AND (PERCENTAGE OF PEOPLE OVER 65 YEARS OF AGE) IS*LS*THAN 20)
12. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF PEOPLE OVER 65 YEARS OF AGE) IS*GR*THAN 20)
13. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF FAMILY INCOME UNDER 3000) IS*LS*THAN 10)
14. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF FAMILY INCOME UNDER 3000) IS*GR*THAN 10 AND (PERCENTAGE OF FAMILY INCOME UNDER 3000) IS*LS*THAN 15)
15. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF FAMILY INCOME UNDER 3000) IS*GR*THAN 15 AND (PERCENTAGE OF FAMILY INCOME UNDER 3000) IS*LS*THAN 20)
16. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF FAMILY INCOME UNDER 3000) IS*GR*THAN 20)
17. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF FAMILY INCOME OVER 10000) IS*LS*THAN 10)
18. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF FAMILY INCOME OVER 10000) IS*GR*THAN 10 AND (PERCENTAGE OF FAMILY INCOME OVER 10000) IS*LS*THAN 15)
19. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF FAMILY INCOME OVER 10000) IS*GR*THAN 15 AND (PERCENTAGE OF FAMILY INCOME OVER 10000) IS*LS*THAN 20)
20. (PRINT CITY WHERE CITIES AND (SMOGGY) AND (MEDIAN FAMILY INCOME) IS*GR*THAN 6000 AND (PERCENTAGE OF FAMILY INCOME OVER 10000) IS*GR*THAN 20))>>

BRIEF.

1. (((CITY . NEW*YORK)))
2. (((CITY . LOS*ANGELES)))
3. NIL
4. NIL
5. NIL
6. NIL
7. NIL
8. (((CITY . LOS*ANGELES)) ((CITY . NEW*YORK)))
9. NIL
10. (((CITY . LOS*ANGELES)) ((CITY . NEW*YORK)))
11. NIL
12. NIL
13. NIL
14. (((CITY . LOS*ANGELES)))
15. (((CITY . NEW*YORK)))
16. NIL
17. NIL
18. NIL
19. (((CITY . NEW*YORK)))
20. (((CITY . LOS*ANGELES)))

DEFINE HIGH-INCOME AS A MEDIAN FAMILY INCOME EXCEEDING \$6,000.
 IN
 DEFINE AGE-INCOME AS PERCENTAGE OF PEOPLE UNDER 5 YEARS OF AGE OR PERCENTAGE OF PEOPLE OVER 21 YEARS OF AGE OR PERCENTAGE OF PEOPLE OVER 65 YEARS OF AGE OR PERCENTAGE OF FAMILY INCOME UNDER \$3,000 OR PERCENTAGE OF FAMILY INCOME OVER \$10,000.
 IN
 DEFINE VALUE-RANGE AS UNDER 10 OR BETWEEN 10 AND 15 OR BETWEEN 15 AND 20 OR OVER 20.
 IN

FIGURE 13—The use of definitions in phrasing complex requests

Examples of data base interrogation

Figures 10 through 16 illustrate some of the interrogation patterns acceptable to the CONVERSE system at present. Figure 10 illustrates some elementary question patterns. Question 1 represents perhaps the simplest, most general (and least interesting) question we can ask, as it translates into a request consisting only of a

- (1) WHAT DOES M33 STAND FOR?
 UNDEFINED WORDS/ST (STAND)
- (2) YOU MAY DEFINE TERMS OR REPHRASE YOUR SENTENCE.
 WHAT DOES M33 MEAN?
 UNDEFINED STRUCTURE:
 NPL(WHAT)
 VERB(DOES)
 NP(M33)
 NP(MEAN)
- (3) WHAT IS M33?
- (4) <<((PRINT (NUMBER OF LOCAL GOVERNMENT EMPLOYEES))>>
 WHAT ABOUT LA?
 <<((PRINT WHERE COUNTIES AND COUNTY IS LOS*ANGELES)
 (PRINT WHERE CITIES AND CITY IS LOS*ANGELES))>>
 WHAT ABOUT MEDIAN FAMILY INCOME?
 <<((PRINT (MEDIAN FAMILY INCOME))>>
 WHAT ABOUT POP FACTORS?
- (6) <<((PRINT (MEDIAN AGE OF PEOPLE)
 (PERCENTAGE OF PEOPLE UNDER 5 YEARS OF AGE)
 (PERCENTAGE OF PEOPLE OVER 21 YEARS OF AGE)
 (PERCENTAGE OF PEOPLE OVER 65 YEARS OF AGE)
 (MEDIAN FAMILY INCOME)
 (PERCENTAGE OF FAMILY INCOME UNDER 3000)
 (PERCENTAGE OF FAMILY INCOME OVER 10000)
 (AGGREGATE FAMILY INCOME IN MILLIONS OF DOLLARS)
 (PERCENTAGE OF PEOPLE OVER 25 WITH LESS*THAN 5 YEARS OF SCHOOLING)
 (PERCENTAGE OF PEOPLE OVER 25 WITH MORE*THAN 12 YEARS OF SCHOOLING)
 (MEDIAN NUMBER OF SCHOOL YEARS COMPLETED) (POPULATION))>>
 (7) WHAT ABOUT PERCENTAGE OF PEOPLE?
 UNDEFINED MEANING:
 WHAT
 (ABOUT (PERCENTAGE (OF PEOPLE)))
 <<NIL>>
- (8) WHAT ABOUT PERCENTAGE OF PEOPLE MEASURES?
 <<((PRINT (PERCENTAGE OF PEOPLE OVER 25 WITH MORE*THAN 12 YEARS OF SCHOOLING)
 (PERCENTAGE OF PEOPLE OVER 25 WITH LESS*THAN 5 YEARS OF SCHOOLING)
 (PERCENTAGE OF PEOPLE OVER 65 YEARS OF AGE)
 (PERCENTAGE OF PEOPLE OVER 21 YEARS OF AGE)
 (PERCENTAGE OF PEOPLE UNDER 5 YEARS OF AGE))>>

FIGURE 14—Accessing general information

```

(1) DO ANY LARGE SMOGGY EASTERN CITIES EXIST?
  <<<(CITIES AND (LARGE) AND (SMOGGY) AND (EASTERN))>>>
  ANSWER.
  YES
(2) LET LARGE INCLUDE A POPULATION THAT IS OVER 500,000 AND A QUANTITY
  OF MANUFACTURING ESTABLISHMENTS THAT IS OVER 8,000.
  IN
(3) WHAT ARE THE LARGE SMOGGY EASTERN CITIES?
  <<<(PRINT CITY WHERE CITIES AND (LARGE) AND (SMOGGY) AND (EASTERN))
  (PRINT CITY WHERE CITIES AND (SMOGGY)
  AND (EASTERN)
  AND (POPULATION)
  IS<GR>THAN 500000 AND (NUMBER OF MANUFACTURING ESTABLISHMENTS)
  IS<GR>THAN 8000)>>>
  ANSWER1.
  (CITY IS NEW*YORK)
  ANSWER2.
  (CITY IS NEW*YORK)
(4) FORGET LARGE.
(5) LET LARGE BE POP OVER 500,000 AND NUMBER OF MANUFACTURING
  ESTABLISHMENTS OVER 8,000.
  IN
(6) WHAT ARE THE LARGE SMOGGY EASTERN CITIES?
  <<<(PRINT CITY WHERE CITIES AND (SMOGGY)
  AND (EASTERN)
  AND (POPULATION)
  IS<GR>THAN 500000 AND (NUMBER OF MANUFACTURING ESTABLISHMENTS)
  IS<GR>THAN 8000)>>>
  ANSWER.
  (CITY IS NEW*YORK)

```

FIGURE 15—Modifying the “meaning” of terms

“print” operation. Execution of a file-searching procedure of this kind would result in a listing of all the values in the complete data base. Similarly, for Question 2, a listing of all of the population values associated with objects in the data base would result. Question 3 compiles into two separate requests. The answer to the first request (ANSWER1) simply confirms the fact that Los Angeles is the name of a specific county since we have stored no additional specific data for the county of Los Angeles. ANSWER2, however, yields the complete record of specific information for the city of Los Angeles. The last question illustrates a simple phrasing of a set membership request.

Figure 11 illustrates some of the patterns for combining and coordinating terms into phrases and sentences. Sentence 1, for example, illustrates how two coordinated proper nouns may be post-modified by a common noun. The proper nouns are distributed into two separate requests, since a single request calling for the population of a “Los Angeles and New York” object would be absurd. Similarly, Sentence 2 results in two requests where the common nouns are both pre-modified by the same adjective. Sentence 3 represents a somewhat more involved coordination pattern as two noun phrases designating generic measures are post-modified by the same prepositional phrase. A single, rather lengthy, request results for this question type. In Sentence 4 two coordinated common nouns are first postmodified

```

(1) A STUDENT IS AN OBJECT.
  PLURAL?
  STUDENTS.
  (F1)IN
(2) STUDENTS HAVE PARENTS.
  (M1)IN
(3) PARENTS MAY BE SEPARATED OR DIVORCED.
  (V1 V2)IN
(4) STUDENTS MAY BE A7 OR B7, MADISON JR HIGH SCHOOL OR COLUMBUS
  JR HIGH SCHOOL.
  (P1 P2 P3 P4)IN
(5) STUDENTS HAVE SCHOLASTIC CAPACITY THAT INCLUDES CTMM AND
  CAT) SCHOLASTIC ACHIEVEMENT THAT INCLUDES READING VOCABULARY
  AND READING COMPREHENSION) SCHOLASTIC PERFORMANCE THAT
  INCLUDES ENGLISH GRADE, MATH GRADE.
  (M3 M4)IN
  (M6 M7)IN
  (M2 M5 M8)IN
(6) SCHOLASTIC PERFORMANCE MAY BE A, B, C, D, OR F.
  (V3 V4 V5 V6 V7)IN
(7) JOE BROWN AND MARY SMITH ARE STUDENTS.
  (F2)IN
  (F3)IN
(8) DEFINE COHI AS COLUMBUS JK HIGH SCHOOL.
  IN
(9) DEFINE MADHI AS MADISON JR HIGH SCHOOL.
  IN
(10) JOE BROWN ON 04/18/68 WAS A COHI STUDENT AND HE HAD A CTMM OF 8.
  A CAT OF 9, AND A READING VOCABULARY OF 72.
  IN
(11) JOE BROWN IS A B7.
  IN
(12) MARY SMITH ON 3/12/68 WAS A MADHI STUDENT AND SHE HAD A CTMM OF 5 AND
  A CAT OF 7.
  IN
  DEFINE SCORES AS CTMM, CAT, READING VOCABULARY.
  IN
(13) WHAT WAS THE NUMBER OF STUDENTS BETWEEN 3/1/68 AND 5/1/68?
  <<<(COUNT STUDENT WHERE STUDENTS AND DATE IS<GR>THAN 680301 AND DATE
  ANSWER.
  2
(14) WHAT IS THE SCHOLASTIC ACHIEVEMENT FOR JOE BROWN?
  <<<(PRINT (READING VOCABULARY)
  (READING COMPREHENSION) WHERE STUDENTS AND STUDENT IS JOE*BROWN)>>>
  ANSWER.
  (READING VOCABULARY IS 72)
(15) WHAT ARE THE SCORES FOR MARY SMITH?
  <<<(PRINT (CTMM)
  (CAT)
  (READING VOCABULARY) WHERE STUDENTS AND STUDENT IS MARY*SMITH)>>>
  ANSWER.
  (CTMM IS 5) (CAT IS 7)

```

FIGURE 16—Describing and interrogating an educational data base

by a compound consisting of two M-names and then premodified by an adjective. This results in the distribution of the terms into two separate requests concerning cities and counties respectively.

Further variety in coordination patterns and additional operations on data are illustrated by the questions in Figure 12. Sentence 1 illustrates the use of the coordination of noun phrases containing embedded relative clauses. For the second question a “brief” form of answer is called for. This causes the printout of the found facts in the same format in which they are stored in computer memory: as lists of LISP dotted pairs. The other examples in this figure illustrate the use of superlative adjectives to modify nouns, and the use of determiners denoting cardinal and ordinal numbers. The resulting requests specify compound operations to be carried out in sequence on the selected data. For example, the “max” operation is defined only for functions that evaluate to numbers. If this operation is preceded by a list of one or more numbers, then the operation is carried out for each numeral on the list. In this way, the “two

largest cities in population" are identified by finding in turn the city with the largest population value and then the city with the next largest population value.

The many devices for compounding and coordination in natural language allow us, in a data management context, to quite easily phrase questions that compile either into complex requests or into a large number of individual requests. As an illustration of the latter case, consider the question in Figure 13: "For the smoggy high-income cities what is the age-income value-range?" In order to derive the necessary data from the data base, twenty requests in the intermediate language result from the compilation of this question. Some of the potential of, and justification for, research work in natural language data processing is evident from this example. The user's alternative to the phrasing of his question in English is to type, in some artificial query language, twenty requests comparable to those shown, or some highly complex nested Boolean statement that is the equivalent of these requests. The computer response (in brief form) allows us to see quickly that requests 1, 15, and 19 are satisfied by the city of New York; 2, 14, and 20 are satisfied by Los Angeles, and that requests 8 and 10 are satisfied by both the city of New York and the city of Los Angeles (future versions of the system will generate answer statements from requests and their non-nil responses). Definitional sentences to specify appropriate concepts of high income, distribution of people according to age and income, and a range of percentage values of interest to the user are shown at the bottom of Figure 13.

Figure 14 illustrates several kinds of feedback messages that may result from user interrogation of the semantic structure of the universe of discourse. Suppose, for example, that a user wants to know the normative descriptive phrase associated with a specific M-name. He might first try a question such as Question 1 which leads, as we see, to an "undefined word" feedback response. Similarly, if he rephrases the question as in Sentence 2, then all words are defined but are not put together in an acceptable syntactic structure. Finally, if he phrases his request as in Sentence 3, he gets a simple direct response as a procedure specifying the normative description phrase for the M-name.

If he is concerned about the referent for a particular noun phrase, a "what about" question, as expressed in Sentence 4, will generate a separate procedure for each referred object. The response to Question 5 indicates that the noun phrase "medium family income" is well formed and designates, in this case, a specific measure function. Similarly in Sentence 5, we see that the phrase "pop (population) factors" designates a list of specific measure functions. In Sentence 7, we see that the constituent "percentage of people," while syntactically well formed, does not evaluate to an object, function, or property, in the intermediate language. Hence, its meaning is undefined and an appropriate feedback message results. However, we may rephrase this question as in Sentence 8, in order to find all measures that are directly or indirectly linked to the phrase.

Figure 15 illustrates one means of changing and extending the meaning of English subset terms. Sentence 1 illustrates the use of "large" as this term was specified during data description. The second sentence in Figure 15 illustrates a definitional extension that adds to the set of interpretations assigned to an English subset constituent. The notion of "large" is extended to include objects that have a "population" over a certain value and a "quantity of manufacturing establishments" larger than a certain value. Question 3, "What are the large, smoggy Eastern cities?" then compiles into two requests as shown. In this case the answers for the first and second requests are the names of the same city. The remainder of the sentences in this figure illustrate how the original interpretation of the word "large" can be replaced by the second interpretation. The ability to change the number and kinds of interpretations associated with a constituent is an essential function in providing a user with a flexible and user-extendable English subset.

Figure 16 illustrates a brief sketch of the description and interrogation of a fragment of a data base concerning information about secondary school students.

CONCLUSION

We have realized a prototype on-line system for describing, updating, and interrogating data bases of diverse content and structure through the use of ordinary English sentences.

We cannot hope to formalize and implement a complete English grammar in the near future; instead, we face a tradeoff between range of expression and reasonable computer processing times. Therefore, in constructing a grammar for an on-line English subset it is as important to avoid some areas of syntax as it is to recognize others.

By composing derived interpretations from lexical interpretations, under the control of syntax recognition, a process of semantic interpretation has been realized to construct the designations of noun phrases, sentences, and embedded clauses. The intermediate language constitutes a database-oriented "deep" structure. It comprises a framework into which new kinds of objects, predicates and terms may be introduced to represent the semantic structures of a very wide range of possible universes of discourse. In a similar way the data management machine's specific fact structures present a framework for the storage of the extension of a universe of discourse.

Several properties of the CONVERSE natural language compiler are of special importance.

- *Ambiguity resolution.* The semantic interpretation process eliminates interpretations that are not admissible according to selectional feature restrictions and general facts. Syntactic ambiguity is resolved by terminating syntactic structures that do not lead to acceptable semantic structures.
- *User feedback.* Undefined word, structure, meaning, and procedure statements are particularly useful in guiding a user to an awareness of the limits of the English subset and in enabling him to extend the English subset to meet his needs.
- *Semantic reflexivity.* This property allows a user not only to introduce new universes of discourse but, through definitional sentences, to increase the vocabulary and paraphrase facilities of the English subset. It allows him to describe a data base at either a superficial or detailed semantic structure level.
- *On-line responsiveness.* In the present version of CONVERSE a user must typically wait from less than five seconds to as much as one minute for sentence-to-procedure translation. The system is presently being converted for use on the IBM 360 Time-Sharing Systems at SDC. This new program promises to be at least one order of magnitude faster than the present one.

In conclusion, we believe that a natural language compiler should be judged on its merits, not with respect to the linguistic possibilities inherent in man-to-man communication, but with respect to existing means of man-to-machine communication. From this viewpoint, we are convinced that natural language compilers will eventually come into widespread use, for the same reasons that conventional compilers are already being widely used. The gain in convenience and expressive power will more than offset the expense of the required computer processing time.

ACKNOWLEDGMENT

Work on CONVERSE has been carried out within the Natural Language Research Program at SDC with the helpful and enthusiastic support of Robert F. Simmons. Carter C. Revard and Larry E. Travis have supplied valuable comments on a first draft of this paper.

APPENDIX

PRINT MEASURES . PROPERTIES .

M1	NIL
M3	(PRODUCTION FACTORS) (M30 M31 M32)
M4	(DISTRIBUTION FACTORS) (M22 M23)
M5	(POPULATION FACTORS) (M7 M8 M9 M10)
M6	(LOCAL GOVERNMENT FACTORS) (M33 M34)
M2	(STATISTICAL INDICATORS) (M3 M4 M5 M6)
M7	(DISTRIBUTION OF PEOPLE BY AGE) (M11 M12 M13 M14)
M8	(DISTRIBUTION OF PEOPLE BY INCOME) (M15 M16 M17 M18)
M9	(DISTRIBUTION OF PEOPLE BY EDUCATION) (M19 M20 M21)
M10	(POPULATION)
M11	(MEDIAN AGE OF PEOPLE)
M12	(PERCENTAGE OF PEOPLE UNDER 5 YEARS OF AGE)
M13	(PERCENTAGE OF PEOPLE OVER 21 YEARS OF AGE)
M14	(PERCENTAGE OF PEOPLE OVER 65 YEARS OF AGE)
M15	(MEDIAN FAMILY INCOME)
M16	(PERCENTAGE OF FAMILY INCOME UNDER 3000)
M17	(PERCENTAGE OF FAMILY INCOME OVER 10000)

M18 (AGGREGATE FAMILY INCOME IN MILLIONS OF DOLLARS)
 M19 (PERCENTAGE OF PEOPLE OVER 25 WITH LESS*THAN 5 YEARS OF SCHOOLING)
 M20 (PERCENTAGE OF PEOPLE OVER 25 WITH MORE*THAN 12 YEARS OF SCHOOLING)
 M21 (MEDIAN NUMBER OF SCHOOL YEARS COMPLETED)
 M22 (RETAIL TRADE) (M24 M25 M26)
 M23 (WHOLESALE TRADE) (M27 M28 M29)
 M24 (NUMBER OF RETAIL TRADE ESTABLISHMENTS)
 M25 (NUMBER OF RETAIL TRADE EMPLOYEES)
 M26 (RETAIL TRADE YEARLY PAYROLL IN THOUSANDS OF DOLLARS)
 M27 (NUMBER OF WHOLESALE TRADE ESTABLISHMENTS)
 M28 (NUMBER OF WHOLESALE TRADE EMPLOYEES)
 M29 (WHOLESALE TRADE YEARLY PAYROLL IN THOUSANDS OF DOLLARS)
 M30 (NUMBER OF MANUFACTURING ESTABLISHMENTS)
 M31 (NUMBER OF MANUFACTURING EMPLOYEES)
 M32 (MANUFACTURING PAYROLL IN THOUSANDS OF DOLLARS)
 M33 (NUMBER OF LOCAL GOVERNMENT EMPLOYEES)
 M34 (LOCAL GOVERNMENT PAYROLL IN THOUSANDS OF DOLLARS)
 M36 (TEMPERATURE FACTORS) (M39 M40 M41 M42)
 M37 (MEAN ANNUAL PRECIPITATION)
 M38 (MEAN ANNUAL HOURLY WIND VELOCITY)
 M35 (CLIMATE FACTORS) (M36 M37 M38)
 M39 (MEAN JAN TEMPERATURE)
 M40 (MEAN JULY TEMPERATURE)
 M41 (HIGHEST TEMPERATURE)
 M42 (LOWEST TEMPERATURE)
 M44 (FARM LAND IN THOUSANDS OF ACRES)
 M45 (NUMBER OF FARMS)
 M43 (AGRICULTURAL FACTORS) (M44 M45)
 P1 (SMOGGY)
 P2 (WARM)

P3 (COLD)
 P4 (EASTERN)
 P5 (WESTERN)
 P6 (LARGE)
 P7 (SMALL)

REFERENCES

- D G BOBROW J B FRASER M R QUILLIAN
Automated language processing
 In Cuadra C A (ed) Annual review of information science and technology New York Interscience 1967 Vol 2 161-186
- H G BOHNERT P O BACKER
Automatic English-to-logic translation in a simplified model
 IBM Thomas J Watson Research Center Yorktown Heights
 New York March 1966 AFOSR 66-1727 AD-637 227 117 pp
- D BOLINGER
The automatization of meaning
 Language
 (October-December 1965) Vol 41 No 4 555-573
- E BOOK
The LISP version of the meta compiler
 SDC document TM-2710/330/00 November 1965
 Bureau of the census
County and City Data Book 1962
 U S Department of Commerce Washington D C 1962
- R CARNAP
Meaning and necessity
 2d ed University of Chicago Press Chicago 1956
- T E CHEATHAM JR S WARSHALL
 Translation of retrieval requests couched in a semiformal English-like language
Communications of the ACM
 1962 5(1) 34-39
- N CHOMSKY
Aspects of the theory of syntax
 Cambridge The MIT Press 1965
- W D CLIMENSON
File organization and search techniques
 In Cuadra C A (Ed) Annual Review of Information Science and Technology
 New York John Wiley and Sons 1966
- W A COZIER W C DENNIS
 QUUP users' manual SDC document TM-2711/000/01
 February 1966
- V E GIULIANO
 Comments (on Simmons (1965))
Communications of the ACM
 1965 8(1) 69-70
- B F GREEN JR A K WOLF C CHOMSKY
 K LAUGHERY
Baseball an automatic questions answerer
 In *Computers and Thought*
 E A Feigenbaum and J Feldman (Eds) McGraw Hill New York
 1963 207-216
- J J KATZ
The philosophy of language
 New York Harper and Row 1966
- C KELLOGG
An approach to the on-line interrogation of structural files of facts using natural language
 SDC document SP-2431/000/00 29 April 1966
- C KELLOGG
On-line translation of natural language questions into artificial

- language queries*
Information Retrieval (in press) (a)
- C KELLOGG
CONVERSE—A system for the on-line description and retrieval of structured data using natural language
Proceedings of the FID/IFIP Conference on Mechanized Information Storage Retrieval and Dissemination
Rome Italy June 1967 North Holland Pub Co (in press) (b)
- C KELLOGG
Designing artificial languages for information storage and retrieval In Borko H (Ed)
Automated Language Processing
New York John Wiley and Sons 1967
- C KELLOGG
Data management in ordinary English: examples
SDC document TM-3919/000/00 3 May 1968
- E H LENNEBERG
Biological Foundations of Language
John Wiley and Sons Inc New York 1967
- J MINKER J SABLE
File organization and data management
In Cuadra C A (Ed) Annual Review of Information Science and Technology New York John Wiley and Sons 1967
- B RAPHAEL
SIR a computer program for semantic information retrieval
AFIPS Conference Proceedings
Vol 26 Part 1 1964 Fall Joint Computer Conference Baltimore Md Spartan 1964 577-589
- R F SIMMONS
Answering English questions by computer: A survey
Communications of the ACM
1965 8(1) 53-70
- R F SIMMOHS
Automatic language processing
In Cuadra C A (Ed) Annual Review of Information Science and Technology
New York John Wiley and Sons 1966
- L TRAVIS
Analytic information retrieval
In Garvin P L (Ed) Natural Language and the Computer
New York McGraw Hill 1963 310-353

Prices and the allocation of computer time

by NEIL M. SINGER

University of Maryland
College Park, Maryland

and

HERSCHEL KANTER and ARNOLD MOORE

Center for Naval Analyses
Arlington, Virginia

INTRODUCTION

The use of prices as a mechanism for allocating resources is generally well understood. Nearly two hundred years have elapsed since Adam Smith, in *The Wealth of Nations*, discussed the functioning of the "invisible hand" in a market economy, but the principles which he enunciated have not been altered or invalidated by ensuing generations of economists. In the United States today, markets are the dominant economic form, and the price system is used to allocate nearly all the product of our private sector, over 75% of gross national product. (Governments also participate in markets, at least to the extent of obtaining resources.) To be sure, the price mechanism does not always work as well in real markets as in theory—a defect shared by other allocative mechanisms—and certain categories of goods and services continue to be allocated by means other than prices. Among these goods are most of those produced by the several levels of governments and nonprofit organizations such as universities. In addition, non-price allocation techniques are frequently used in instances when market allocation violates social canons of equity, for example during periods of rationing in wartime.

Prices are not the dominant allocative mechanism for computer time, and our purpose in this paper is to examine whether they should generally be used. We will need first to establish the general conditions under which pricing will be efficacious, and then to determine whether the allocation of computer time satisfies those conditions. Finally, we will examine some of the allocative techniques

used instead of prices, to see whether non-price methods can be expected to work as well as prices on this particular problem.

Why use prices at all?

Any economic system must solve the problem of how to use scarce resources. Most resources can be used to produce many goods; most products are useful to many consumers. Due to the scarcity of resources, however, the system is closed: resources used by one producer are not available to any other and goods consumed by one person reduce the total consumption possibilities of all others. Some determination must therefore be made of the preferences of different economic units for the same economic resources or products.

The price system is the vehicle by which economic units express their preferences in a market context. It is axiomatic that the preferences which underlie prices cannot be measured other than through the behavior of buyers. In other words, if a consumer buys a shirt for \$10, it is possible to infer only that he derives at least \$10 worth of satisfaction from that shirt. It is not generally possible to measure that satisfaction in any way other than as \$10 worth.

If preferences are expressed predominantly by prices, then some other desiderata can be obtained within market resource allocation. One such desideratum is that those consumers who place the greatest value on a good are the ones who obtain it (assuming an equitable distribution of income). In a properly functioning market, prices

will be bid up (or down) to the point at which the available supply is allocated to the consumers willing to pay the highest prices.

Implicit in this approach is that the value to society of the goods is the maximum price paid for the goods by any consumer. Prices normally cover costs of production (including profits), but in some instances particular allocations of resources impose costs (or benefits) on third parties other than the producer and consumer. For example, the location of a junkyard may lower surrounding property values, thus imposing costs on persons other than the owner of the junkyard and his customers. The *social* value of the junkyard will be less than its *private* value (to the owner) unless the owner is required to charge prices high enough to pay compensation to surrounding property owners. Prices high enough to compensate for external costs are efficient (in the technical sense that no other allocation of resources will produce output with a greater net value). In general, external costs can be included in a price-system allocation when the third parties can enter into the market determination of price. Typically, the goods whose production imposes unrecognized or uncompensated external costs, will be underpriced and oversupplied compared to the amount of production which is socially optimal.

Not only does a price system establish a priority of users, it also establishes a priority of wants. The producer willing to pay the highest price for a resource will be the one whose product is valued most highly by the members of society (adjusting for differences in the productivity of resources). Those wants to which the members of society give priority will be able to obtain the resources necessary for their satisfaction—and they will be satisfied only to the extent that no other want receives greater priority.

Finally, a price system can solve the problem of allocating resources dynamically. It does so by transmitting information about consumers' demands to producers, and offering consumers information about the cost of satisfying different wants. If consumers express great demand for some goods relative to its current supply, the price will be bid up above the cost of production. The ensuing profitability of production will offer producers an incentive to increase production, bidding resources away from less profitable allocations. Over a period of time, labor and capital will be drawn to produce a particular goods only if de-

mand for the goods is high enough to cover the cost of the additional resources.

In summary, a price system can solve the problems of distributing goods and services among consumers, allocating resources among producers, and conveying information to determine the flow of resources among different allocations over time. If prices are to solve these problems, they must be able to respond in certain ways. When demand rises or falls, prices must rise or fall sufficiently to ration the available supply. Over time, changes in supply must affect the price of any goods; that is, constancy of price over time is neither necessary nor desirable for efficient resource allocation. And since the role of price is to allocate resources, the price at any point in time need not bear any relation to the cost of production at that time. This point should be stressed: prices are a rationing device, not a mechanism for recovering cost. If demand for a good is low, its price may well fall below cost, transmitting information to the producer that demand is inadequate. Unless price is permitted to fall below cost, the proper information about demand may never be obtained, and the allocation of resources can never adjust properly to the unprofitability of that good.

Can computer time be priced?

Two aspects of the question of pricing computer time must be discussed. The first is whether prices can allocate computer time of a fixed quantity and productivity (as determined by machine configurations at any time). The second is the extent to which pricing can allocate computer time over periods in which demand patterns and machine productivity both change.

As noted above, the conditions for pricing to work are fairly simple: users must be unable to obtain any scarce resource at a zero price; social and private direct benefits must be identical, so that neither benefits nor costs are incurred except by the buyer and seller; and prices must be free to vary without regard to cost of production. (These are rigorous requirements; in fact, pricing will usually work if these requirements are satisfied loosely.) Applied to computers, the first condition requires that computer time be made available to any user willing (and able) to pay the price. The price may be stated in dollars, as at computer service bureaus, or it may be stated in terms of some fiat money. It is essential, however, that the user feels that he is truly paying a price.

If the price is stated in dollars, then the user's budget must be limited. If the price is simply an accounting of computer time, then the user's overall access to the computer must be limited. Only if there is a budget constraint will the user have an incentive to evaluate the benefits of computer time relative to its cost to him and other users, for otherwise the cost will be zero.

The second condition requires that all costs and benefits be incurred within the market. We have noted elsewhere¹ that a user of computer time imposes costs on all other users by increasing their turnaround times. The same point has been expressed by other authors, including Marchand² and Greenberger³ (implicitly). The external costs imposed by one user on all others are no less important a category of cost than the direct use of the computer, and efficient allocation will result only if each user pays a price high enough (in principle) to compensate all other users for the reduction in quality of their service. This situation is by no means unique to the allocation of computer time. Every rush-hour commuter imposes external costs on all others in the form of longer commuting times, and every attendee at a Broadway opening night imposes costs on those unable to obtain tickets in the form of reductions in their satisfactions. One difference between these two examples is that social policy has dictated a zero price for highway commuters, thus prohibiting the use of prices to eliminate congestion; whereas show ticket prices can fluctuate widely, and in fact a premium is charged for many opening nights. The problem of planning to eliminate or reduce congestion can be attacked through queuing theory (as by Kleinrock² and Greenberger³ among many others), but it appears perfectly feasible to eliminate congestion *to the extent desired by users* by letting them bid up prices. Equilibrium will be reached where each user is paying a price which is greater than the total value of the decrements in service to all other affected users. Note that these external costs will arise only for a heavily-used computer on which each job seriously affects all other turnaround times. For a batch processor, such "capacity" use could occur for any turnaround time, or set of turnaround times. For a time-sharing system, the related notion of access time (to a console or to the central processing unit) could be used.

The third requirement for pricing to be effective is that prices be free to fluctuate without re-

gard to the cost of production. In general, a good should be supplied to any consumer willing to pay the cost of the additional resources needed to produce it. In the case of computer time, the additional resources necessary for production are virtually free (at least within eight-hour blocks) and in any case are inexpensive relative to the amortization cost of the computer. Nonetheless, price is not a cost-recovery mechanism, and it is proper *at any point in time* to disregard completely the capital cost of a computer in setting price. If a computer is utilized at less than capacity, the external costs imposed by an additional user will normally be low, and the price of computer time should correspondingly be low so as not to discourage use of an essentially free facility. The extent to which amortized cost exceeds revenue is an operating loss, and should be treated in the same way as, for example, a loss experienced by a realtor on an apartment house with some unrented units.

We have been using the term "capacity" to mean the ability to process some maximum number of jobs (per time period) of *given quality of service*, or turnaround time. (There is, of course, no reason why the turnaround time used to estimate machine capacity should be that desired by users, or why all users should prefer the same turnaround times.) We conclude from our discussion that pricing is a feasible allocative device for a computer with fixed capacity in the short run, when demand is stable. The proper price will cover the marginal cost of operating the computer to satisfy the demand of the additional user. If the computer is so heavily utilized that each user imposes external costs on others by lengthening their turnaround times, the price must be high enough to cover these external costs, as well as other costs of using the computer.

In the long run, when demand and supply conditions are free to vary, the pricing problem becomes more complex. We have described elsewhere¹ what we envision as the typical time pattern of efficient prices for computer use—a pattern in which prices initially are low, to encourage use of a large, fast, new computer, and eventually rise reflecting use at or near the capacity of the computer. Throughout this pricing cycle, minor investments may be made, adding to the capacity of the computer, and eventually a new main frame is procured. At each point in time, prices are set in accordance with the principles stated above. If the computer is an efficient investment, over-re-

covery of cost during periods of capacity use will just offset early losses. Of course, there is no requirement that cost recovery be exactly equal to operating loss, any more than any investment is required to yield benefits exactly equal to costs. Typically, some computers will prove to be efficient investments and others will not.

In this context, it is worth noting that we have previously expressed our view that idle time is not an indication of inefficient use of a computer, since capacity cannot be defined without reference to the quality of service. If users place a high value on short turnaround times, idle time will be valuable to users as a guarantee of high-quality (i.e., fast) service. In this case, users should be willing to pay a price for idle time at least equal to the value of that time to the excluded users. Our conclusion followed from explicit consideration of the quality of service as a variable to be set by the computer center. The same conclusion is reached from queuing-theoretic considerations by Araoz and Malmgren:⁵

“Some idle capacity can now be seen as a device of efficient production in many cases, and the ‘excess capacity’ which is often observed . . . may sometimes, if not often, arise out of a seeking of efficiency rather than restriction. This will be true insofar as there are any rigidities (or what an economist might call indivisibilities with respect to time) . . . which might give rise to a queuing process.” (p. 209)

Do alternative techniques work?

We will discuss three types of non-price allocation mechanisms: average-costing, overhead charges, and various priority plans. Average-costing is used widely as an internal cost-recovery device, and is imposed externally in many cases by federal government accounting regulations. Its basis is that a facility should recover its costs in any time period, but while that basis is not unreasonable over the lifetime of a computer (or any other capital goods), serious misallocations result when the time period is chosen as a year or even less. We have discussed elsewhere¹ the various abuses that may arise under average-costing, so they will only be listed here: (1) computer centers are unable to provide service to additional users at marginal (social) cost; (2) the resulting pattern of charges encourages use during peak

periods and discourages use when the computer is idle; (3) over-investment is likely to result if contractors are guaranteed recovery of average cost; and (4) under federal government auditing regulations, funding of computers by foundations and other non-users is discouraged. Average-cost charging is a popular internal accounting device due to its ease of administration, but the incentives it offers are unlikely to promote efficient resource allocation.

When a facility is used widely and the cost of its services is difficult to impute to individual users, the facility is frequently called an “overhead expense” and its cost is then allocated to users on an arbitrary basis. When computers are treated as overhead, the full costs (including amortization) of the computer center are included in the firm’s general overhead pool which is imputed to individual projects on a basis such as total labor costs, total man-hours, or total operating costs of each project. Some universities have used this method for recovering the costs of their computer centers.

It should be obvious, however, that overhead charges can offer the proper incentives to neither the user of the computer nor the administration concerned with supplying computer time. Each user will prefer to substitute computer time for other resources, thus reducing his basis for overhead charges, and the overall effect must be to increase the demand for computer time. (In addition, overhead charges will discriminate against projects which are not computer-intensive, thus creating an inequitable set of charges.) If each user substitutes the same ratio of computer time for direct charges, the result will be to leave the pattern of charges unchanged, but to bias upward total use of the computer. The supplier of computer time will then be misled into overestimating the demand for the computer, resulting in overinvestment in subsequent computer facilities. Finally, the overhead rates that the firm must charge (to sponsors, if it is a contractor, or to purchasers, if it is a producer) will be inflated by the misallocation of resources and overinvestment, leading eventually to declining revenues and to reduced profits in the case of a firm.

Priority mechanisms have received wide attention in the literature on managerial and operations-research problems. In contrast, they have been virtually ignored by economists. The reason for the disinterest of economists appears to be that

priorities are simply a surrogate set of prices that may in some instances work as well as a true price mechanism but will almost never be superior. For their part, operations analysts seem unaware that priorities are a form of prices; thus, Kleinrock⁴ discusses "bribes" which are merely prices, and Greenberger⁵ tries to minimize the cost of delay, a cost which can never be known except in terms of the price users would pay to avoid the delay.

There are two distinct types of priority rules: one which determines the access pattern of a given set of users, and another which offers incentives to potential users in determining their demands for computer time. Some of the variants of the first type of rule are listed by Greenberger: first-come-first-served, the *c/t* rule (the next job served is that with highest waiting cost per service time), shortest-job-next, and round-robin scheduling. The difficulty with rules of this sort is that an implicit assumption must be made about the value placed on computer time *by each user*. For example, application of the *c/t* rule involves "judicious approximation"—in other words, arbitrary judgments about users' costs—unless all costs are equal and either constant or exponentially-discounted functions of time. In general, users will not value time equally, nor consider waiting equally costly, so such a rule will not allocate time properly. Moreover, the only way for the computer center to discover if all users value waiting and access equally is to ask them—that is, to establish a market in which preferences may be expressed. Priority rules of this type, therefore, are equivalent to prices under the special assumption that all users place the same values on computer time and experience the same costs for waiting. These conditions are, of course, most unlikely to be satisfied in practice, but under any other conditions priority rules will result in a misallocation relative to the one obtainable by pricing.

The second type of priority is the one which rations access to the computer according to the relative importance attached to the user's project. Such priorities are a form of artificial money, for a high priority in the absence of dollar prices has the same effect as a large project budget if prices are set in dollars. There are, however, two problems. First is the question of flexibility—are priorities reset regularly, as prices must be, reflecting changing conditions of user importance? If not, projects are likely to have either "too easy" or "too hard" a time obtaining access to the com-

puter. Are priorities set on a sliding scale, reflecting the varying importance of successive quanta of computer time? The answer is usually that they are not variable, but that all users in a project receive the priority attached to that project.

Second, a priority system for computer access discourages efficient substitution of other resources for computer use. A project leader whose access to the computer depends on his project's priority will use computer time extensively if the priority is high, even if the computer is not the proper research tool. The computer administrator may react to pressures from projects by assigning more high priorities than the computer can accommodate. This situation is akin to a government's attempt to obtain resources by printing money: the result is to inflate all prices, or in this instance to downgrade the value of all priorities. There will be no incentive for any project to estimate the value of computer time, since the costs are not expressed in any systematic way and since trades of computer time for other resources generally cannot be made.

The one remaining advantage of priorities over prices is that they are inexpensive to administer. An efficient price system must provide periodic—if not continuous—variations in price in response to fluctuations in supply and demand. It might be expensive to "make a market," that is, to provide price information to suppliers and demanders. This is largely an empirical question, and well beyond the scope of this paper. One bit of evidence, however, indicates that the incremental cost of establishing queues of variable length with prices that fluctuate fairly often, given the existence of equipment for accounting for users' time, is of the order of no more than one per cent of the total cost of a batch processor. Even if the cost of pricing a time-sharing system were twice as great, it seems likely that the net gain in efficiency would be substantial. In any case, a price system which adjusted no more frequently than a priority system would be no more expensive.

SUMMARY

We have made several contentions which may appear radical in view of current costing procedures for computer systems. At the heart of our recommendations is the assertion that prices are not a mechanism for recovering cost, but are instead a device for allocating scarce resources and obtain-

ing the efficient level of investment over time. We believe that it is feasible to use prices to allocate computer time, whether the users are a firm's internal projects or its external customers. Prices are only now beginning to be used for batch processing, and it is too early to evaluate the difficulties in pricing time-sharing computers. But the conditions for pricing will be present in time-sharing systems—identification of output, costs to excluded users (either at the central processor or at the console), and scarcity of total resources—and so we expect prices to be a feasible allocative mechanism.

The role of prices is enhanced by the unsatisfactory nature of the alternatives. Average-cost pricing creates a perverse set of incentives for user and supplier. Overhead charges are even worse, for by themselves they establish no mechanism for allocating computer time, and they are likely to be inequitable. Priority rules are the least un-

satisfactory alternative, but their desirable properties are the ones they share with prices. Priority systems are unlikely to obtain an allocation of computer time preferable to that of prices, or to cost less to administer.

REFERENCES

- 1 H KANTER A MOORE N SINGER
The allocation of computer time at university computer centers
Journal of Business 41 375 1968
- 2 M MARCHAND
Priority pricing with respect to time-shared computers
Working Paper No 247 Center for Research in Management
Science Univ of California Berkeley
- 3 M GREENBERGER
The priority problem and computer time sharing
Management Science 12 888 1966
- 4 L KLEINROCK
Optimum bribing for queue position
Operations Research 15 304 1967
- 5 A B ARAOZ H B MALMGREN
Congestion and idle capacity in our economy
Review of Economic Studies 28 202 1961

The use of hard and soft money budgets, and prices to limit demand for centralized computer facility

by SEYMOUR SMIDT

*Cornell University
Ithaca, New York*

INTRODUCTION

A fundamental problem in any large organization is how to decentralize decision-making, and still insure that the decision-makers will act in a manner that is consistent with the goal of the larger organization. The advantages of decentralization are well-known, and will not be discussed. Two possible disadvantages of decentralization are relevant here.

First, the decentralized decision-makers may have goals that conflict with the goals of the larger organization. Second the actions of a decentralized decision-maker may affect other parts of the organization; and the decision-maker may either be unaware of these effects, or be unable to estimate their significance. This paper deals with situations in which the goals of the individual decision-makers are consistent with the goals of the larger organization, so that goal conflicts are not a problem. The paper concentrate on ways of overcoming the second type of disadvantage of decentralization.

A centralized computer facility is a good example of a situation in which the second disadvantage mentioned above may be a problem. Computers are subject to economies of scale, so it is often logical for an organization to encourage its decentralized decision-making units to share the use of a common computer facility. In the rest of the paper the word user will mean a decentralized decision-making unit that is, at least potentially, a user of this shared computer facility.

Since the use of the computer by one user imposes costs on the other users and/or on the central organization, some means of limiting or controlling demand for computers is essential.

One means of limiting demand is to charge users for the use of the computer. If the user is a profit-center, and the computer charge is part of his costs, the user has an incentive to use the computer only in ways that will increase the profits of his unit. In this case the

charge is a self-policing device and there is no reason to limit such user's computer demand by an arbitrary budget constraint.

However, many users have goals whose achievement cannot easily be measured in monetary terms. This is the usual situation in universities and governmental organizations, and it also obtains in many research or staff activities in business organizations. In this case a charge for using computers will be effective only if the user is subject to some effective budget constraint.

This paper deals with the problem of effectively limiting demand for computers in a decentralized organizations in which the computer users are attempting to maximize some goal that cannot be expressed in monetary terms. The users' goals are assumed to be consistent with the goals of the organization. The users seek to maximize their goals subject to some budget constraints, in circumstances in which they are charged for their use of the computer. The user's budget is provided by the central organization (which is itself subject to a budget constraint) or by an outside organization. In any case, the user treats the amount of his budget as an externally determined factor that is outside of his control. The size of a user's budget reflects a judgment by the central organization or an outside agency, about the relative importance of the goals the user is attempting to achieve.

This paper is concerned with two related issues that

*The single price for computer services represents the average price paid by users and credited to the central computer facility during the relevant planning horizon. This average price may be the result of a rather complex pricing system in which the amount charged for a particular job depends on the characteristics of the job, the priority assigned to the job, the load on the system at the time the job is processed, and other relevant considerations. A number of authors have considered the characteristics that should be incorporated in a pricing system for computer services. See bibliography.

arise in this context. The first issue is what basis should be used in establishing prices (or charges) for computer usage. The second is whether or not users should be given a separate budget that is applicable only to their computer usage.

The conclusions reached in this paper are based on a series of mathematical models, that are described in an appendix. Readers interested in the details of the logic should consult this appendix. Only the main assumptions made, and the conclusions that result are summarized in the paper.

Decision variables available to the organization

The central organization has several sets of decision variables it can manipulate to achieve its objectives. However, these decision variables are interrelated. This section will attempt to describe these interrelations. One set of decision variables relates to the amount and kind of computing facilities that will be provided, including both hardware and software. These decision variables can be thought of as determining a supply schedule of computer services. A second set of variables relates to user budgets. The organization can determine how large a budget it will allocate to each user. It can also specify, if it wishes, that a certain amount of the user's budget can be spent on computers, but not on other goods and services. Given the characteristics of users, these decisions can be thought of as determining the demand schedule for computer services. A third set of variables that the organization may specify relates to the terms on which computer services are made available to users. In this paper a single price variable will be assumed to represent these terms.*

Because the three sets of variables are interrelated, the organization cannot arbitrarily pick levels for all three variables independently. Rather, if levels are set for any two of the three, a level for the third variable is implied. For example, suppose that the demand variables and the supply variables have been determined. Then the price variable must be left free to adjust demand and supply. If the central organization attempts to fix the price variable as well, then some other aspect of the terms on which computer services are available will perform the adjustment. For example, if the price for computer services is not free to rise when demand exceeds supply, then turnaround times will increase, which in turn will impose costs and inconveniences on users. The effects are similar, if not quite identical, to those that would result from an increase in the price a user must pay to achieve a given rate of turnaround. Alternatively if the organization has determined the demand schedule by fixing user budgets, and wishes to maintain a certain price level without compensating changes in other non-price terms, it must be prepared

to supply the amount and kind of computing facilities that will be demanded. Finally, the organization may decide on the amount and kind of computer facility it is willing to provide, and the terms on which computer services will be made available, and then try to adjust the demand variables in a manner that is consistent with the levels of the other two variables.

In practice, an organization cannot accurately predict what price level for computer services will result from a given set of decisions about the supply and demand variables. If the price level that results is not what was desired, some adjustments will be necessary. Either the demand or supply variables, or both, should be adjusted to move toward the desired equilibrium. Whether the initial response to a disequilibrium price takes place by adjusting the demand variables or the supply variables depends on how long it takes to adjust one or the other of these variables. If user budgets take a long time to change, but changes in hardware and software can be made relatively quickly, then the appropriate initial response to a disequilibrium is to adjust the supply variables, "or" taking into account the fact that the demand variables will not be changed for some time. In other circumstances it may be that hardware and software changes take a long time but budget changes can be made relatively quickly.

The conclusion from these comments is that for long-range planning one should consider all three sets of variables as subject to control by the decision-maker, while for short-run adjustments it may be necessary to treat either the demand variables or the supply variables as being outside of the control of the decision-maker. The first three models considered in this paper take a long-range viewpoint. The fourth model assumes that the supply conditions have been fixed, but that user budgets are subject to control.

General assumptions

The class of models considered in this paper have the following characteristics. Each user receives a budget allocation from the central organization.* The user tries to purchase amounts of computer services and of non-computer inputs that maximize his utility subject to his budget constraint. The users treat the prices of all inputs as given.

The central organization must pay for the costs of the computing facility, and for any non-computer inputs purchased by users. It has available, to meet these payments, a predetermined quantity of its own resources plus any amounts that users have added to their budget

*One model allows for the possibility that a user may receive budget allocations from other sources as well as from the university.

allocations from outside sources. The only resource allocation decisions made directly by the administration are those concerning the type and capacity of the computing facility. The organization allocates budgets to users who make the detailed decisions about the use of the computing facility, and the amounts of non-computer inputs purchased. The organization indirectly controls the behavior of users by setting the pricing schedule for computer services, by determining the budget allocation assigned to each user, and possibly by requiring that a certain amount of the user's budget allocation can be spent only on computers. It is assumed that the pricing system used does not discriminate among users.

The paper assumes that there is no conflict between the goals of users and the goals of the central organization. Specifically, this means that if a user can increase his utility without reducing the utility level of any other user, then the level of utility achieved by the organization will also increase.

In describing the various models, the terms hard and soft will be applied to monetary amounts that constrain a decision-maker. Specifically, hard money is money that can be spent for any purpose, while soft money is money that can be used only in some limited way. The adjectives hard and soft are also applied to budgetary allocations.

These adjectives are often used in informal discussions of computer budgeting problems. Their formal use is justified by analogy to their usage in international trade theory. A hard currency is acceptable as a medium of exchange in one country, and is easily convertible into the currencies of other countries. Thus hard currency is effectively usable anywhere. A soft currency is acceptable as a medium of exchange in one country, but is not (easily) convertible into the currencies of other countries. Thus soft currency can be spent only on a limited range of goods.

Efficient pricing with and without external financing of users

The first model to be considered assumes that all users receive their budget allocations entirely from the central organization, and that the organization does not impose any restrictions on how the user can allocate the budget assigned to him. Furthermore this model assumes that the organization sets a price level for computer services, and then purchases (or rents) whatever amount and kind of computing facility is necessary to satisfy the user's demands at the given price.

Under these conditions the central administration has two independent decisions to make. It can decide how large a budget to allocate to each user, and it can decide

the price it will charge for computer services. To maximize its utility the administration should assign budgets to each user so that at the margin the increase in satisfaction the organization derives from each additional dollar allocated to a particular user's budget is the same for all users. In practice, it is assumed that the ordinary budget-setting procedures approximate this formal requirement. Second, the administration should price computer services at their marginal cost to the organization.* This second condition is correct provided that the organization would not be better off with no computer facility.** The results from this set of assumptions are hardly surprising; but they provide a benchmark for comparing the results from other sets of assumptions.

The next model to be considered continues the assumptions that only hard money budgets are used, and that computer capacity is adjusted to demand. However, this model allows for the possibility that some users receive at least part of their budget allocations from sources outside the central organization.†

Budget allocations that a user receives from outside sources are assumed to be hard money from the point of view of the user, but soft money from the point of view of the organization. Specifically the organization is assumed free to reduce the budget it allocates to a user if the user receives funds from outside. However, the organization cannot take funds a user receives from outside, and re-allocate them to other users. Thus the total budget allocation of a user (from all sources) must be at least as large as the allocation he receives from outside.

The fact that a particular user receives some budget allocations from outside sources (and that the organization receives a corresponding amount of funds) may or may not change any of the conclusions of the previous model. If, when a particular user receives some funds from outside, the organization reduces the amount it allocates to that user by a corresponding amount, and reallocates those funds among all users, then the effect is exactly the same if the funds had been given directly to the organization for its unrestricted

*If some flexible pricing system is used, this means that the average charges earned per day by the computer facility should equal the incremental costs of expanding capacity (including operating costs).

**This need not mean that users have no computer services available to them. It might mean that higher priced computer services are purchased from outside sources.

†Examples would be a university in which some professors receive research support from non-university sources, or a local government in which some programs are supported in part by grants from the federal government.

use. Under these conditions the formal results of the previous model are equally applicable here.

However, if some users who are financed from outside sources receive larger budget allocations than they would have received from the organization, then it no longer follows that computer services should be priced at marginal cost. In general, under these circumstances, the optimal price for computer services will be somewhere between marginal cost and the price at which marginal cost equals marginal revenue. The extreme case in which the optimal price is set so that marginal cost equals marginal revenue would occur only if every user who actually used computer services was financed from outside, and if this outside financing was so generous that the organization would gain no additional utility from an additional dollar allocated to the budget of such a user.

Significance of marginal cost pricing

The importance of the possibility that the optimum price may exceed marginal cost will be clearer if the cost structure of the computing center is considered. It is commonly believed that computers are subject to economies of scale. That is, a one per cent increase in the amount spent on owning and operating a computer leads to more than a one per cent increase in the quantity of computing that is possible. Under these cost conditions, if prices are set equal to marginal cost, the total revenue of the computing center will be less than its total cost. That is, the computing center will operate at a deficit.

By contrast, setting prices so that marginal costs equal marginal revenue is the rule to follow if one wants to maximize the profits (or minimize the deficits) of the computing center.

To the extent that some users receive larger budget allocations from outside than they would have received from internal sources, raising prices above marginal cost becomes advisable. (The higher prices would apply to all users.) By paying more than marginal cost, outside financed users tend to reduce the deficit of the computing center and thus increase the amount the organization has available for internally financed users. In effect, by setting prices above marginal cost the organization uses the computing center as an indirect means of re-allocating funds from externally financed users to internally financed users.

The use of hard and soft money budgets

In the next model, the assumption that all users are financed from internal sources is reinstated, and the assumption that capacity is adjusted to effective demand is retained. However, in this model, the organ-

ization is permitted to make two types of budget allocations to users.

The hard money budget allocation can be used for either computer services or for other inputs. The soft money budget can be used only for computer service charges. Under these conditions, if the optimal amounts of hard and soft money are allocated to each user, the total budget allocation each user receives will be the same as he would have received if only hard money had been allocated. Furthermore, the amount of computer services each user purchases will be the same as in the hard budget model. Also the optimal price for computer services is still at marginal cost.

In summary, if all users are financed from internal sources, and computer capacity is adjusted to demand, there is no advantage to be gained from distinguishing between hard money and soft money in making budget allocations to users. In practice, distinguishing between hard money allocations and soft money allocations under these conditions is likely to lead to a less efficient use of resources, since users have less possibility of adjusting their expenditure patterns as circumstances change within the budget period.

The fourth and final model considered is one in which users receive no outside financing, both hard and soft money budgets are allowed, and the capacity of the computing facility is assumed to be fixed. Since computing capacity is fixed, it is assumed that the price charged for computer services is free to adjust so as to equate the quantity demanded to the fixed supply.* The only decisions that remain to be made by the organization are how much hard money and how much soft money to allocate to each user. Under these conditions, it may be desirable to use both hard and soft money budget allocations.

The necessary condition for an optimum under these assumptions is that the marginal utility the organization derives from an additional dollar of hard or soft money allocated to a particular user's budget be proportional to the extra cash outlays that will result for the organization. If computer capacity were variable, and were priced at marginal cost, each dollar of hard or soft money allocated to a user would lead to an expenditure of one dollar by the organization. But when computer capacity is fixed an additional dollar spent by a user on computer services will cause less than one dollar of additional expenditure by the organization. The immediate effect of the user's expenditure is simply to

*Although strictly speaking the mathematical model assumes that prices are flexible, in practice if prices were also fixed, demand and supply would be equated by variations in the non-price costs of using the computing facility. For example, if the demand function increased, and prices could not rise, turnaround times would increase, queues would form at remote terminals, etc.

drive up the price of computer services. Additional cash outlays by the organization occur only to the extent that the higher price causes some other users who have been spending hard money budget allocation on computer services to spend less on computer services, and more on other inputs.**

Suppose a fixed computer capacity is described as excess (deficit), to the extent that it exceeds (is less than) the organization would have chosen if capacity were variable. In practice, soft-money budgets are likely to be useful to an organization only during periods when there is excess computer capacity. Under these conditions allocating soft money to users encourages them to make use of the excess capacity, at a lower dollar cost to the organization than if they had been allocated hard money. By contrast if there is a deficit of computer capacity reducing a user's soft-money budget past a certain point has the same effect on his computer usage as reducing his hard money budget. There soft money is not effective in limiting demand when there is a shortage of computer capacity. †

When soft money budgets are appropriate, their amounts should be determined by comparing the benefits the user receives from his additional use of the computer, to the real dollar cost that his usage causes the organization. As explained above, when computer capacity is fixed, the real dollar cost to the organization will be greater than zero, but less than the soft-dollar expenditure by the user.

APPENDIX

Long range planning with only internal financing: Model I

Resource allocation decision of users—In the planning horizon under consideration, the organization has available to it a fixed dollar amount, K, available for expenditure. Detailed decisions about resource allocation are decentralized by assigning budgets to users. A user is any decision-making unit in the organization that is free to make its own resource allocation decisions subject to the budget constraints determined by the central organization. Assume all users receive funds only from the central organization, and that one

user cannot transfer funds to another. Two commodities (really categories of commodities) are available to users. Commodity one is purchased from outside the organization at a market price that cannot be changed by the behavior of the organization, or by users individually or collectively. Commodity two, representing computer services, is produced by the organization, and decisions about the quantity available, and the price at which it will be sold are made by the organization. However, given the allocation of users budgets, there is a one-to-one relation between the supply of computer services and the price at which it is sold to users. This is because the organization's policy is to allow prices for computer services to fluctuate to equate supply and demand. For users, that price is taken as given. For the organization it is convenient to think of price as a policy variable and computer capacity as determined by the amount demanded at that price. Users try to maximize their utility. The following variables are needed to express the behavior of a user in mathematical terms.

- P_1 = price of commodity one
- P_2 = price of commodity two
- q_{i1} = amount of commodity one used by user i
- q_{i2} = amount of commodity two used by user i
- I_i = budget assigned to user i by the organization
- u_i = an index of the satisfaction derived by user i where

$$u_i = U_i(q_{i1}, q_{i2}) \tag{1}$$

Equation (1) implies that user satisfaction is determined by the amounts of each commodity he consumes. With respect to the utility function U_i , the following assumptions are made

$$\frac{\partial U_i}{\partial q_{ij}} > 0 \quad \text{all } i, j = 1, 2. \tag{2}$$

$$\frac{\partial^2 U_i}{\partial q_{ij}^2} < 0 \quad \text{all } i, j = 1, 2. \tag{3}$$

The i^{th} user faces the following problem:

Maximize $u_i = U_i(q_{i1}, q_{i2})$
 subject to $P_1 q_{i1} + P_2 q_{i2} = I_i$ (4)

and $q_{ij} \geq 0 \quad j = 1, 2.$ (5)

Using a Lagrange multiplier technique the necessary

**This statement assumes an elastic demand for computer services by users who are, at the margin spending hard money allocations on computers. If the demand, schedule of these users is inelastic, allocating soft money to other users will reduce the cash outlays of the organization in the short-run.

†If soft money budgets have been used to encourage demand because of excess capacity, and the amount of excess capacity is reduced, then reducing the amounts of soft money allocated may be useful.

conditions for a maximum for the user can be determined by differentiating (6) partially with respect to q_{i1} , q_{i2} , and λ_i .

$$\begin{aligned} \phi_i(q_{i1}, q_{i2}, \lambda_i) = & U_i(q_{i1}, q_{i2}) \\ & + \lambda_i(I_i - P_1 q_{i1} - P_2 q_{i2}) \end{aligned} \quad (6)$$

The partial derivatives are:

$$\frac{\partial \phi_i}{\partial q_{i1}} = \frac{\partial U_i}{\partial q_{i1}} - \lambda_i P_1 = 0. \quad (7)$$

$$\frac{\partial \phi_i}{\partial q_{i2}} = \frac{\partial U_i}{\partial q_{i2}} - \lambda_i P_2 = 0 \quad (8)$$

$$\frac{\partial \phi_i}{\partial \lambda_i} = I_i - P_1 q_{i1} - P_2 q_{i2} = 0 \quad (9)$$

Let q^{*ij} be the values that satisfy this maximization problem. In general, the optimum values will depend on the values of P_1 , P_2 and I_i . Thus

$$q^{*ij} = d_{ij}(P_1, P_2, I_i) \quad \text{all } i, j = 1, 2 \quad (10)$$

Equation (10) is the demand schedule of the i^{th} user for commodity j . From equation (4), q^{*i1} can be expressed as

$$q^{*i1} = \frac{I_i - P_2 q^{*i2}}{P_1} \quad (11)$$

Let Q_1 and Q_2 be the total demand by all users for these two commodities. Then, if there are n users,

$$\begin{aligned} Q_j = \sum_{i=1}^n q^{*ij} &= \sum_{i=1}^n d_{ij}(P_1, P_2, I_i) \\ &= d_j(P_1, P_2, I_1, I_2, \dots, I_n) \end{aligned} \quad (12)$$

The optimum allocation at the central organization level depends on how each user's utility changes as a function of his budget-allocation and the prices of the inputs he uses. Consider the budget allocations first. Using equation (11), the utility of the i^{th} user, if he is at an optimum position, can be expressed as:

$$u_i^* = U_i\left(\frac{I_i - P_2 q^{*i2}}{P_1}, q^{*i2}\right) \quad (13)$$

$$\begin{aligned} \frac{dU_i^*}{dI_i} &= \frac{\partial U_i}{\partial q^{*i1}} \cdot \frac{\partial q^{*i1}}{\partial I_i} + \frac{\partial U_i}{\partial q^{*i2}} \cdot \frac{\partial q^{*i2}}{\partial I_i} \\ &= \frac{\partial U_i}{\partial q^{*i1}} \left(\frac{1}{P_1} - \frac{P_2}{P_1} \frac{\partial q^{*i2}}{\partial I_i} \right) + \frac{\partial U_i}{\partial q^{*i2}} \cdot \frac{\partial q^{*i2}}{\partial I_i} \end{aligned} \quad (14)$$

The simultaneous solution of equations (7) and (8), which are necessary for an optimum, require that $\frac{\partial U_i}{\partial q^{*i1}} / \frac{\partial U_i}{\partial q^{*i2}} = \frac{P_1}{P_2}$, and therefore that

$$\frac{\partial U_i}{\partial q^{*i1}} = \left(\frac{P_1}{P_2} \right) \frac{\partial U_i}{\partial q^{*i2}} \quad (15)$$

Substituting this in (14) and rearranging, gives

$$\begin{aligned} \frac{du_i^*}{dI_i} &= \left(\frac{P_1}{P_2} \right) \left(\frac{\partial U_i}{\partial q^{*i2}} \right) \left[\frac{1}{P_1} - \frac{P_2}{P_1} \cdot \frac{\partial q^{*i2}}{\partial I_i} \right] \\ &+ \frac{\partial U_i}{\partial q^{*i2}} \cdot \frac{\partial q^{*i2}}{\partial I_i} = \left(\frac{1}{P_2} \right) \frac{\partial U_i}{\partial q^{*i2}} \\ &- \frac{\partial U_i}{\partial q^{*i2}} \cdot \frac{\partial q^{*i2}}{\partial I_i} + \frac{\partial U_i}{\partial q^{*i2}} \cdot \frac{\partial q^{*i2}}{\partial I_i} \\ &= \frac{1}{P_2} \frac{\partial U_i}{\partial q^{*i2}} \end{aligned} \quad (16)$$

Noting that P_2 is always positive (since it is a price), it follows from condition (2) and (3) that the utility of the i^{th} user will increase at a decreasing rate as his income increases.

Now consider how the i^{th} user's utility changes as a function of a change in P_2 . Differentiating equation (13) partially with respect to P_2 gives

$$\frac{\partial U_i}{\partial P_2} = \frac{\partial U_i}{\partial q^{*i1}} \cdot \frac{\partial q^{*i1}}{\partial P_2} + \frac{\partial U_i}{\partial q^{*i2}} \cdot \frac{\partial q^{*i2}}{\partial P_2} \quad (17)$$

From equation (11),

$$\frac{\partial q^{*i1}}{\partial P_2} = \left(-\frac{1}{P_1} \right) \left(P_2 \frac{\partial q^{*i2}}{\partial P_2} + q^{*i2} \right) \quad (18)$$

Substituting equations (15) and (18) into (17), and rearranging, gives

$$\frac{\partial U_i}{\partial P_2} = -q^{*i2} (1/P_2) \frac{\partial U_i}{\partial q^{*i2}} \quad (19)$$

It follows from conditions (2) and (3) that the utility of the i^{th} user will decrease as P_2 increases.

Central organization decisions—The central organization's decisions are how much money to allocate to each user, and what price to set for computer services. Assume the organization's objective is to maximize a linear combination of user utilities. (More complicated functions of user's utility might also be consistent with the assumption of no goal conflict between users and the organization; but these complications will not be considered in this paper.) The organization's objective function can be written as

$$U_0 = \sum_{i=1}^n \mu_i u_i^* \quad (20)$$

$$\mu_i \geq 0 \quad i = 1, 2, \dots, n$$

The organization's budget constraint is that the amount it can allocate to users is the sum of its fixed resources, K , plus the profits of the computing center. The revenues of the computing center are $P_2 Q_2$. Its costs are

$$C_2 = f(Q_2) \quad (21)$$

Thus the budget constraint of the organization is

$$K + P_2 Q_2 - C_2 = \sum_{i=1}^n I_i \quad (22)$$

The necessary conditions for a maximum at the organization level, can be obtained from the Lagrangian expression

$$\begin{aligned} \theta = & \sum_{i=1}^n \mu_i u_i^* \\ & + \lambda_0 (K + P_2 Q_2 - f(Q_2) - \sum_{i=1}^n I_i) \quad (23) \end{aligned}$$

Differentiating θ partially with respect to I_i , and setting equal to zero gives

$$\frac{\partial \theta}{\partial I_i} = \mu_i \frac{du_i^*}{dI_i} + \lambda_0 \left(P_2 \frac{\partial Q_2}{\partial I_i} - \frac{\partial f}{\partial I_i} - 1 \right) \quad (24)$$

$$i = 1, 2, \dots, n$$

Using equation (16), a sufficient condition for $\frac{\partial \theta}{\partial I_i} = 0$ is that

$$\mu_i \left(\frac{1}{P_2} \right) \frac{\partial U_i}{\partial q_{i2}^*}$$

$$= \lambda_0 \left(1 + \frac{\partial f}{\partial Q_2} \cdot \frac{\partial q_{i2}^*}{\partial I_i} - P_2 \frac{\partial q_{i2}^*}{\partial I_i} \right)$$

$$= \lambda_0 \left[1 - \frac{\partial q_{i2}^*}{\partial I_i} \left(P_2 - \frac{\partial f}{\partial Q_2} \right) \right],$$

$$i = 1, 2, \dots, n. \quad (25)$$

Note that $\frac{\partial f}{\partial Q_2}$ is the marginal cost to the organization of an additional unit of computer output. The expression $\left(P_2 - \frac{\partial f}{\partial Q_2} \right)$ is the marginal contribution earned by the organization from the computer center as a result of a unit increase in the quantity demanded at a constant price. The term $\frac{\partial q_{i2}^*}{\partial I_i} \left(P_2 - \frac{\partial f}{\partial Q_2} \right)$ is the computing center's "marginal profit" (revenue minus costs) for each marginal dollar allocated to user i 's budget. The expression $\left[1 - \frac{\partial q_{i2}^*}{\partial I_i} \left(P_2 - \frac{\partial f}{\partial Q_2} \right) \right]$ occurring on the right hand side of equation (25) is the net amount by which the organization's remaining resources are reduced for each marginal dollar allocated to user i . This can be called the net budget drain of a dollar to user i . The left hand side of equation (25) is the increase in utility to the organization from each marginal dollar allocated to user i . Since the term λ_0 is common to all n such equations, equation (25) says that an optimal allocation of budgets to users is one in which the ratio of the marginal utility derived by the organization from a dollar allocated to user i , to the corresponding net budget drain, is the same for all users.

If computer services are priced at marginal cost, $P_2 = \frac{\partial f}{\partial Q_2}$, then the net budget drain is unity. Suppose that $\frac{\partial q_{i2}^*}{\partial I_i}$, the marginal propensity to use computer services, is positive for all users. If the price of computer services is greater than their marginal cost, that is, if $P_2 > \frac{\partial f}{\partial Q_2}$, then the net budget drain is less than unity.

In this case the computing center, at the margin, is "profitable" for the organization.* In these circumstances, the organization, in maximizing its utility would make larger budget allocations to users with a high propensity to spend money on computers than would be the case if marginal cost equals price. Simi-

*A computing center could be profitable at the margin, but still show a deficit if $\frac{\partial^2 f}{\partial Q_2^2} < 0$. Computer center costs very likely have this characteristic.

larly, if the computing center is unprofitable at the margin, so the $P_2 < \frac{\partial f}{\partial Q_2}$, the organization would allocate relatively smaller budget amounts to users with a high marginal propensity to use computer services.

Next, consider the problem of optimum price setting. Differentiating (23) partially with respect to P_2 gives

$$\frac{\partial \theta}{\partial P_2} = \sum_{i=1}^n \mu_i \frac{\partial U_i}{\partial P_2} + \lambda_0 \left[P_2 \frac{\partial Q_2}{\partial P_2} + Q_2 - \frac{\partial f}{\partial Q_2} \cdot \frac{\partial Q_2}{\partial P_2} \right] = 0. \quad (26)$$

From equation (19) $\frac{\partial U_i}{\partial P_2} = q^*_{i2} \left(\frac{1}{P_2} \right) \frac{\partial U_i}{\partial q^*_{i2}}$.

Therefore

$$\begin{aligned} \mu_i \frac{\partial U_i}{\partial P_2} &= \mu_i (-q^*_{i2}) \left(\frac{1}{P_2} \right) \frac{\partial U_i}{\partial q^*_{i2}} \\ &= -q^*_{i2} \left[\mu_i \left(\frac{1}{P_2} \right) \frac{\partial U_i}{\partial q^*_{i2}} \right] \end{aligned} \quad (27)$$

Substituting the right hand side of equation (25) for the expression in square brackets in equation (27), equation (26) can be written as

$$\begin{aligned} \frac{\partial \theta}{\partial P_2} &= \sum_{i=1}^n \left\{ -q^*_{i2} \lambda_0 \left[1 - \frac{\partial q^*_{i2}}{\partial I_i} \left(P_2 - \frac{\partial f}{\partial q^*_{i2}} \right) \right] \right\} \\ &+ \lambda_0 \left[P_2 \frac{\partial Q_2}{\partial P_2} + Q_2 - \frac{\partial f}{\partial Q_2} \cdot \frac{\partial Q_2}{\partial P_2} \right] = 0 \end{aligned}$$

From the above, it follows that a necessary condition for a university "lend" maximum is that

$$\begin{aligned} Q_2 - \left(P_2 - \frac{\partial f}{\partial q^*_{i2}} \right) \sum_{i=1}^n \frac{\partial q^*_{i2}}{\partial I_i} \\ = Q_2 + \frac{\partial f}{\partial Q_2} \left(P_2 - \frac{\partial f}{\partial Q_2} \right) \end{aligned} \quad (28)$$

Condition (28) will be satisfied if $P_2 = \frac{\partial f}{\partial Q_2}$, that is if computer services are priced at their marginal cost of production.

Effect of outside financing for some users: Model II

Suppose now that there are m users who either re-

ceive no outside financing, or so little, that the optimal budget allocation of the organization requires that they receive additional funds. This category of users can be called internally financed users. There are additional (n-m) users who receive from outside more funds than they would receive from the organization. These are externally financed users.

Rewriting equation (26) to distinguish between these two categories gives

$$\begin{aligned} \frac{\partial \theta}{\partial P_2} &= \left[\sum_{i=1}^m \mu_i \frac{\partial U_i}{\partial P_2} \right] + \left[\sum_{i=m+1}^n \mu_i \frac{\partial U_i}{\partial P_2} \right] \\ &+ \lambda_0 \frac{\partial Q_2}{\partial P_2} \left[P_2 + Q_2 \frac{\partial P_2}{\partial Q_2} - \frac{\partial f}{\partial Q_2} \right] = 0 \end{aligned} \quad (29)$$

First consider an extreme situation. From equation (19), if for any user $q^*_{i2} = 0$, then $\frac{\partial U_i}{\partial P_2} = 0$. Suppose all internally financed users are in this situation. Suppose also that for $i > m$, $\mu_i = 0$; that is the organization receives no direct satisfaction from computer usage of externally financed users. Under these conditions, a sufficient condition for $\frac{\partial \theta}{\partial P_2} = 0$ is that

$$P_2 + Q_2 \frac{\partial P_2}{\partial Q_2} = \frac{\partial f}{\partial Q_2} \quad (30)$$

The sum of the first two terms on the left is the marginal revenue of the computing center. Thus equation (30) implies that the organization should set prices so that marginal cost equals marginal revenue. This is the profit maximizing price.

Thus at one extreme, with all users internally financed, the rule for an optimum is that the price should equal marginal costs. At the other extreme, with all users externally financed, the rule for an optimum is that the price should be set so that marginal cost equals marginal revenue, which of course implies a higher price. If some users are in each category, the optimum price would be somewhere between these two levels.

Hard money versus soft money; No external financing: Model III

Hard money can be used for any purpose. Soft money is usable only for computer expenditures. Let h_i be the amount of hard money allocated to the i^{th} user, and let s_i be the amount of soft money allocated to him.

The effective constraint facing an individual user will depend on the relative sizes of $P_2 \cdot q^*_{i2}$ and h_i . If $P_2 q^*_{i2} > h_i$, then the deficit in the user's soft money

budget must be made up with hard money, and there is effectively only one budget constraint, equation (4), with $I_i = {}_h I_i + {}_s I_i$. This case has already been considered. If $P_2 q^*_{i2} = {}_s I_i$, then there are effectively two budget constraints, as follows:

$$P_1 q^*_{i1} = {}_h I_i \quad (31)$$

and

$$P_2 q^*_{i2} = {}_s I_i \quad (32)$$

This case will be considered below. Finally if $P_2 q^*_{i2} < {}_s I_i$, then computer services are essentially a free good to the i^{th} user and the only effective constraint is equation (31).

In cases where equations (31) and (32) are the effective constraints, the user's final decision can be thought of as the result of maximizing the following Lagrangian expression

$$\begin{aligned} \phi(q_{i1}, q_{i2}, \lambda_h, \lambda_s) &= U_i(q_{i1}, q_{i2}) \\ &+ \lambda_h({}_h I_i - P_1 q_{i1}) + \lambda_s({}_s I_i - P_2 q_{i2}) \end{aligned}$$

The necessary conditions for a maximum are

$$\frac{\partial \phi}{\partial q_{i1}} = \frac{\partial U_i}{\partial q_{i1}} - \lambda_h P_1 = 0,$$

and

$$\frac{\partial \phi}{\partial q_{i2}} = \frac{\partial U_i}{\partial q_{i2}} - \lambda_s P_2 = 0 \quad (34)$$

and equations (31) and (32).

Next consider the effect on a user's level of utility as a result of a change in the values of ${}_s I_i$, ${}_h I_i$, and P_2 .

$$\frac{dU_i}{d{}_s I_i} = \frac{\partial U_i}{\partial q^*_{i2}} \cdot \frac{\partial q^*_{i2}}{\partial {}_s I_i} + \frac{\partial U_i}{\partial q^*_{i1}} \cdot \frac{\partial q^*_{i1}}{\partial {}_s I_i} \quad (35)$$

From equation (31), the second term on the right will be zero. From equation (32), $\frac{\partial q^*_{i2}}{\partial {}_s I_i} = \frac{1}{P_2}$. Thus

$$\frac{dU_i}{d{}_s I_i} = \frac{\partial U_i}{\partial q^*_{i2}} \frac{1}{P_2} \quad (36)$$

Similarly, consider a change in the hard money budget.

$$\frac{dU_i}{d{}_h I_i} = \frac{\partial U_i}{\partial q^*_{i1}} \cdot \frac{1}{P_1} \quad (37)$$

Noting from equation (32) that $\frac{\partial q^*_{i2}}{\partial P_2} = -\frac{q^*_{i2}}{P_2}$, and from equation (31) that $\frac{\partial q^*_{i1}}{\partial P_2} = 0$,

$$\begin{aligned} \frac{dU_i}{dP_2} &= \frac{\partial U_i}{\partial q^*_{i2}} \cdot \frac{\partial q^*_{i2}}{\partial P_2} + \frac{\partial U_i}{\partial q^*_{i1}} \cdot \frac{\partial q^*_{i1}}{\partial P_2} \\ &= \frac{\partial U_i}{\partial q^*_{i2}} \left(-\frac{q^*_{i2}}{P_2} \right) \end{aligned} \quad (38)$$

Next consider the organization's problem of optimally allocating hard and soft money to the n users. By analogy to equation (22) the organization's overall budget constraint can be written

$$K + P_2 Q_2 - C_2 = \sum_{i=1}^n {}_h I_i + \sum_{i=1}^n {}_s I_i$$

But if equations (31) and (32) hold for every user, then

$\sum_{i=1}^n {}_h I_i = P_1 Q_1$ and $\sum_{i=1}^n {}_s I_i = P_2 Q_2$. Substituting and simplifying gives the following budget constraint

$$K - C_2 = P_1 Q_1 \quad (39)$$

In this case the central organization will seek to maximize

$$\theta = \sum_{i=1}^n \mu_i U_i + \lambda_0 (K - f(Q_2) - P_1 Q_1) \quad (40)$$

Differentiating partially with respect to ${}_h I_i$, gives

$$\frac{\partial \theta}{\partial {}_h I_i} = \mu_i \frac{\partial U_i}{\partial {}_h I_i} + \lambda_0 \left(-\frac{\partial f}{\partial Q_2} \cdot \frac{\partial Q_2}{\partial {}_h I_i} - P_1 \frac{\partial Q_1}{\partial {}_h I_i} \right) = 0 \quad (41)$$

It follows from equation (31) and the definition of Q_2 that $\frac{\partial Q_2}{\partial {}_h I_i} = 0$ and that $\frac{\partial Q_1}{\partial {}_h I_i} = \frac{1}{P_1}$. Substituting these relations, and equation (37), equation (41) can be written as $\mu_i \frac{\partial U_i}{\partial q^*_{i1}} \left(\frac{1}{P_1} \right) - \lambda_0 = 0$. Thus

$$\lambda_0 = \frac{\mu_i \frac{\partial U_i}{\partial q^*_{i1}}}{P_1} \quad (42)$$

Similarly, differentiating (40) with respect to ${}_s I_i$, gives

$$\begin{aligned} \frac{\partial \theta}{\partial sI_i} &= \mu_i \frac{\partial U_i}{\partial sI_i} + \lambda_0 \left(-\frac{\partial f}{\partial q_2} \cdot \frac{\partial Q_2}{\partial sI_i} - P_1 \frac{\partial Q_1}{\partial sI_i} \right) = 0 \\ &= \mu_i \frac{\partial U_i}{\partial q^*_{i2}} \cdot \frac{1}{P_2} + \lambda_0 \left(-\frac{\partial f}{\partial Q_2} \frac{1}{P_2} \right) = 0. \end{aligned} \quad (43)$$

Rearranging equation (43) gives

$$\lambda_0 = \frac{\mu_i \frac{\partial U_i}{\partial q^*_{i2}}}{\frac{\partial f}{\partial Q_2}} \quad (44)$$

Equations (42) and (44) are both necessary conditions for an optimum when hard and soft money budgets are used, provided that soft money is not so plentiful that equation (32) is violated. Taken together, these two equations are identical to equation (15). The latter is a necessary condition for an optimum when only hard money budgets are considered. Thus the optimum allocation of hard and soft money is one that yields the same expenditure pattern for every user as if only hard money had been used.

Hard money versus soft money: Computer capacity fixed: Model IV

Up to this point the assumption has been that the amount of computer capacity available could be varied to meet the demand. It has also been assumed that the organization has been free to choose an optimal price for computer services. These assumptions are appropriate if the time horizon is long enough so that capacity can be varied. In this section, these assumptions are reversed. Assume that capacity is fixed, and that the price of computer services varies to allocate the available supply among users. These assumptions are appropriate to a short time horizon in which capacity changes are not feasible. However since user budgets are assumed to be variable, the time horizon must be somewhat longer than the budget period.

The discussion of user level maximization in the previous section is applicable here as well, since the individual user can still control the amount of computer service he uses. The only difference is that an increase in demand by one user now leads to a higher price (and therefore decreased demands from other users) whereas before it would have led to increased capacity. It is assumed that each user accounts for a small enough part of the total usage so that he ignores the effects of his own behavior on the level of prices.

For the central organization, the Lagrangian expression to be maximized

$$\begin{aligned} \theta = \sum_{i=1}^n \mu_i U_i + \lambda_0 \left(K + P_2 Q_2 - C_2 \right. \\ \left. - \sum_{i=1}^n (hI_i + sI_i) \right) \end{aligned} \quad (45)$$

Since capacity is fixed, and the price must be free to adjust demand to the available supply, the only decision variables under the control of the university are the hard and soft money budgets of each user. The necessary conditions for a maximum are

$$\begin{aligned} \frac{\partial \theta}{\partial hI_i} &= \mu_i \frac{\partial U_i}{\partial hI_i} + \lambda_0 \left(Q_2 \frac{\partial P_2}{\partial hI_i} - 1 \right) = 0, \\ i &= 1, 2, \dots, n \end{aligned} \quad (46)$$

and

$$\begin{aligned} \frac{\partial \theta}{\partial sI_i} &= \mu_i \frac{\partial U_i}{\partial sI_i} + \lambda_0 \left(Q_2 \frac{\partial P_2}{\partial sI_i} - 1 \right) = 0, \\ i &= 1, 2, \dots, n \end{aligned} \quad (47)$$

To find a more meaningful expression for $\frac{\partial P_2}{\partial hI_i}$ and $\frac{\partial P_2}{\partial sI_i}$, consider the aggregate demand for computer services when there are both hard and soft money budget allocations. By analogy to equation (10), let

$$q^*_{i2} = d^*_{i2}(P_1, P_2, hI_i, sI_i) \quad (48)$$

be the demand schedule for computer services of the i^{th} user. The aggregate demand schedule is then

$$\begin{aligned} Q_2 &= \sum_{i=1}^n d^*_{i2}(P_1, P_2, hI_i, sI_i) \\ &= d_2^*(P_1, P_2, hI_1, hI_2, \dots, hI_n, sI_1, \\ &\quad sI_2, \dots, sI_n) \end{aligned} \quad (49)$$

Using the formula for implicit differentiation, the relevant partial derivatives are

$$\frac{\partial P_2}{\partial hI_i} = -\frac{\frac{\partial Q_2}{\partial hI_i}}{\frac{\partial Q_2}{\partial P_2}} = -\frac{\frac{\partial q^*_{i2}}{\partial hI_i}}{\frac{\partial Q_2}{\partial P_2}} \quad (50)$$

and

$$\frac{\partial P_2}{\partial_s I_i} = - \frac{\frac{\partial Q_2}{\partial_s I_i}}{\frac{\partial Q_2}{\partial P_2}} = - \frac{\frac{\partial q^*_{i2}}{\partial_s I_i}}{\frac{\partial Q_2}{\partial P_2}} \quad (51)$$

For soft money users, $\frac{\partial q^*_{i2}}{\partial_h I_i} = 0$, and $\frac{\partial q^*_{i2}}{\partial_s I_i} = \frac{1}{P_2}$,

since, by definition, these users spend all increments of hard money on noncomputer uses, and all increments of soft money on computers. For these soft-money users, the necessary condition (46) can be expressed as

$$\mu_i \frac{\frac{\partial U_i}{\partial_h I_i}}{\lambda_0} = 1. \quad (53)$$

In equation (47), the expression

$$Q_2 \frac{\partial P_2}{\partial_s I_i} = - Q_2 \frac{\frac{1}{P_2}}{\frac{\partial Q_2}{\partial P_2}} = - \left(\frac{Q_2}{P_2} \right) \frac{\partial P_2}{\partial Q_2}. \quad (54)$$

The expression on the right in equation (54) is $-\frac{1}{m}$, where m is the aggregate price elasticity of demand for computer services. Thus equations (47) can be written as

$$\mu_i \frac{\frac{\partial U_i}{\partial_s I_i}}{\lambda_0} = \frac{1}{m} + 1. \quad (55)$$

For hard money users, $\frac{\partial q^*_{i2}}{\partial_s I_i} = \frac{\partial q^*_{i2}}{\partial_h I_i}$. Denoting by δ_i , the fraction of each additional dollar of income devoted to computer services by the i^{th} hard money user, then $\frac{\partial q^*_{i2}}{\partial_s I_i} = \frac{\delta_i}{P_2}$ for hard money users. For these users equations (46) and (47) are satisfied if

$$\mu_i \frac{\frac{\partial U_i}{\partial_s I_i}}{\lambda_0} = \mu_i \frac{\frac{\partial U_i}{\partial_h I_i}}{\lambda_0} = \frac{\delta_i}{\eta} + 1 \quad (56)$$

In practice it seems likely that the demand for

computer services is price elastic. That is, a one percent decline in the price of computer services will lead to a more than one percent increase in the quantity demanded. This means that $\eta < -1$. Furthermore, since

$0 \leq \delta_i \leq 1$, the following inequality will hold true for all users.

$$\frac{1}{\eta} + 1 \leq \frac{\delta_i}{\eta} + 1 \leq 1, \quad i = 1, 2, \dots, n \quad (57)$$

When computer capacity was assumed to be variable, a necessary condition for an optimum budget allocation was that the ratios on the left of equations (53) and (55) be identical and equal to unity for all users, which in turn implied that the optimum budget allocation was one that made all users hard money users.

Under the present assumptions this conclusion does not necessarily hold. With capacity fixed, prices elastic, and all users having strictly concave utility functions, it may be desirable to give some users more soft money, and less hard money than they would have received if capacity were variable.

The logic for using soft money under these circumstances can be explained in more intuitive terms as follows. If capacity is fixed, an additional dollar of soft money allocated to a soft-money user will be spent entirely on computer services. This will represent less of a drain on the organization's resources, than a dollar of hard money allocated to such a user (all of which would be spent on non-computer goods) or a dollar (of hard or soft money) allocated to a hard money user (part of which will be spent on non-computer goods).

The dollar of soft money allocated to a soft-money user will however cause some hard money to be spent outside the organization. By trying to spend his money on computer services, the soft-money user will drive up the price of these services, making them less attractive to hard money users, who will reduce their demand for computer services, and increase their demand for other services.

BIBLIOGRAPHY

- 1 M GREENBERGER
The priority problem and computer time-sharing
Man Sci 12 888 1966
- 2 H KANTER A MOORE N SINGER
The allocation of computer time by university computer centers
J Bus 41 375 1968
- 3 S SMIDT
Flexible pricing of computer services
Man Sci 14000 1968 Also see papers by Marchand Nielsen and Singer in this volume

Priority pricing with application to time-shared computers

by MAURICE MARCHAND*

University of Chicago
Chicago, Illinois

INTRODUCTION

Where a commodity cannot be stored, the producer is generally unable to adjust the supply instantaneously to randomly fluctuating demands. When the demand is greater than the supply, some consumers can be put into queues. Since consumers can differ significantly in the urgency of their requests, important social costs may be involved if adequate procedures are not designed to favor the individuals whose requests are the most urgent.

Surprisingly enough, the contributions of economists to the solution of this problem have so far been rather limited. The first object of this study is, accordingly, to propose a framework within which various solutions can be discussed. Our general approach will be to show how priorities can improve allocation of resources and decentralize decisions. Not less important, this study is also intended to open a dialogue with the researcher whose main task is to build probabilistic models in queueing theory.

Though primarily directed to the pricing of time-shared computing systems, our approach is suitable for a great variety of situations where the lack of flexibility in production capacity causes queues to occur. Examples are the maintenance of randomly failing machines or the allotting of priorities in telephone networks. Accordingly, the first part of this study will be conducted in terms as general as possible. The first section outlines how a pricing procedure with several levels of priority can remedy the defects of the simple first-come-first-served rule. In section two a model of

general equilibrium allows us to answer the following questions: Which characteristics of the individual demands must be charged? and, What conditions must the prices satisfy in order to be Pareto optimal? The economic interpretation and the practical implications of the optimal conditions are derived in section three.

The second part of the study investigates the extent to which our results may help in the design of efficient pricing procedures for computer centers which use time-shared systems. Directions for further research are suggested.

Priority pricing

Outline of the problem

In most of the situations where users of a facility form queues to receive service, the variable production costs associated with the servicing itself are rather negligible. In such situations, any pricing mechanism, if advisable, must be primarily intended to reduce the level of disutilities that the users are causing each other through their presence in the queue. In fact, we deal here with a typical example of external diseconomies, the users' utilities being not independent of each other.

The services rendered by the facility in question are henceforth considered as the completion of jobs submitted by the users. For the same user, the urgency of completion is allowed to vary from one job to another.

If we first focus our attention upon the first-come-first-served discipline, an optimal price can, in principle, be figured out. The general principle here is that, at the optimum, the social utility of an additional job should compensate the social disutility that its presence is causing to the users of the facility. Let us assume that the incomes are optimally distributed among the individuals. Since the additional job is marginal, its social utility is then correctly evaluated by its private cost, i.e., the money amount it is charged under the cur-

*An early draft of this paper was published as Working Paper No. 247 at the Center for Research in Management Science, University of California, Berkeley while the author was a C.R.B. Graduate Fellow 1967-1968 (Belgian American Educational Foundation). The author wishes to thank Professor C. B. McGuire for his personal encouragement, and J. P. Brown and E. Sharon for helpful discussions.

rent price. On the other hand, the social disutility of its presence may be estimated by adding up the money amounts that the users are willing to pay not to see this additional job compete with their jobs for the facility.

One may criticize this approach in two ways. On the one hand, the social disutility caused by an additional job can only be estimated very approximatively. On the other hand, the first-come-first-served discipline rules out any possibility of discriminating among the jobs according to their urgency. Since all submitted jobs are treated on the same basis, more urgent jobs can be delayed by less urgent jobs for a long time.

Now let us show how an allocation rule based on several levels of priority can remedy the shortcomings of the first-come-first-served rule. From the user's standpoint, several levels of priority are to be distinguished by the expected queueing time that the user must expect to face on entering a job into a specific priority class—shorter queueing time being associated with a higher priority. It is up to the manager of the facility to specify in a contract these expected queueing times and to adjust the prices for the various levels of priority in such a way that his commitments will actually be fulfilled.

Pricing priorities allows the ranking of the submitted jobs in the queue according to the urgency of their completion. Knowing the set of prices, the user is, in fact, able to decide into which priority class to enter a particular job by considering both its urgency and the relationship that links priority and price.

Furthermore, the pricing mechanism allows to optimally allocate the resources of the whole economy between the facility in question and the rest of the production sector. By relating his receipts and the investment costs, the manager of the facility can decide what capacity is optimal.

The model

The method followed in this section is to maximize a linear combination of individual utility functions subject to technological constraints and market clearing equations. The dual variables associated with the market clearing equations will turn out to be the prices. Some conditions will be found that the prices must satisfy in order to lead to a Pareto optimal allocation of resources. Before doing so, the notation and the assumptions of the model will be presented, and some results of queueing theory will be recalled. In particular, these results will allow us to determine which characteristics of the demands for jobs must be charged.

In order to focus our attention upon the problem of pricing priorities, we consider an economy whose n consumers ($i = 1, \dots, n$) derive satisfaction from two forms of consumption: first, the services rendered by the

facility in question, referred to as jobs, and second, a so-called composite commodity which summarizes all other goods available in the economy.

The limited capacity of the facility and the short-run stochastic fluctuations of the demand for jobs lead to the occurrences of queues at random times. The manager of the facility establishes levels of priority ($p = 1, 2, \dots$). In accord with normal usage, priority level 1 refers to highest priority, 2 to next highest priority, and so on. As is pointed out in the previous section, the manager specifies the average queueing time Q_p that a p -priority job must expect to face. Of course

$$Q_1 < Q_2 < Q_3 \dots \quad (1)$$

Let us recall that, as usual, queueing time is here defined as the duration of the period which elapses between the instant at which the job arrives at the facility and the instant at which it starts being served.

By assumption, the jobs are served one at a time in order of priority and within priorities in order of arrival. In the case where a job of higher priority arrives when a job of lower priority is being served, we assume that the facility interrupts the current service and immediately starts serving the job of higher priority. The service of the job of lower priority is resumed from the point of interruption when no more jobs of higher priority are present.*

Let us divide the jobs into streams according to their levels of priority. The priority streams of jobs are assumed to arrive at the facility in accordance with independent Poisson processes.** Each consumer is allowed to enter jobs into each priority stream. Let λ_p^i ($i = 1, \dots, n; p = 1, 2, \dots$) be the arrival rate of the p -priority jobs of consumer i .

Each job is characterized by its service time, i.e., the duration of the period between the instant at which the facility starts serving the job and the instant at which the job is finished provided that no interruption occurs. The service times of the jobs are supposed to be mutually independent random variables and independent of the arrival times. Denote by $H_p^i(z)$ the probability that the service time Z_p^i of a p -priority job of consumer i is $\leq z$

$$H_p^i(z) = \Pr(Z_p^i \leq z), \\ i = 1, \dots, n; p = 1, 2, \dots \quad (2)$$

*In queueing theory, this service policy is referred to as the pre-emptive-resume case. As a matter of fact, our model implicitly assumes that the facility does not waste any time on turning its attention from one job to another.

**To focus attention on the stochastic fluctuations of the demand for jobs, the model ignores its short-run periodicity. This is implicitly involved in the assumption of Poissonian arrivals.

Let α_p^i and β_p^i be the two first moments of the random variable Z_p^i

$$\alpha_p^i = \int_0^\infty z dH_p^i(z), \quad i = 1, \dots, n; p = 1, 2, \dots, \quad (3)$$

$$\beta_p^i = \int_0^\infty z^2 dH_p^i(x), \quad i = 1, \dots, n; p = 1, 2, \dots. \quad (4)$$

The service time of the jobs depends directly upon the capacity, or the speed of the facility. Since the speed of the facility will be a key decision variable, it must be stressed here that the above distribution functions of service times and their two first moments are defined with reference to a facility of unit speed.

Next, let us define

$$a_p^i = \lambda^i \alpha_p^i, \quad i = 1, \dots, n; p = 1, 2, \dots, \quad (5)$$

$$b_p^i = \lambda^i \beta_p^i, \quad i = 1, \dots, n; p = 1, 2, \dots. \quad (6)$$

Henceforth the quantities a_p^i and b_p^i will be referred to as the first and second moments of consumer i 's p -priority stream.

In the same way, the first and second moments of the aggregate p -priority stream, a_p and b_p , can be defined as

$$a_p = \sum_{i=1}^n a_p^i, \quad p = 1, 2, \dots, \quad (7)$$

$$b_p = \sum_{i=1}^n b_p^i, \quad p = 1, 2, \dots. \quad (8)$$

Finally, let

$$\bar{a}_p = \sum_{r=1}^p a_r, \quad p = 1, 2, \dots, \quad (9)$$

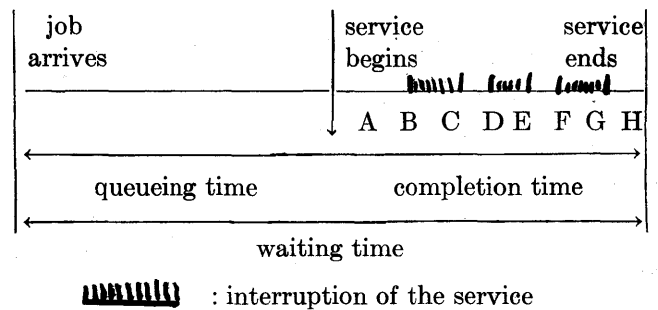
$$\bar{b}_p = \sum_{r=1}^p b_r, \quad p = 1, 2, \dots. \quad (10)$$

From Takács¹ it can be shown that, in order that the actual expected queueing times do not go beyond the specified times Q_p , the first and second moments of the aggregate streams, a_p and b_p ($p = 1, 2, \dots$), and the speed of the facility, s , should satisfy the following tech-

nological constraints*

$$Q_p \geq \frac{\sum_{r=1}^p b_r}{2 \left(s - \sum_{r=1}^p a_r \right) \left(s - \sum_{r=1}^{p-1} a_r \right)} = \frac{\bar{b}_p}{2(s - \bar{a}_p)(s - \bar{a}_{p-1})} \quad p = 1, 2, \dots. \quad (11)$$

Up to now, we have argued as if the satisfaction that a consumer derived from a stream of completed jobs depended only upon the expected queueing time. As a matter of fact, his satisfaction depends on the average waiting time, which is equal to the sum of the average queueing time and of the average completion time. The completion time is defined as the duration of the period that elapses between the instant at which the job starts being served and the instant at which it is completed (taking into account the interruptions due to the occurrences of higher priority jobs). It can be shown that the average completion time of a p -priority job is $(s - \bar{a}_{p-1})^{-1}$ times its expected service time (on a facility of unit speed).



$$AB + CD + EF + GH = \text{service time of the job}$$

The preferences of the consumers are assumed to be representable by convex, twice continuously differentiable utility functions

$$U^i = U^i(x^i; a_1^i, b_1^i; a_2^i, b_2^i; \dots; \bar{a}_1, \bar{a}_2, \dots, s), \quad i = 1, \dots, n, \quad (12)$$

*For the priorities labeled with numbers greater than some value, the expected queueing times are so long that even without any charge, few users are willing to submit jobs at these priority levels. As a consequence, the number of jobs is not large enough to make the actual expected queueing time reach the specified time. This is taken into account by the inequality sign in (11).

where x^i denotes the quantity of composite commodity consumed by i . As a matter of fact, the satisfaction a consumer derives from the facility depends on the arrival rates λ_p^i ($p = 1, 2, \dots$) and the distribution functions of service times $H_p^i(z)$ ($p = 1, 2, \dots$). However, because of the way these characteristics of the individual streams of jobs appear in the technological constraints (11), only the first and second moments of the individual streams, a_p^i and b_p^i ($p = 1, 2, \dots$), are relevant here. As for the speed of the facility s and the parameters \bar{a}_p ($p = 1, 2, \dots$), they are included in the utility functions to take account of their influence on the completion times.

Finally, the production possibilities per period of the economy are constrained by the convex, twice continuously differentiable function

$$f(x, s) = 0 \tag{13}$$

in which x represents the quantity of the composite commodity supplied per period.** The transformation function (13) involves the maintenance and the replacement of a facility of speeds.

The conditions for a Pareto optimum will be found by maximization of a linear combination (with arbitrary weights) of the individual utility functions, subject to the appropriate constraints, namely

$$\max_{\substack{x^i, \bar{a}_p^i, \bar{b}_p^i, a_p, \\ b_p, \bar{a}_p, \bar{b}_p, s, x}} \sum_{i=1}^n \omega^i U^i(x^i; a_1^i, b_1^i; \bar{a}_2^i, \bar{b}_2^i; \dots; a_1, a_2, \dots, s) \tag{14}$$

subject to

$$a_p = \sum_{i=1}^n a_p^i, \quad A_p \quad p = 1, 2, \dots \tag{14}$$

$$b_p = \sum_{i=1}^n b_p^i, \quad B_p \quad p = 1, 2, \dots \tag{15}$$

$$\bar{a}_p = \sum_{r=1}^p \bar{a}_r, \quad \bar{A}_p \quad p = 1, 2, \dots \tag{16}$$

$$\bar{b}_p = \sum_{r=1}^p \bar{b}_r, \quad \bar{B}_p \quad p = 1, 2, \dots \tag{17}$$

**The transformation function (13) implies that the facility is operated without variable costs of production and that the maintenance costs are independent of its use. These assumptions are chosen to avoid complicating the discussion.

$$x = \sum_{i=1}^n x^i, \quad \chi \tag{18}$$

$$Q_p \geq \frac{b_p}{2(s - \bar{a}_p)(s - \bar{a}_{p-1})}, \quad \gamma_p \quad p = 1, 2, \dots \tag{19}$$

$$f(x, s) = 0, \quad \phi \tag{20}$$

where the symbols on the right stand for the dual variables associated with the constraints.

The first-order conditions for a maximum are (in addition to (14) through (20))

$$\omega^i \frac{\partial U^i}{\partial x^i} - \chi = 0, \quad i = 1, \dots, n, \tag{21}$$

$$\omega^i \frac{\partial U^i}{\partial a_p^i} - A_p = 0, \quad i = 1, \dots, n; p = 1, 2, \dots, \tag{22}$$

$$\omega^i \frac{\partial U^i}{\partial b_p^i} - B_p = 0, \quad i = 1, \dots, n; p = 1, 2, \dots, \tag{23}$$

$$\sum_{r=p}^{\infty} \bar{A}_r - A_p = 0, \quad p = 1, 2, \dots, \tag{24}$$

$$\sum_{r=p}^{\infty} \bar{B}_r - B_p = 0, \quad p = 1, 2, \dots, \tag{25}$$

$$\begin{aligned} \bar{A}_p - \gamma_p \frac{\bar{b}_p}{2(s - \bar{a}_p)^2(s - \bar{a}_{p-1})} \\ - \gamma_{p+1} \frac{\bar{b}_{p+1}}{2(s - \bar{a}_{p+1})(s - \bar{a}_p)^2} \\ + \sum_{i=1}^n \omega^i \frac{\partial U^i}{\partial \bar{a}_p} = 0, \quad p = 1, 2, \dots, \tag{26} \end{aligned}$$

$$\bar{B}_p - \gamma_p \frac{1}{2(s - \bar{a}_p)(s - \bar{a}_{p-1})} = 0, \quad p = 1, 2, \dots, \tag{27}$$

$$\varphi \frac{\partial f}{\partial s} + \sum_{p=1}^{\infty} \gamma_p \bar{b}_p \frac{1}{2(s - \bar{a}_p)(s - \bar{a}_{p-1})} \left[\frac{1}{s - \bar{a}_p} + \frac{1}{s - \bar{a}_{p-1}} \right] + \sum_{i=1}^n \omega^i \frac{\partial U^i}{\partial s} = 0, \quad (28)$$

$$\chi + \varphi \frac{\partial f}{\partial x} = 0, \quad (29)$$

$$\gamma_p \left(Q_p - \frac{\bar{b}_p}{2(s - \bar{a}_p)(s - \bar{a}_{p-1})} \right) = 0, \quad p = 1, 2, \dots \quad (30)$$

Our first task will be to see how the prices emerge from our analysis. Under the restriction that the incomes r^i are such that

$$\omega^i \frac{\partial U^i}{\partial r^i} = \omega^j \frac{\partial U^j}{\partial r^j}, \quad i, j = 1, \dots, n, \quad (31)$$

the first-order conditions (21), (22), and (23) are simply the conditions for a maximum of the individual utility functions. From that, one can infer that, up to a change of dimension, A_p and B_p stand for the prices of the first and second moments of the p -priority stream and χ for the price of the composite commodity.

Before going further, let us define

$$P = \max \left\{ p: Q_p = \frac{\bar{b}_p}{2(s - \bar{a}_p)(s - \bar{a}_{p-1})} \right\}. \quad (32)$$

For the priority levels labeled with numbers greater than P , the expected queuing times are such that the stream of jobs is not large enough to make active the corresponding constraints (19).

From (30), one can conclude that

$$\gamma_p = 0 \quad \text{for all } p > P. \quad (33)$$

From (25) and (27), it then follows that

$$B_p = \bar{B}_p = 0 \quad \text{for all } p > P. \quad (34)$$

Substituting the γ_p 's from (27), the first-order conditions (26) can be rewritten as follows

$$\bar{A}_p = \frac{\bar{b}_p}{s - \bar{a}_p} \bar{B}_p + \frac{\bar{b}_{p+1}}{s - \bar{a}_p} \bar{B}_{p+1} - \sum_{i=1}^n \omega^i \frac{\partial U^i}{\partial a_p}, \quad p = 1, 2, \dots, P - 1 \quad (35)$$

$$\bar{A}_P = \frac{\bar{b}_P}{s - \bar{a}_P} B_n - \sum_{i=1}^n \omega^i \frac{\partial U^i}{\partial a_P} \quad (36)$$

Together with (24) and (25), (35) and (36) provide the first-order conditions that the prices of the facility must satisfy in order to be Pareto optimal.

Finally, substituting φ from (29) and the γ_p 's from (27), (28) can be rewritten as follows

$$-\chi \frac{dx}{ds} = \sum_{p=1}^P \bar{b}_p \bar{B}_p \left[\frac{1}{s - \bar{a}_p} + \frac{1}{s - \bar{a}_{p-1}} \right] + \sum_{i=1}^n \omega^i \frac{\partial U^i}{\partial s}, \quad (37)$$

in which dx/ds stands for the transformation rate between x and s

$$-\frac{dx}{ds} = \frac{\partial f}{\partial s} / \frac{\partial f}{\partial x} \quad (38)$$

To be Pareto optimal, the speed of the facility must satisfy the first-order condition (37).

Interpretation and implication of the results

A. Let us imagine that the manager of the facility would be only concerned about his commitments on the expected queuing times, and would ignore the conditions that his prices must satisfy in order to be Pareto optimal. Since the values of the a_p 's and b_p 's satisfying the active technological constraints (11) are not uniquely determined, the manager would have some degree of freedom in the choice of his prices. However, if the manager of the facility is also concerned about the efficiency in resource allocation, the conditions (35) and (36) restrict his possible choices.

B. From (24) and (25), one can rewrite the first-order conditions (35) as follows

$$A_p = \sum_{r=p}^{\infty} \frac{1}{s - \bar{a}_r} [\bar{b}_r(B_r - B_{r+1}) + \bar{b}_{r+1}(B_{r+1} - B_{r+2})] - \sum_{r=p}^{\infty} \sum_{i=1}^n \omega^i \frac{\partial U^i}{\partial \bar{a}_r},$$

$p = 1, 2, \dots$ (39)

Rearranging the terms of (39) and taking account of (34), we obtain

$$A_p = \frac{\bar{b}_p}{s - \bar{a}_p} B_p + \left[\frac{\bar{b}_{p+1} - \bar{b}_p}{s - \bar{a}_p} + \frac{\bar{b}_{p+1}}{s - \bar{a}_{p+1}} \right] B_{p+1} + \sum_{r=p}^{P-2} \left[-\frac{\bar{b}_{r+1}}{s - \bar{a}_r} + \frac{\bar{b}_{r+2} - \bar{b}_{r+1}}{s - \bar{a}_{r+1}} + \frac{\bar{b}_{r+2}}{s - \bar{a}_{r+2}} \right] B_{r+2} - \sum_{r=p}^{P+1} \sum_{i=1}^n \omega^i \frac{\partial U^i}{\partial \bar{a}_r}$$

$p = 1, 2, \dots, P$. (40)

Our task will be now to provide an economic interpretation of the last conditions. For this purpose, let us increase a_p by an infinitesimal increment while keeping constant the first moments of the aggregate priority streams. In order that the active constraints (11) remain satisfied, it is then necessary to modify the b_p 's. Since the occurrence of a higher priority job interrupts the servicing of lower priority jobs, the p -priority jobs do not affect the jobs with smaller priority numbers. It follows that a shift in a_p must be offset only by shifts in b_p, b_{p+1}, \dots and b_P . Differentiating with respect to a_p, b_p, b_{p+1}, \dots and b_P those of the constraints (11) which correspond to the priority levels p to P , we obtain

$$\frac{db_p}{(s - \bar{a}_p)(s - \bar{a}_{p-1})} + \frac{\bar{b}_p da_p}{(s - \bar{a}_p)^2 (s - \bar{a}_{p-1})} = 0, \quad (41)$$

$$\frac{db_p + \dots db_r}{(s - \bar{a}_p)(s - \bar{a}_{r-1})} + \bar{b}_r \left[\frac{1}{(s - \bar{a}_r)^2 (s - \bar{a}_{r-1})} + \frac{1}{(s - \bar{a}_r)(s - \bar{a}_{r-1})^2} \right] da_p = 0,$$

$r = p + 1, \dots, P$. (42)

Dividing everywhere by (da_p) , we have a system of equations whose unknowns are the rates of technical substitution: the Q_p 's being held constant

$$\left(-\frac{db_p}{da_p} \right), \left(-\frac{db_{p+1}}{da_p} \right) \dots \text{and} \left(-\frac{db_P}{da_p} \right).$$

Solving the system, we obtain

$$-\left(\frac{db_p}{da_p} \right) = \frac{\bar{b}_p}{s - \bar{a}_p}, \quad (43)$$

$$-\left(\frac{db_{p+1}}{da_p} \right) = \frac{\bar{b}_{p+1} - \bar{b}_p}{s - \bar{a}_p} + \frac{\bar{b}_{p+1}}{s - \bar{a}_{p+1}}, \quad (44)$$

$$-\left(\frac{db_{r+2}}{da_p} \right) = -\frac{\bar{b}_{r+1}}{s - \bar{a}_r} + \frac{\bar{b}_{r+2} - \bar{b}_{r+1}}{s - \bar{a}_{r+1}} + \frac{\bar{b}_{r+2}}{s - \bar{a}_{r+2}}, \quad r = p, \dots, P-2 \quad (45)$$

Finally, introducing these results into (40), we have

$$A_p = -\sum_{r=p}^P \left(\frac{db_r}{da_p} \right) B_r - \sum_{r=p}^{P+1} \sum_{i=1}^n \omega^i \frac{\partial U^i}{\partial \bar{a}_r},$$

$p = 1, 2, \dots, P$. (46)

Under this form, the economic interpretation of the first-order conditions on the prices is straightforward. Let us increase a_p by an infinitesimal quantity, Δa_p . Since marginal, the social utility of this increment is correctly evaluated by $(A_p \Delta a_p)$. However, the increase in a_p imposes to change the b_p 's in order that the expected queueing times do not go beyond the specified values Q_p . The social disutility due to these shifts in the b_p 's is estimated by Δa_p times the first term on the right-hand side of (46). On the other hand, an increase in a_p causes the completion times of the jobs with priority numbers equal to and greater than p to be extended, the social disutility of which is reflected in the second term on the right-hand side of (46). One can then conclude that condition (46) expresses that at the optimum, the social utility of an increase in a_p should offset its social disutility.*

C. The first-order condition (37) on the optimal speed of the facility can be interpreted in exactly the same way. Let us simply remark that the left-hand side of (37) stands for the marginal cost of the speed.

D. As can be seen from (37) and (46), our approach fails to completely avoid resting on an evaluation of the disutility that the users of the facility are causing each

*Even if the capacity of the facility is not optimal, the conditions (46) on prices remain optimal.

other. However, one can point out that the last terms on the right-hand side of (37 and (46) refer only to the effects of s and the a_p 's on the completion times of the jobs. Without expressing any opinion on the importance of these terms, the extent to which our pricing proposal relies upon the evaluation of external disutility is clearly much less significant than in the case where the simple first-come-first served rule is adopted.**

E. Pricing both the first and second moments of the individual priority streams affects the behavior of the consumer in two ways. First, the consumer will be enticed to reduce the service time variance of the jobs he enters into each priority stream. Second, the consumer will be encouraged to make his job as short as possible. To show this, let us first imagine that the consumer i generates a p -priority stream of arrival rate λ^i_p , and that the service time is not random, say α'^i_p . He would then be charged $\lambda^i_p \alpha'^i_p A_p + \lambda^i_p (\alpha'^i_p)^2 B_p$. Let us now imagine that he could manage to make twice as short the time of his jobs while doubling his arrival rate. His charge would then be reduced by $\frac{1}{2} \lambda^i_p (\alpha'^i_p)^2 B_p$.

F. The adjustment process through the prices has so far been assumed to take place prior to the beginning of the period considered. Given any prices announced by the manager, each consumer lets him know the first and second moments of the individual streams he would enter at each level of priority. By trial and error, the manager could then adjust the prices so as to fulfill his commitments about the expected queueing times.

As a matter of fact, the process toward equilibrium is, in the real world, carried out through "tatonnement" over time. This fact, at least in principle, does not involve any further difficulties. Each job performed by the facility will be charged individually, one component of the price being proportional to its service time and the other to its squared service time.***

**As far as conditions (37) are concerned, this failure is due to the assumption that the service of a job is interrupted when a higher priority job is submitted. Excluding interruptions makes the expected completion time of a job equal to $(s)^{-1}$ times its expected service time. In this case, the expected completion time is no longer a function of the \bar{a}_p 's.

***Important remarks must be made at this point. Both the service time and its square being charged, rational behavior would lead the user to divide each of his jobs into as many separate jobs as possible. As a consequence, instead of submitting one job at a time, he would simultaneously submit a number of jobs. Applying without qualifications the results of the model would then bring about violation of the assumption of Poissonian arrivals on which our model is based. The only way to avoid this conclusion is to charge such jobs as one unique job, whose service time is obtained by adding up the service times of the separate jobs.

On the other hand, the assumption of Poissonian arrivals implies that no information is provided to the consumer about the queueing size at the time he takes his decision. Otherwise, he could vary his arrival rate with the instantaneous queueing size.

Application to a particular case: the time-shared computer

The time-sharing systems we are concerned with are ones in which users have access to the same computer through remote consoles. Many users may be simultaneously connected to the computer, i.e., on-line. Typically jobs performed on a time-shared computer, such as those involving trial-and-error procedures, are of a conversational nature. Thus the amount of computation required per request from on-line terminals is usually much less than that required by typical batch jobs. While on line the users spend a considerable amount of time evaluating the results of their previous requests and preparing their future requests; neither of these require computing time. Rapid turn-around is essential to a time-sharing system because users are waiting at their consoles for replies to their requests without engaging in other productive activity.****

At first sight, the technical characteristics of time-sharing systems seem to accommodate the assumptions upon which our results in the first part rest. In particular, the time-shared computers have high flexibility in rescheduling their operations so that interrupting the servicing of a job does not raise any technical difficulties. Therefore, one could apparently use our results to design an efficient procedure for charging the services rendered by a time-shared computer, especially the computation time of the requests according to their priority level. Putting things in this way, we are implicitly assuming that the central processing unit is the only element which limits the capability of a time-shared computer. Other elements, however, should be considered as scarce resources, namely the core memory and the entry channels whose number limits the possible number of users on line. It follows that efficiency in resource allocation requires that prices paid by users reflect their use of all scarce resources: computation time, memory utilization, and access time.

Another salient feature of time-shared computers, which the assumptions of our model do not accommodate is switch time, i.e., the nonproductive period during which the central processing unit is turning attention from one request to another.† Within the proposed scheduling procedure, the number of switchings may become large since the servicing of an incumbent request is allowed to be interrupted if a higher priority request is submitted. Let us define the technical performance of the scheduling technique as the per-

****An introduction to economic aspects of time-sharing can be found in W. F. Bauer and R. H. Hill.³ See also M. Greenberger.³

†Switching from one request to another may involve, for example, swapping programs and data from the core memory to the auxiliary memory and vice versa.

centage of time the central processing unit spends in productive computation. This technical performance may be achieved only at the expense of the economic efficiency of the scheduling procedure. Excluding interruptions entirely would lead to maximum technical performance, but then a request of lower priority may delay a request of higher priority for a long time, contrary to efficient operation of the facility from an economic point of view. This means that the technical performance should be somewhat sacrificed in order to achieve economic efficiency. On the other hand, to interrupt the servicing of a request that is nearly completed at the occurrence of a higher priority request may cause inefficiency in the allocation of computer time. For instance, one can think of the following rule. In the case where a request of higher priority occurs, the decision to interrupt the servicing of an incumbent request would depend on three factors: the expected switch time, the expected time to complete the incumbent request, and the relative levels of priority of the two jobs in question.

More general remarks can be made about the realism of our analysis. First, no attention has been given to the dispersion of the waiting time around its expected value. As is well known, the more rapidly the slope of the cost (or disutility) function of waiting time increases for a certain request, the more important it is to take into consideration the variance of the waiting time.* Going even further, to disregard moments of order higher than two is correct only if either the waiting time is normally distributed or the cost function of waiting time is quadratic.** Though apparently less crucial here than for the jobs executed on a batch basis,*** this problem deserves further research.

Second, the major purpose of the proposed pricing and scheduling procedure is to distinguish the requests according to their urgency. In principle, this attempt requires that the expected queueing times for two adjacent priority levels be as close as possible, and consequently that the number of priority levels be large. However, the cost of implementing the proposed scheduling technique increases with the number of priority levels. Our general equilibrium model completely ignores these costs which must be taken into account while implementing the proposed scheduling technique.

*Discontinuities in the cost function of waiting time, due to deadlines for instance, seem less likely to occur than for the jobs executed on a batch basis.

**This assertion can be inferred from the theory of portfolio selection. See Tobin.⁴

***During each of his on-line stays, the user usually generates several requests. The larger their number is, the better the average of queueing time per request is approximated by the expected queueing time (law of large numbers).

Two interrelated questions arise here. First, for any particular number of priority levels being imposed, what expected queueing time should be assigned to each of these priority levels? Second, what is the optimal number of priority levels? An optimal compromise should clearly be found between the cost of implementing the pricing procedure and its efficiency to distinguish the requests according to their urgency.*

Third, the proposed pricing remains essentially static, in the sense that the prices are fixed for a certain period of time (say one month), and are not allowed to change according to the stochastic short-run fluctuations of the demand. At the time an idle period occurs, a "dynamic" pricing scheme would reduce the rates to fill the gap. Such dynamic pricing requires that the potential users be aware of the rates at any time. A time-sharing system can apparently fulfill this requirement without significant costs since information channels exist from the computer center to the user's console.** Before implementing a dynamic pricing scheme, many difficulties must be overcome, and the development of additional results in queueing theory is not the least.

Further, in the case when a user is not aware of the exact amount of computation time that a particular request will require, he may encounter very complex problems to decide at which priority level to enter his request. In fact, the price he will be charged for choosing any priority level is uncertain. To alleviate this difficulty, the user might, in advance, order the computer center to stop the computation and to place the incomplete request at a lower priority level in the case where the request would require a computation time larger than some specified quantity.

Fifth, the periodic short-run fluctuations of demand have been ignored in order to focus attention upon its stochastic fluctuations. In our opinion, taking account of the demand periodicity does not raise any theoretical difficulties. The prices should vary according to the periodicity in the demand, and be fixed so that the technological constraints (11) on the expected queueing times are satisfied at any time. The optimal condition on the capacity, or the speed, of the central processing unit could easily be redefined to take account of this periodicity.

*Fundamentally, the problem we deal with here is to optimally choose the commodity space, namely its dimension (number of priority levels) and the qualitative characteristics of included commodities (expected queueing times).

**The argument collapses if the communication network between the computer system and the users' consoles mainly use telephone lines which are not private.

All these critical comments, the list of which is undoubtedly not closed, show the limits of our results. Hopefully, this study provides a framework within which improvements of the current pricing procedures of computer centers can be discussed.

REFERENCES

- 1 L TAKACS
Priority queues
- 2 W F BAUER R H HILL
Operation Research 12 63-74 1964
Economics of time-shared computing systems
Datamation Vol 13 11 48-55 and 12 41-49 1967
- 3 M GREENBERGER
The priority problem and computer time sharing
Management Science 12 11 888-906 1966
- 4 J TOBIN
The theory of portfolio selection
In *The Theory of Interest Rates* Frank Mahn ed Macmillan
1965

Flexible pricing: An approach to the allocation of computer resources

by NORMAN R. NIELSEN

Stanford University
Stanford, California

INTRODUCTION

The past twenty years have seen the rapid development of an extremely powerful new tool—namely the electronic digital computer. The technology associated with these machines has advanced rapidly, permitting the development of capabilities which only a few short years ago would have staggered the imagination. Man has been equally progressive with respect to the application of this tool in his endeavors. In particular, the area of business management has built rapidly upon the advantages offered by computer processing. Although the computer only served as a glorified bookkeeping machine in the earliest applications, its usage quickly developed.

Today there are complex production scheduling and planning programs, sales forecasting and analysis routines, elaborate simulations, and much work is going into the development of large-scale information systems. Yet, despite the rapidity with which management has seized upon the computer for assistance in making better business decisions, there has been little movement by computer center managements to bring management techniques to bear upon the administration of computer services. This is particularly surprising in the light of today's multimillion dollar investments in computing equipment.

Although there are many management problems involved in operating any computer installation, this paper is focused primarily upon those problems associated with the making of resource allocation decisions. These problems become particularly acute for managers of large, centrally located systems which serve many diverse general purpose users.

Background

Partly for historical reasons and partly because of the pervasiveness of government funding of computer users, there has been a great emphasis upon the costing of computer services. Although the various costing procedures adequately recover the costs of an operation, they all fail to allocate the scarce computing resources in an appropriate manner. For example, under indirect costing the expenses connected with the acquisition and operation of a computer are charged against various overhead accounts, so that the user does not "pay" in proportion to his usage. Thus the computer is looked upon as something of a "free good," and the demand for service readily exceeds the supply. Since the computer can provide only a certain maximum quantity of service in a unit of time, a form of de facto allocation must take place. For example, long turnaround times may serve to discourage a sufficient amount of usage to bring supply and demand into balance. Certain administrative rules may also be used to enable the center to provide a more suitable service for the majority of its users (e.g., only jobs of 10 minutes or less can be run during the prime shift).

Further, in any environment there are always certain users who request special treatment, such as exemption from the standard scheduling procedures. Often a center's management is forced to pass judgment upon the importance or worth of these projects in deciding whether or not to grant such requests.* In summary indirect cost-

*Alternatively, personal friendships with computer operators or other such considerations may decide these questions.

ing results in perpetual saturation, so that management has no guide as to when additional capacity should be installed. Furthermore, allocation of the available computing resources must be made on somewhat arbitrary grounds rather than for maximum user benefit.

On the other hand, under direct costing the user pays the cost of his usage; the more he runs, the greater his charge. Thus management has some gauge as to what the real demands on the center are. However, the allocation problem still remains. Since the cost to provide a given amount of computing is nearly independent of the turnaround or service given to any particular job, resort must still be made to administrative or other somewhat arbitrary procedures for determining the service which a job is to receive.

A further effect of direct costing stems from the fact that the provision of computer services can be characterized in the short run as a high fixed cost, low marginal cost operation. Thus, the cost per unit of computing is greatly influenced by the total usage of the facility. When demand is heavy, the cost is low, thereby encouraging even further usage and worsening the turnaround situation. When demand is light, the cost is high, discouraging use of an already lightly loaded facility. In situations where users are required to use the cheapest facilities available, work is driven off lightly loaded systems to more heavily loaded ones; just the opposite of what would be desired for the most efficient utilization of the available resources.

There is, however, a growing awareness of these problems. For example, the Committee on Governmental Relations of the National Association of College and University Business Officers discussed these issues in Washington, D.C. (July 11-12, 1967), and a conference on computer pricing was held at the RAND Corporation (August 2-3, 1967). A number of individuals have also made proposals.^{1,2,3,4} One of the most widely mentioned alternatives is some form of pricing. That is, rates which are set so as to govern usage rather than to reflect strictly the cost of providing the service. Suggestions run all the way from a level price for the life of a computing system to a flexible price that can be adjusted in response to demand as it varies from year to year, or from month to month, or even from hour to hour. Needless to say there is much discussion about the mechanisms to be used to adjust prices. Proposals range all the way from periodic reappraisals by

management to dynamic adjustments based upon the actual workload. Despite their diversity all of these proposals have one thing in common; they are concerned with the use of prices to allocate computer resources in a rational and effective manner.

Unfortunately, despite the desirability of pricing, most of the discussions which have been carried out in this area have treated the computer as a single resource. Thus, a price is used to allocate "computer time." Under many older batch processing systems, the user tied up all of the resources of the system when he ran (by preventing others from using them) even though he himself was not actually using all of them. Under these conditions the use of a single price is quite satisfactory.

However, we are now faced with a quite different situation. In today's multiprogrammed and/or time-shared computing environments there is no single resource that can be priced in order to allocate the usage of the entire system, for there are a multitude of resources which must be allocated. Further, a single user need not utilize all the resources of the system at any one time, and it is possible that other users may be able to utilize some of the remaining resources. Accordingly, there has been a movement toward charging each user for only those resources which he actually uses or otherwise renders unavailable to others. Already many computer centers are employing multirate structures. For example, the University of Michigan Computation Center has a set of nine rates covering the use of its system.

However, in order to allocate the resources of the entire system so as to maximize the utilization of those resources, it is necessary to have not just an array of prices but an array of flexible or adjustable prices that are responsive to demand. In other words service as well as computer resources must be taken into consideration. Just as prices are used to allocate the limited resources of the national economy, so can responsive pricing be used to allocate the limited resources of a computing system.

Furthermore, today's computer systems are modular in design. It is possible to adjust the number and speeds of disk storage units, to adjust the number of I/O channels, etc. Often even the size and speed of the memory can be altered. Thus, the management of a computer center is faced with a much more difficult situation than the de-

termination of whether or not additional capacity in the form of a larger or an additional computer is required. It is now necessary to make such decisions on each of the individual components of the system. Thus, it is more important than ever to be able to have pricing data for comparison with acquisition and installation costs in order to assist in making appropriate decisions.

Thus, by having a set of flexible prices which are responsive to the demands and requirements of the users, it is possible for the system's resources to be allocated as effectively as possible. Further, such a structure would enable management to make better decisions with respect to the quantities of the various resources to be provided. This paper discusses the considerations which are involved in developing a flexible pricing procedure and describes an experiment, currently being conducted, in which such a pricing procedure is being employed in the operation of a computer center.

Utilization measurements

Before proceeding to the development of a flexible pricing structure, it is well to consider the various measurements which might be made concerning a job's utilization of a computing system. A list of potential measures is shown in Table I; an example of a possible measurement unit for each item is shown in brackets. It should be noted that these measures are for a generalized time-sharing or multiprogramming system with a hierarchical secondary or file storage organization. In any particular system, software design or hardware availability may preclude certain of these measures.

The first four measures are concerned with a job's utilization of the "main system." Elapsed time measures the period during which a job is in some sense "on" the system, while compute time indicates the period for which the CPU was employed. The memory measures are concerned with both the extent and the duration of memory occupancy. Although the page is used to illustrate the units of space, any other unit could be employed equally well.

The next eight measures are concerned with the I/O operations associated with the processing of a particular job. Swap space measures the amount of space required over time for the storage of program images. This would be applicable only in a system in which a job need not be totally core resident for the duration of its elapsed time (i.e., a system employing roll-in roll-out, swap-

TABLE I—Possible utilization measures

ET:	Elapsed Time, the time from the start to the finish of a job [seconds]
CT:	Compute Time, the actual CPU time used by a job [seconds]
MT:	Memory Time, the integral over time of memory space occupied by a job [page seconds]
PMT:	Productive Memory Time, the integral over time of memory space occupied by a job while CPU cycles were being used [page seconds]
SS:	Swap Space, the integral over time of secondary storage space used by a job for swap image storage (or overlay storage, or virtual memory storage) [page seconds]
ST:	Swap Time, the amount of time spent swapping (overlapping or paging) a job in and out of main memory [seconds]
IOT _a :	Input Output Time, the sum of a job's I/O times for file references employing device <i>a</i> (tape drive, disk unit, device controller, I/O channel, PPU, etc.) [seconds]
ION _a :	Input Output Number, the number of I/O operations initiated by a job for file references employing device <i>a</i> [operations]
CR:	Cards Read, the number of input cards for a job [cards]
CP:	Cards Punched, the number of output cards for a job [cards]
LP:	Lines Printed, the number of output lines for a job [lines]
PT:	Plotting Time, the time from the start to the finish of a job's plot [seconds]
SU _e :	Set Ups (of tape reels, disk packs, special forms, etc.) required by a job on devices of class <i>e</i> [set ups]
FS _f :	File Storage, the integral over time of storage space for a user's files on unit <i>f</i> [page days]
TR:	Tape Reels, the integral over time of the number of reels reserved for a user [reel days]
DP:	Disk Packs, the integral over time of the number of packs reserved for a user [pack days]

ping, paging, dynamic overlaying, or other such technique). This is also true for the swap time measure, which is merely an indication of the amount of information swapped or otherwise transferred in and out of the main memory.

The other six I/O measures are concerned with the amount of I/O caused directly by a job (i.e., file references). Input output time and number provide measures of a job's usage of the various peripheral devices, control units, channels, peripheral processing units, etc. The measures in terms of cards and lines provide an indication of productive work, independent of the speed of the particular device and independent of operator interventions for card jams, paper breaks, etc.

The set ups measures provide an indication of the special work which was required on the part of the operations staff in order to enable a job to be processed. This would encompass the mounting of reserved tapes, the mounting of private

disk packs, and the loading of special forms on a printer, etc.

The last three measures in the table are concerned not so much with the servicing of a job as they are with the servicing of a user over a period of time. Thus, file storage, tape reels, and disk packs measure services which are provided to a user independent of the particular jobs he may have processed. In principle all of these measures are integrals over time. However, it is more likely to be practical to inventory a user's holdings on a daily or other periodic basis.

It should be noted that the list of measures shown in Table I is by no means complete. Rather it is an attempt to indicate the type and range of measures which one could employ to record a job or user's utilization of the various system resources. Not only are there other measures which might be used, but there are also variations of those listed. At the same time it should be realized that it is exceedingly unlikely that any single installation would wish to make use of all of these measures. Finally, it should be noted that the concern at this stage is with individual user's utilization of the system, not with the overall performance of the system. In the latter case there are a number of other measures (such as length of I/O queues, percentage of available CPU cycles productively employed, etc.) which would be of great interest.

Relating resources to utilization measures

In order to develop prices to associate with the aforementioned measures of usage, it is necessary to relate the various resources of the system to these measures. Although this might appear to be a rather straightforward procedure, it turns out to be more complicated in actual practice. First, it is necessary to select only those measures to which users can easily relate their resource usage. For example, in a multiprogrammed environment the elapsed time and the memory time of a job are not directly related to that job's activities. Since the user perceives very little direct control over his usage as indicated by these measures, there is little to be gained from the attachment of prices to these measures (other than the recovery of dollars). The prime motivation for a price structure is to influence usage, so the user must perceive himself as able to exert control over his usage.

Secondly, it is important to select only those measures which can be obtained easily from the

particular computer system in question. No matter how effective a particular pricing procedure might be in allocating a system's resources, if it requires excessive amounts of system resources to operate, it will be inferior to an otherwise less effective procedure that employs fewer resources. For example, on some systems it is a very time-consuming process to measure the length of I/O operations. Accordingly, it might be more effective to employ the number of I/O operations performed as a surrogate for the volume of information transmitted. Although such a substitution would not be without side effects (e.g., encouraging fewer but longer I/O transmissions), it still might prove to be advantageous.

Third, the same reasoning can be applied to the user himself. If the pricing procedure requires too much of his own personal resources (e.g., time), he will tend either to (a) go elsewhere to do his computing or (b) ignore the pricing system altogether. In either case this type of behavior defeats the purpose of having a pricing procedure in the first place.

In summary, it is desirable to select as few measures as possible, but these must be measures of activities over which the user has substantial control and which taken together account for a major portion of the system's resources. Table II illustrates how resources in a time-shared or multiprogrammed system might be related to the various measures developed in the previous section.

It should be noted though that for any particular system these assignments would be modified in response to the unique characteristics of that system. Consider, for example, some of the alternatives with respect to memory usage. If it is possible for a job doing a great deal of I/O and very little computing to tie up memory space during all of the I/O transmissions, it would be preferable to use memory time as the usage measure. On the other hand, if it is possible for a job to be stranded in memory for long periods of time because other jobs do not relinquish the CPU, it would be preferable to use productive memory time as a measure. If the multiprogramming algorithm is such that one job is always given first priority for the CPU, it might be desirable to use memory time as a measure for that job and productive memory time as a measure for the other jobs in the system.

The use of swap space and swap time also depend upon the particular system used. If every

TABLE II—Possible relations between measures and resources

Elapsed Time:	terminals, transmission control units, telephone switching gear, terminal buffers in memory, any other portions of the system which are tied up for the duration of the life of a job
Compute Time:	central processing unit
Memory Time or Productive Memory Time:	high speed memory
Swap Space:	drums or highest speed disks
Swap Time:	drum (or disk) control units, swap channels, CPU cycles stolen
Input Output Time or Input Output Number:	tape units, disk drives, device control [units], I/O channels, PPU's, etc.
Cards Read:	card reader, reader control unit, I/O channel
Cards Punched:	card punch, punch control unit, I/O channel
Lines Printed:	printer, printer control unit, I/O channel
Plotting Time:	plotter, plotter control unit, I/O channel
Set Ups:	operations staff, any system facilities disabled during the set up period.
File Storage:	disks or other units on which on-line files are maintained
Tape Reels:	tapes, storage facilities for tapes
Disk Packs:	packs, storage facilities for packs

job is given a fixed amount of drum space, independent of the actual amount of space required, drum space could just as well be related to the elapsed time measure. If the user does not have substantial control over the volume of data which is swapped the drum, control unit, and channel resources could be related to swap space or elapsed time.

Input output time and number are viewed as measures of file activity. In principle they could also be used for the peripheral input output units. However, a measure such as the number of cards read is more likely to be meaningful to a user than the number of seconds of card reader time on a certain card reader. This is particularly true when there are similar devices having different speeds and when the user has little or no control over the actual device employed. The use of measures such as cards read or lines printed also eliminates the concern about card jams, paper breaks, and other incidents over which the user has little control.

Pricing

The next step in the procedure is to assign a

set of prices to the various measures which have been selected. This too can be a difficult task. Not only is it necessary to set prices to allocate the available system resources in an appropriate fashion, but it is also necessary that the usage of the resources in each category which will result at the selected prices be such as to provide a sufficient aggregate revenue for the center.* Meeting this latter requirement can be a problem when there are a number of special considerations which dictate that certain prices be relatively low (high) in order to encourage (discourage) particular types of system usage (see below).

Since in most cases it is not possible for a user to shift readily the mix of resources which his programs require, it is desirable that the relationships between the various prices or base rates do not fluctuate rapidly. Even weekly adjustments could cause undue grief for many users. On the other hand computing loads do change during the year, if only because of seasonal or secular trends. Accordingly, some price adjustments must be made from time to time. Thus, management might plan to review system performance monthly, bimonthly, or quarterly depending upon the characteristics of their installation.

If usage in a particular category dwindles, it is an indication that the price should be reduced. If the price falls too much below "full cost" it is an indication that there is over capacity for that service and that some of those resources should be removed from the system (unless there are unusual circumstances or requirements which haven't been considered). The opposite situation holds in the case of excessive demand. Clearly a price increase will ration demand, but if the price rises above "full cost," it is a signal that additional capacity is required. It would, however, be a very unusual situation in which it would be necessary to adjust every base rate as frequently as quarterly.

As was mentioned previously it is often desirable to use the price structure to influence users' behavior. Table III illustrates how charges might be levied for usage of the various categories of service. Again, the specific implementation would be dependent upon the particular system as well as the goals and usage patterns of the installation. Most of the charges are based upon flat

*Note that in situations where the interests of the users are economically consistent with those of the center, it is total value (revenue plus consumers' surplus) rather than revenue that is the relevant measure.

rates, so nothing more need be said. However, some comment is appropriate for the other charges.

If most of an installation's users work during the day, management might judge it desirable to provide as short a turnaround service as possible during the prime shift.** In which case the compute time rate should be an increasing function of the compute time. This would discourage long running jobs during the daytime, but it would not exclude them entirely. It would thus be up to each user to determine whether his job was sufficiently important to be run during the day, and a center's management would not have to pass judgment upon the relative worth of specific projects. By the same token, during the night when turnaround is less important, long running jobs could be encouraged by means of a reduced or decreasing rate. The same type of argument can also be applied to printing, plotting, etc. Generally, though, long jobs in these areas are not as significant a problem since the lower per unit costs of the peripheral devices usually enables a larger number of these units to be provided to handle peak loads.

In many installations the handling of a few cards of punched output can be particularly troublesome. The holes in the card have to be scanned to separate and identify the cards belonging to different jobs, and then the cards must be associated with the printed output for those jobs. As a result it may well be desirable, from the point of view of operations, to discourage the casual punching of cards. At the same time it is desirable to avoid impacting those users who do indeed have need for large numbers of punched cards. Accordingly, a fixed charge could be levied in addition to a flat rate per card. This would make the punching of a few cards relatively expensive, but would have little effect upon the total charge for large punching jobs.

A comment is also in order about the rates for file storage. If there is a hierarchy of on-line file storage devices, then there should be a hierarchy of storage rates with each rate being related to the speed of a class of device (in terms of access time and transfer rate). Each user could then determine for his own particular situation the trade-offs between greater speed and higher rates.

**This would presume that the greater productivity of the majority resulting from the more rapid turnaround would more than offset the reduced productivity of the minority having longer jobs.

TABLE III—Possible base charge structure

Elapsed Time:	$A \cdot (ET)$ where A is the rate in cents/second
Compute Time:	$B_i \cdot (CT)$ for prime shift jobs where i is a monotonically increasing function of CT and $B_1 < B_2 < \dots < B_n$, $C_j \cdot (CT)$ for non-prime shift jobs when j is a monotonically increasing function of CT and $C_1 > C_2 > \dots > C_m$ and where B_i and C_j are rates in cents/second
Memory Time:	$D \cdot (MT)$ where D is the rate in cents/page second
Productive Memory Time:	$D' \cdot (PMT)$ where D' is the rate in cents/page second
Swap Space:	$E \cdot (SS)$ where E is the rate in cents/page second
Swap Time:	$F \cdot (ST)$ where F is the rate in cents/second
Input Output Time:	$\Sigma G_a \cdot (IOT_a)$ where G_a is the rate for device a in cents/second
Input Output Number:	$\Sigma G'_a \cdot (ION_a)$ where G'_a is the rate for device a in cents/operation
Cards Read:	$H \cdot (CR)$ where H is the rate in cents/card
Cards Punched:	$K + L \cdot (CP)$ where K is a fixed charge in cents and L is the rate in cents/card
Lines Printed:	$P_i \cdot (LP)$ for prime shift jobs where i is a monotonically increasing function of LP and $P_1 < P_2 < \dots < P_n$, $Q_j \cdot (LP)$ for non-prime shift jobs where j is a monotonically increasing function of LP and $Q_1 > Q_2 > \dots > Q_m$ and where P_i and Q_j are rates in cents/line
Plotting Time:	$R \cdot (PT)$ where R is the rate in cents/second
Set Ups:	$\Sigma S_c \cdot (SU)$ where S_c is the rate in cents/set up for devices of class c
File Storage:	$\Sigma U_j \cdot (FS_j)$ where U_j is the rate for unit j in cents/page day
Tape Reels:	$V \cdot (TR)$ where V is the rate in cents/reel day
Disk Packs:	$W \cdot (DP)$ where W is the rate in cents/pack day

There could also be a separate rate for users wishing to rely upon the system to select the most appropriate locations for their files.*

Within this framework it is also possible to work out a low contractual rate for each device. This rate would be applied to users who contract in advance for specified large quantities of storage space for quarterly, yearly, or other time periods. This offers an inducement to large users

*This presumes the existence of a "percolate-trickle" procedure whereby files are brought up to fast storage when they begin to be used and are trickled down to slower and slower devices as they become and remain inactive.

to "inform" management of the long term nature of their requirements, even if they will not require all of that space every day. The reduced rate can be interpreted as a rebate for assuming some risk-bearing from management or as a volume discount (or both).

The only other charges shown which do not result from flat rates are those from input output operations and for setups. In each case there is an array of rates based upon the devices or device classes actually used, and the charge is the sum of the products of those rates with the respective amounts of usage. The number of rates employed can be made as large or small as desired.

The charge structure described above and outlined in Table III is by no means a recommended one for every situation. Rather it is an attempt to indicate the kinds of considerations that can be incorporated into such a structure. Since the needs of installations will vary, items in the charge structure must be modified or adjusted in order to reflect the particular goals or operating considerations of an installation.

Flexible pricing

The prices constructed in the previous section are concerned with the allocation of the computer's resources over relatively long periods of time (e.g., quarterly). That is, they are designed to influence the long run behavioral patterns of a facility's user population. In many cases this will suffice for an effective allocation of many of the resources. For example, in most centers a single card punch can provide ample capacity, so that there is rarely any bottleneck in providing punched card output. Likewise the number of magnetic tape reels stored for users does not fluctuate sharply during the day.

On the other hand there are some resources for which demand may vary significantly over the course of a day or week. For example, consider compute time. During certain periods of the day it would not be surprising if the quantity of compute time demanded would exceed the quantity supplied by a factor of three or four. Yet over the course of a full day the CPU would at times be idle. In other words there is ample capacity on average, but there is insufficient capacity (by a significant amount) during the peak periods. The pricing procedures outlined thus far are concerned only with the allocation of the system's resources to users (i.e., how much compute time will a particular user be permitted to receive);

they are not concerned with the timing with which those resources will be provided.

Compute time is not the only resource for which there may arise sharp fluctuations in demand. During the day there may be many more users seeking time-sharing service than the system can accommodate. Accordingly, someone must be denied access. The same may be true for access to a printer or for access to a certain amount of high speed memory or for access to some other resource of a particular system. However, service or turnaround is merely another aspect of the computer center's offerings. It is a valuable (and limited) resource which must be allocated just like the other resources. Given that there are several users wanting to run batch jobs at a particular point in time, it is clear that some of those jobs are going to have to wait. The question is which will have to wait and for how long. It is the aim of flexible pricing to resolve this type of problem, so that administrative edicts or other arbitrary procedures need not become involved in the allocation of service.

One often-heard proposal is for the system to compute the rates dynamically as a function of the load or waiting demand. However, this results in the user knowing neither the price he will be charged nor the service he will receive. Further, even if estimates are satisfactory, this approach assumes that users finding the system heavily loaded will go away and return later. In many cases the user does not have this degree of flexibility (or patience).

Another proposal would utilize a bidding procedure whereby the highest bidder would be the next to receive the desired computer resources. Although this would maximize revenue, it has several disadvantages. Either the user must stay on-line so that he can participate in the bidding from time to time, or he must leave a program that does his bidding for him. In either case a substantial amount of system resources as well as user resources would be required. Further, if the user does not develop a sophisticated bidding procedure, he will not have any clear idea of when his job might be completed.

The most attractive proposal is a modification of the bidding procedure. A user would request a quote on a particular amount of a resource within a particular time period. For example, he might request three minutes of compute time within the next two hours. Based upon the existing load and estimates of future demands, the system would quote a rate. If this is satisfactory

to the user, a contract is made covering both price and service. If the quote is too high the user can request a new quote for the resource but over a longer time period, etc. Unfortunately, however, no one has developed a program to perform this kind of calculation in an appropriate manner while employing an acceptable level of system resources.

A compromise approach is based upon the use of priorities. Not only is this technique effective in this situation, but other investigators of time-sharing systems have found it useful as well.^{5,6} For the case in question a number of queues could be set up. Associated with each would be a priority and a price (or rate). Jobs would be selected for service from the front of the highest priority non-empty queue. If a user is concerned about turnaround, he can scan the workload waiting in the various queues and select a queue appropriate for his needs. If he is concerned about price, he can select the queue having a rate appropriate for his requirement. There is, however, a greater uncertainty about turnaround time associated with the lower priorities. This can be minimized to some degree by providing the user with an ability to query the system to determine the progress of his job (and to change the priority if he is unsatisfied with the service).

One of the most frequent criticisms of the multi-queue procedure concerns the problems faced by users trying to develop long term computing budgets, since they must determine in advance what priorities they will need in order to process their jobs. However, no matter what type of an allocation procedure is instituted, it is not possible to specify both the rate and the level of service (unless the user is in some sense privileged). The problem stems from the fact that users are currently accustomed to fixed rates for computing (with variable service). Now they would like to take advantage of the opportunity to fix service (but without having to give up fixed rates).

Table IV shows how flexible pricing might be employed in allocating some of a system's resources. Although the table illustrates a multiplicative structure (a constant times the base charge), an additive structure could also be used (a flat charge added to or subtracted from the base charge). It should be noted that for most computer systems it would not be necessary to use flexible prices for both compute time and memory space, since normally only one of these factors is controlling.

TABLE IV—Possible flexible price structure

Elapsed Time:	W_i (base charge) where i is the number of terminals on the system and the W_i are real numbers ≥ 1 and $W_1 \leq W_2 \leq \dots \leq W_n$.
Compute Time:	X_j (base charge) where j represents the priority of service and the X_j are real numbers.
Productive Memory Time:	Y_k (base charge) where k represents the priority of service and the Y_k are real numbers.
Lines Printed:	Z_l (base charge) where l represents the priority of service and the Z_l are real numbers.

The situation for time-sharing access* is somewhat different. Although users' jobs can tolerate delays before they are processed, the users themselves are unwilling to spend periods of time at a remote terminal waiting to gain access to the system. A user either gains access or he gives up and tries again at some other time; he rarely waits. Accordingly, a set of differential waiting times is of little use since all waiting times are very nearly equally unacceptable. Thus, a somewhat different queue structure is necessary.

Rather than denying access to an additional user when the system is "full," it would be possible to let the new entrant bid against the other users already on the system to see whether he or one of them will be denied access. However, this is likely to be very unsatisfactory to those already on the system. Having gained access they wish to concentrate upon the problem which they are attempting to solve. Being interrupted every few minutes to bid against a potential new entrant would not only be very annoying to a user but also quite detrimental to his productivity.

An alternative is to raise the elapsed time rate for new entrants as the number of users begins to approach the system's capacity. By making the price increases sufficiently sharp, the center can manage to have at least one line into the system available most of the time. Thus, users would seldom encounter the "busy signal" problem which denies them access without any alternative. Under the increasing rate procedure a user can hang up without penalty if he does not feel that immediate access is as important as the quoted rate.

*Note that access refers to the connection of a terminal to the system and is unrelated to the response of the system to requests from that terminal.

But if access is important, he will be able to gain it even though the system is heavily loaded. The size of the step function in the rate and the number of lines or ports which are affected will depend to some degree upon the user population of an installation. However, the extensive use of the higher priced access lines is a clear indication to management concerning the need for additional terminal handling capability.

Because of a user's vulnerability once he is in the midst of a terminal session, it is not desirable to vary the rates for users already on the system. On the other hand, in order to prevent abuse, it may be necessary to limit the time period for which a user's rate would apply. After such a period (e.g., one hour) the user would be treated like a new entrant; he would be quoted the current entering rate and would be given the option of accepting (i.e., continuing) or of signing off. Otherwise, users entering at a slack time would not feel any economic pressure to shorten their sessions if demand subsequently increased.

The Stanford experiment

Despite the apparent advantages afforded by flexible pricing, it is important to demonstrate how satisfactorily it might serve in an actual operating environment. Further, the implementation and utilization of such a procedure in a large computer center would also provide an indication of the operational problems that might be encountered. Fortunately, the management of the Stanford Computation Center was very receptive to participating in such a test.

Management's interest in pricing had arisen some time previously when an acute shortage of magnetic tapes had developed at the Campus Facility of the Stanford Computation Center. Users were reserving tapes at an ever increasing rate; yet they were releasing very few tapes. Pleas were made for users to release all tapes that were not absolutely essential to their work, but this met with almost no response. Finally, in an attempt to cover the mounting tape costs, management decided to levy a nominal charge of \$1/tape/month. On the first day that the charge was instituted more than $\frac{1}{3}$ of the "absolutely essential" reserved tapes were released. Management has been a firm believer in pricing ever since.

Thus, it was decided to attempt to allocate the computing resources of the Center's Campus Facility through flexible pricing. These resources consist of a large-scale third generation comput-

ing system with a corresponding amount of peripheral equipment. On a typical day this system is called upon to process some 1500 instructional and research jobs. The system provides a batch processing capability from on-line card readers located at the Facility as well as a text editing, file handling, and remote job entry capability from some 100 remote terminals.

There was, however, one more hurdle to surmount before a flexible pricing scheme could be implemented. In addition to serving the student computing load, the Campus Facility also provides computing services to faculty and staff research projects. Since many of these projects are funded by the federal government, the computing charges must be allowable costs for the projects. In order for these charges to qualify, it is necessary that the Campus Facility operate with a direct costing procedure in compliance with the applicable government regulations on costing specialized facilities.*

Accordingly, negotiations were entered into with the government in an attempt to obtain a waiver from these requirements. Although the government representatives were sympathetic to the operational problems encountered under direct costing, they were reluctant to permit any deviation from existing regulations. However, costing problems were becoming more serious with the growing use of large-scale multiprogrammed and/or time-shared systems. Thus it was finally agreed to conduct a pilot experiment in order to obtain data about the use and effectiveness of computer pricing procedures.

The experiment was authorized for the Campus Facility of the Stanford Computation Center during the period September 1967 - June 1968. Essentially the government agreed to permit Stanford to employ a pricing scheme during the period of the experiment so long as the prices were not discriminatory with respect to a user's source of funds. Stanford also agreed to operate the Campus Facility under a long-term no-profit rule and to provide the government with full disclosure on the results of the experiment.

The base charge structure was introduced in September at the beginning of the experiment. Because of the particular characteristics of the Stanford system, a number of modifications were made to the charges previously described. The resulting structure is similar to but a subset of

*Bureau of the Budget Circular A21, Paragraph J-37, pertaining to specialized service facilities operated by an institution.

the one illustrated in Table III.

At the present time there is a shortage of high speed memory on the Campus Facility system, so that only a single batch job is being processed at any one time. Accordingly, the CPU rate absorbs the functions of the memory time and productive memory time rates. Each terminal user on the system is automatically assigned the maximum permissible amount of swap space. Accordingly, the swap space rate is reflected in the elapsed time rate. In the terminal system the user has only a limited I/O capability; in the batch system (since there is but a single job at any one time) the user pays for compute time during many of his I/O waits. Accordingly the swap time and input/output rates were dropped.

The rate structure for compute time has also been modified slightly. Rather than charging one of a set of rates for the entire amount of compute time, a step function is used. There is a base rate for daytime computing and a somewhat lower base rate for overnight computing. During the day the base rate is increased by 50% for all time in excess of five minutes and by 100% for all time in excess of ten minutes. During the night the rate is reduced by approximately 7% of the base rate after every thirty minutes of continuous processing by a job. Thus long jobs, which are discouraged during the day, are encouraged at night.

With respect to card punching, the structure shown in Table III has been augmented with a higher rate for the first 1000 cards punched by a job. Because of the number of line printers available and because of the characteristics of the Stanford user population, there has been little problem with excessive printing by jobs. Accordingly there is but a single rate for printing, independent of the time of day and the amount of printing.

Due to the characteristics of the operating system, there are setup charges only for the mounting of private disk packs and for the substitution of special forms on a printer. Also due to hardware problems there is only one level of secondary storage for files; hence, there is but a single file storage rate. All the other rates are of the form indicated in Table III.

Flexible prices were introduced in February after the users had gained some experience with the system. In the Stanford implementation a combination of the multiplicative and additive forms is used. Since a job requiring very little compute

time is rather insensitive to rate changes, it was necessary to employ increasing fixed charges for higher compute time priorities. Otherwise the rate changes necessary to influence the small compute time users would overly impact larger users. On the other hand a standby or "when idle" priority is available at a reduction in rate.

Default print priorities are assigned on the basis of the amount of printing required. These priorities can be over-ridden for a fixed charge. The very highest print priority consists of a fixed charge plus a doubling of the rate.

At the present time there is no flexible price for elapsed time or terminal access duration since the system is able to handle the present level of demand. However, with the introduction of a time-sharing system in the near future, such a capability will be implemented.

Although the experiment is still under way, there is every indication that it will be successful. Most of the users have become well versed with the pricing procedure and make appropriate use of it. As a result the overall performance of the computing system has been very good. The premium rates for compute time have restricted the submission of long jobs for daytime running, such that the average turnaround time for all batch jobs averages about one hour. On the other hand long executing jobs are not excluded. Approximately 2½% of the daytime jobs execute for more than five minutes and approximately ½% exceed ten minutes.

Because of the low average turnaround time, there is not an excessive use of the higher priorities of service. However, when backlogs have accumulated (near the end of the daytime service block, in times of hardware trouble, or following the processing of long jobs), there is a substantial increase in priority usage. In general users have been quite happy with this ability to gain superior service when their circumstances require it.

A further indication of the value which users place upon good service stems from their use of the idle priority. Jobs in this category are granted a discount on the compute time charge but are processed only when there is no other work for the system to perform. Initially this discount was set at 25%, but less than 1% of the jobs representing less than ½ of 1% of the workload were submitted with this priority. An increase in the discount to 50% succeeded only in doubling usage, so that jobs in this category remain a relatively small portion of the total work-

load. In other words, to a vast majority of users, good service is more attractive than even a 50% discount.

From management's point of view the system is very successful. No longer must decisions be made on whether or not a particular user's plight is sufficiently urgent to warrant processing his job ahead of the previously submitted work of other users. Also, the use of idle and overnight rates has encouraged users to identify those of their jobs for which rapid turnaround is not required. This enables service to the other users to be improved. Finally the operation of the pricing procedure has proven to be quite practical, requiring less than 1% of the system's CPU cycles. Although the present pricing structure will be further adjusted or changed as additional experience is gained, the initial stages of the experiment have shown this type of procedure to be very satisfactory.

SUMMARY

This paper has presented a procedure whereby flexible prices can be used for the allocation of a computer system's resources in such a way as to improve the utilization of the available resources and to increase the users' utility or satisfaction. Further, these prices can be used to encourage or discourage particular types of system usage or user behavior which have an impact upon the performance of a particular computer system.

The pricing structure which has been outlined should serve to illustrate the concept. Although it

is unlikely that any installation would employ the structure exactly as set forth, the concepts are sufficiently general that they can be applied with little difficulty in many different environments. Not only are the prices flexible, but the goals which they can serve are flexible as well.

Such a pricing procedure is currently undergoing test in an actual operating environment and has proven quite successful thus far. It is demonstrably practical from an operating standpoint, and it has met with acceptance by the users. In addition, it has assisted in improving the performance of the computer facility while freeing management from having to make numerous allocation decisions.

REFERENCES

- 1 W F SHARPE
Computer use charges
Unpublished paper University of Washington Nov 1965
- 2 SENKE H KANTER A MOORE N SINGER
Proposed pricing of computer centers with government contracts
Washington D C Office of the Assistant Secretary of Defense System Analysis August 1966
- 3 S SMIDT
A flexible price system for computers
Working paper Cornell University May 1967
- 4 A MOORE H KANTER N SINGER
The allocation of computer time by university computer centers
To be published in the Journal of Business
- 5 M GREENBERGER
The priority problem and computer time-sharing
Management Science Vol 12 No 11 July 1966 pp 888-906
- 6 M MARCHAND
Priority pricing with application to time-shared computers
Working Paper No 247 Center for Research in Management Science University of California Berkeley March 1968

Data structures and techniques for remote computer graphics

by IRA W. COTTON

Sperry Rand Corporation
Philadelphia, Pennsylvania

and
FRANK S. GREATOREX, JR

Adams Associates
Bedford, Massachusetts

INTRODUCTION

It has been adequately demonstrated^{1,2,3} that computer graphics systems need not require the dedication of a large scale computer for their operation. Computer graphics has followed the trends of computing in general, where remote access, time sharing, and multi-programming have become the key phrases. The problems involved in providing a remotely accessed, interactive computer graphics system are more formidable than for a dedicated system, or even than for a local time shared system. The requirement is basically to obtain a real time response for the display console operator, while at the same time minimizing the overhead imposed on the central computer facilities. Also present, of course, are the classic graphical problems such as that of providing refresh data for cathode ray tube displays,

and of relating the appearances of a picture on the tube face to its description in the data structure.

Figures 1 and 2 illustrate a typical configuration as it

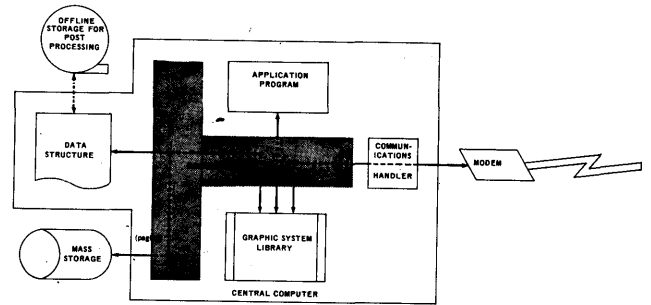
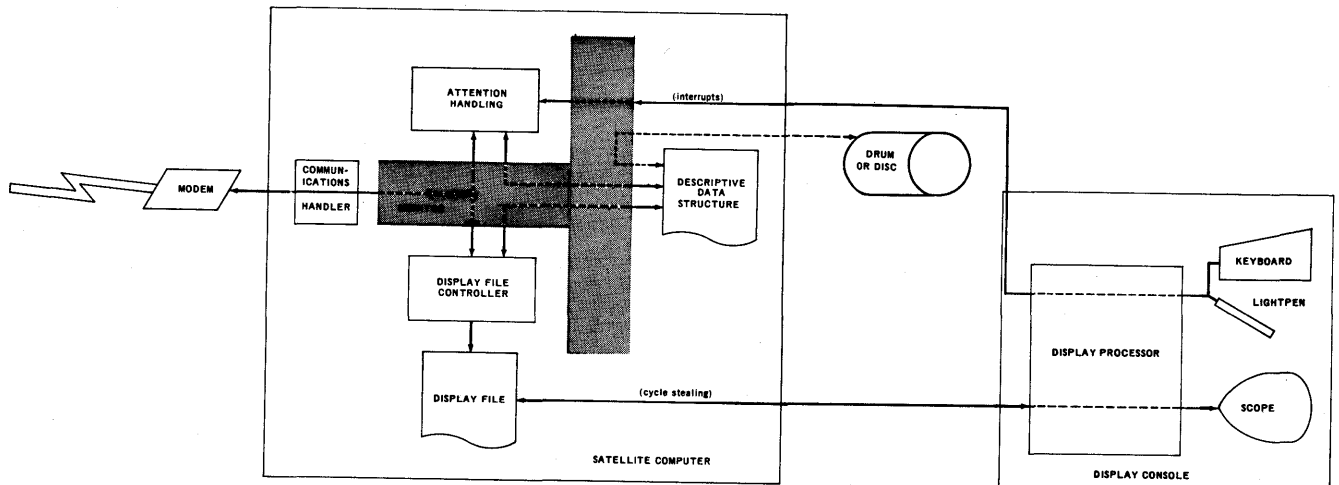


FIGURE 1—Graphics system organization in the central computer

FIGURE 2—Graphics system organization in the satellite computer



would appear to a single user. Actually, the central computer is time-shared by a number of users, both graphic and non-graphic, and there may be a number of other special features. For example, if the graphic system library is reentrant, the same copy may be used by all application programs. There may also be a large number of remote stations, and it may be possible for one satellite processor to drive more than one display. A number of configurations have been described^{1,2,4} and it is not the intent of this paper to describe a particular hardware implementation. Such an implementation is currently in progress and will be described at a later date, but the system has been designed to be essentially hardware independent in the sense that the implementation either in the central or satellite computers may be recoded for different or improved hardware, while still maintaining the same interface with each other and with the application program. The structure of the system is also sufficiently modular so that enhancements such as the ability to support additional input devices may be added without great difficulty. This paper will describe this system, show how the data structure is an integral and dynamic part of it, and focus on how interactions with the data structure occur both at the central and satellite computers.

System software requirements

Interactive graphics systems should provide the means for the basic functions of computing, data management, image generation, and attention handling. Certain software is required to provide these facilities in a remotely accessed environment (central site software for time-sharing is assumed):

- 1) There must be an *application program* which provides ultimate control of the work being done by the user at the display console.
- 2) There should be a *data base* with sufficient structure to provide the applications program, the display presentation software, and any post processing routines with the means of defining and integrating the actions of the user in formulating valid images.
- 3) There should be a group of *service routines* with the ability to manipulate the central site data structure under the direction of the application program.
- 4) There must be *telecommunication routines* for the control of messages between the remote graphics processor and the central site computer.
- 5) There must be a *monitor program* in the remote computer to build and maintain the display file from the description of the picture in the data base and to direct attention handling at the satellite so

as to provide acceptable real-time response to operator actions at the display console.

Application program

The application program is provided by the user. Several points are significant. First, the user is completely free to perform whatever computations are required as an adjunct to his display requirements. Second, the user may have an independent data base apart from the graphics system, or it may be integrated into the graphics system. Normally, the interface between application programs and the system is desired via FORTRAN-type subroutine calls. Application programs will not be considered further in this paper.

Data base

The system data base, referred to as the *Entity Table*, is a hierarchically organized data file, dynamically expandable and modularized to permit paging. Structures in this list are indexed by a hash-coded directory, also modularized, for the purpose of rapid retrieval. The creation, manipulation and retrieval of structures in the Entity Table are performed by system supplied subroutines. The list processing technique of ring chaining⁴ is extensively employed to improve the efficiency of scanning the data base. Since the length of each entry varies, depending on its type, garbage collection routines are embedded in the system unbeknownst to the user, to prevent excessive fragmentation and minimize storage requirements.

At this point, it might be asked why a data structure is required at all for computer graphics applications. The main reasons for using a hierarchically organized data structure such as will be described may be summarized as follows:

- 1) It appears to arise normally from the organization of the class of pictures normally considered in computer graphics;⁵
- 2) It allows the system to take advantage of the graphical subroutining capability being built into current display hardware;⁶
- 3) Such a data structure is important not only to the display, but also for the use of the applications program presumably directing the course of the graphical processing (for example, the same data structure used to store the description of an electronic circuit for display could also contain the data used in the analysis of the circuit).

A viable data structure is crucial to the success of any computer graphics system, especially if it is resolved that the system is to serve as a useful tool, rather than just an expensive toy.

The basic philosophy of the data structure is to embed the facility for hierarchical picture definition within a list processing scheme with sufficient power to allow easy manipulation and retrieval of the pictorial elements, and sufficient flexibility to be essentially hardware independent both in its own design and in the description of pictorial elements.

This goal of hardware independence has a number of virtues. First of all, hardware independence means that the applications programmer need not be burdened with worrying about hardware idiosyncrasies of the display device. Secondly, it means that the system can be employed to drive a number of output devices, including digital plotters for hard copy, and is amenable to post-processing of various sorts (the area of numerically controlled tools comes immediately to mind). Finally, hardware independence at the central computer means that a consistent application program interface could be maintained if that computer were replaced; this also applies to the satellite processor if hardware independence can be achieved there also. Eventual replacement of at least a portion of the hardware is almost a certainty, given the rapid rate of new developments in computer technology. A sufficiently flexible, hardware independent system guarantees that technological advances will not make the system prematurely obsolete.

Service routines

The service routines constitute the library of list-processing subroutines which allow the user to build and manipulate his data structure in the central computer. The data structure is accessible only through these routines. Since the system is implemented in a time-shared environment, it is desirable that these routines be re-entrant, so that a number of application programs may "simultaneously" make use of them.

A number of list-processing schemes have been proposed for computer graphics.⁷ The subroutines provided in the system are naturally determined by the requirements of the particular data structure selected. A representative set of routines for the data structure to be described are listed in the appendix, with a summary of their function.

Communications

Telecommunications routines are of two types: physical and logical. "Physical" routines are required to mediate the transmission of all required messages between the central and satellite computers. They are assumed to be provided by the executive system which permits the satellite processor to remotely access the central processor. "Logical" routines provide the user with a means of sending and receiving information to

and from each of the processors, possibly in an encoded or parametrized or metalinguistic form. Clearly, such routines must be provided in both computers.

An interactive remote graphics system faces three major problems with respect to communications: 1) the time required to transmit data could be excessively long, 2) the overhead load on the central computer would be high if it is required to service all interrupts, and 3) as a result of the above, the reaction time to operator actions at the display would be very slow. Experience with graphic systems indicates that some feedback to operator actions should be provided almost immediately (such as intensifying a light-pen detected image), with the effects of his actions apparent within seconds, at most.

It is assumed that because of line-rental costs, transmission is restricted to a single linkage such as a 210-B modem, which would make available, at most, a transmission rate of 2400 bits per second.

Obviously, the transmission time for lines and points cannot be improved. However, the transmission times for conics (circles, ellipses, parabolas, hyperbolas) can be decreased by including in the remote processor sufficient power to generate the straight line approximations to these conics from a parametric representation. To minimize the number of expansion routines in the remote processor, the parametric homogeneous mathematics for conics⁸, has been selected.

An additional reduction can be achieved in the quantity of data which must be sent when pictures are to be displayed with a high degree of shape repetition. The description of a shape can be transmitted only once, with subsequent instances of that shape requiring only a reference to that shape as previously sent. This is one further argument for a hierarchically organized data structure.

Remote monitor

Software in the remote processor includes a routine to store the logical organization of a picture sent to it from the central site. Another routine builds from this data a display file, containing the actual vector and character hardware commands for that picture's display. This routine contains the logic required to expand the parametric representation of conics to a short vector approximation. Sufficient structure is present in the organization of the data and sufficient power is provided in the monitor to permit the identification and manipulation of elements within the picture being displayed. The ability to respond locally to operator actions is a very powerful feature which can be built into remote systems containing a satellite processor. This raises the interesting question of the division of labor:

how much work is the central site processor to do and how much is the remote processor to do?

Interactive graphic systems are characterized by large numbers of short-burst program executions associated with scope operator actions such as pushing a button, tracking a light target, etc. Seconds can elapse between these bursts, each of which might only consist of retaining the identity of a button pushed or of moving a tracking cross. In a time-shared environment, these short bursts can overload the system if the application program has to be retrieved from auxiliary storage; also, since the system operates remote displays at the end of relatively slow transmission lines (as a 201-B modem), response times could be too slow.

An interpretive language has therefore been designed to allow the user to program his own strategy for attention handling and other functions to be performed in the remote computer. Because these programs in some sense reflect the logic of the particular application, they are referred to as Logic Tables. Logic Tables are executed (interpreted) in the remote processor independently of application program execution at the central site. The intent is to allow the application program complete freedom over the method of system controls and to minimize the interactive communication between the resident programs in the satellite computer and the Worker in the central processor. Individual Logic Table commands may be thought of as subroutine calls that respond to attentions (e.g., keystrokes), conditionally transfer control as a function of input from the operator, request light-pen tracking and pointing, perform insertions and deletions to the satellite data structure, and initiate the transmission of internal messages to the Worker Program.

This approach has three important features not yet available in other graphic systems: 1) it allows the application to design its own control program (not forced to accept a rigid scheme), 2) it provides for remote-console real-time responses, but 3) it frees the application program at the central site of the requirement to provide such responses, thus lowering the time-shared overhead. In a sense, the user has the ability to "fine-tune" the system for his own particular requirements.

Data structures

The data structures in the system are of two main types: a data structure for the definition and organization of graphic structures in the central and satellite computers, and a data structure for image presentation and logical attentional handling. These data structures have an intimate relationship, the second being constructed directly from the first. Each data structure, however, may be independently modified.

Entity table elements

To facilitate the organization and manipulation of the graphic representation being defined or manipulated by the application program, a central data structure called the Entity Table has been defined, which serves as the repository for all the data within the system. Associated with the Entity Table is a modularized, hash-coded reference table called the Directory, which provides rapid access to prime structural entries in the Entity Table. A third table serves as an External Directory, correlating user defined alphanumeric names with their internal numeric identification codes in the Directory and Entity Table.

The prime structural element in the Entity Table is the *group*. Composed of graphical entities called *items* and *uses* (instances) of other groups, it provides the basic "subroutining" capability in the data structure. All sub-structures within groups are defined in terms of the group's coordinate system and are collectively referenced by the identification associated with the group.

A group by itself is essentially an unpositioned organizational structure which may not itself be displayed. A group may be used, however, so that the graphical sub-entities in its organization be displayed.

The concept of group use is basically one of taking a group structure, complete except for its orientation relative to the user's drawing coordinates, and applying an affine transformation relative to the coordinate system of the group receiving the use entry (called the parent group). If a link is formed by an iterative chain with the universal or master group, the group use is then defined relative to the drawing coordinates and its position can be determined for display purposes.

This system allows a structure (of groups and uses) to be defined of considerable complexity with no duplicative effort, since the same group definition can be used repeatedly in different contexts. With this ability to define groups in terms of other groups, it is entirely possible to cause a recursive definition. Since no recursive constraints have been provided, such recursion would be infinite, and hence cannot be allowed. A violation would be discovered when the data structure was searched in preparation for transmission to the satellite, and an error code would be returned to the application program.

Thus far we have described a picture-defining structure with nothing that is displayable (actually we have defined a directed graph with no terminals). Item entities are logical combinations of points, lines, and test (called components) which form a picture and which are primitive elements within groups. Items provide the actual graphical information which may be displayed. They respond to freely positionable graphic sub-

routines which are repeatedly "called" when the group of which they are a member is used. Items themselves may not be multiply used, but are defined once within a particular group. A group may be defined, however, with only a single item entry, and be repeatedly used.

Groups are represented in the Entity Table as a header with a pointer to the chain of entities (uses and items) in the group. The group is identified by its unique internal identification code. Pointers are provided to associated data (managerial data) and to a chain of all the uses of this group. The Directory serves as an index to all groups in the Entity Table. A group's internal identification code serves as a virtual pointer to the group header which is resolved through the Directory.

A use entry has a pointer to the next entry within the parent group structure, a pointer to the next use of the same dependent group, a pointer to the component ring for the use (examples of components for uses are external name, positioning vector and transformation matrix), and a pointer to associated managerial data. The Logic Table associated with this use is also identified if one has been defined. The group referenced by the use is linked by its unique internal identification code via the Directory).

Item entries require a pointer to their component ring (components for items include the actual text, vectors and points displayed on the tube face), but naturally do not include any group references.

Components themselves contain a count of the number of parameter words following and a pointer to the next entry in the item component chain. The order of components in the ring is significant (the only case where this is so), since they may contain polystrings of relatively positioned points and vectors.

Figure 3 presents a simple schematic meta-language for describing a structure of groups, uses, and items, and Figure 4 illustrates how a portion of such a structure would be represented internally at the central site.

Attributes

Associated with each entry in the Entity Table are a number of logical and functional attributes which serve as modifiers or descriptors for that entry. Not all attributes are applicable to all entities. The concept of attribute is a general one, and the system could easily be modified to accommodate additional attributes in response to additional user requirements or new hardware capability (such as color, for example). The attributes following are representative, and are taken from the implementation currently in progress.

First of all, an entity may have an external name associated with it for the user's convenience in referencing it. However, the system works with internal identification codes which are unknown to the user, and does not

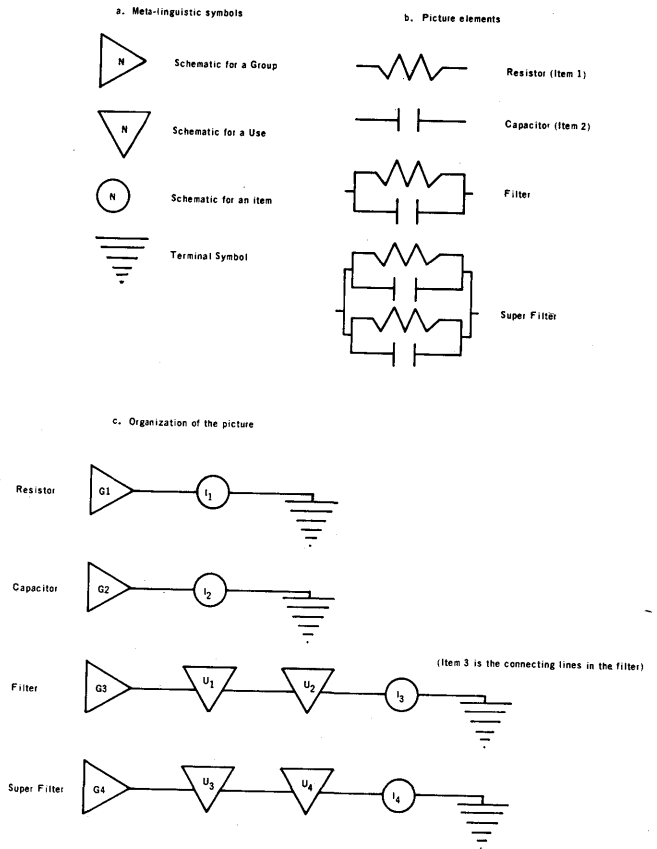


FIGURE 3—Schematic representation of the organization of a picture

require all entities to have external names in order to be referenced; if the graphics terminal has a light-pen, entities may be identified by pointing to them (an example of "the positive power of grunting"). If a light-pen is not available, one may be simulated using the keyboard. An attribute is provided which specifies if the entity is eligible for light-pen detection.

Other attributes similarly refer to the graphic characteristics of the entity. One attribute defines whether or not the entity is displayable, and another defines if it is to be displayed; thus an entity can be temporarily deactivated without being removed from the data structure. Attributes define line type (for example, solid, dashed or end point, intensity (most displays allow at least two settings in addition to off), and thickness (if hardware permits). Color is an obvious extension which can easily be accommodated when such hardware becomes generally available. Attributes can also be defined to specify interest, threshold, and security level. Interest and security levels are self-explanatory; threshold level is simply a function of magnification and the resolution capability of the display device: as magnification increases, more detail can be shown.

With respect to the graphic attributes of displayable

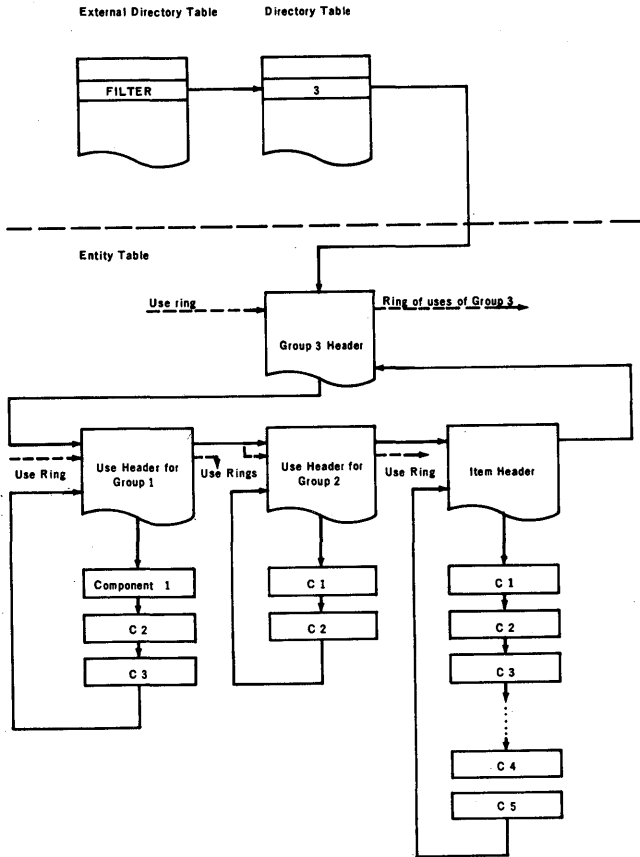


FIGURE 4—Example of a portion of the data structure in the central computer for the representation of Group 3 in Figure 3

items, the hierarchical structure of the picture definition is employed to resolve conflicts which may arise in the data structure. The attributes of a group use override the attributes of the referenced group, which in turn override the group uses within the definition of that group. The rule is that higher level definitions take precedence.

Three functional attributes have special significance to the efficient operation of the remote graphics system. When a picture description is to be sent to the satellite computer, its entire structure must be retrieved, scanned, and transmitted (at one time or another). An attribute is provided to aid in the efficiency of this scan. Whenever the scan of a group is completed, it is marked as having been searched. If this entity is encountered again in the scan of the whole structure (such as a group used repeatedly), it need not be scanned again, but may be referenced by its internal ID. Similarly, when an entity is transmitted to the remote computer, it is marked as having been processed. If the need later arises to transmit such an entity to the remote, only its identification code need be sent, since a copy of it is known to exist in the data base of the satellite.

The third attribute of special significance arises from

the ability within the system to specify the display of only a segment of the picture which has been defined, and to later dynamically alter the extent of that segment.

The extent of the coordinate system within which pictures may be defined is limited only by the magnitude of the largest floating point number which the central computer can represent. Pictures defined in this universe will be said to be in "user coordinates." A segment of this universe is selected for display by the definition of a drawing mask, which identifies a rectangular section of the universe. The picture is scissored to lie within this section, and appropriately scaled for display so as to map onto a selected display window on the face of the CRT. The size of this coordinate system is limited by the CRT raster itself. Figure 5 illustrates this concept. There may be a number of masks simultaneously mapped onto associated display windows and simultaneously displayed. Each window's content and display is independent, however. Both masks and windows may overlap.

Specifying a segment or region of the user defined picture to be displayed on the scope involves scissoring pic-

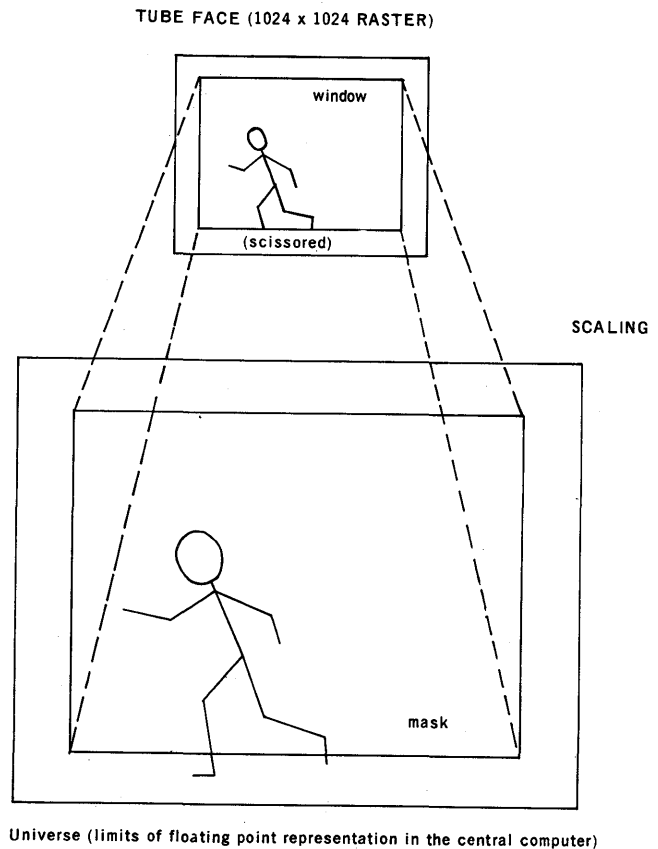


FIGURE 5—Defining and displaying an image. The Universe constitutes the domain of definition for pictures. The mask selects a segment to be mapped onto a particular window for display

ture elements which lie partially or entirely outside of the mask. For reasons of transmission efficiency and storage economy at the satellite, scissored data are eliminated from the picture's description wherever possible, thus altering its description. The description which is sent is marked as having been scissored, so that programs in the satellite may be aware that the picture's description is incomplete. If a complete description is later required for one reason or another, it may be necessary to refer back to the central computer to obtain it.

The problem of scissoring assumes somewhat greater proportions when there are a number of windows displayed on the tube face of less than full screen size, and when interactive work by the user is likely to result in the movement of pictures on the tube face. Hardware assistance can usually be obtained for the problem of full screen scissoring,³ but this is not generally the case when arbitrary boundaries are to be established beyond which images will not be permitted to extend. It becomes the responsibility of the software to provide this service, within the framework of the graphics system and the data structure provided. This service must be provided whenever a call is given to display an image.

Initiating a display

A call to initiate or alter a display sets off a complex chain of events. Starting with the master group for each window, the Entity Table must be searched and all the appropriate graphic elements extracted. This requires essentially a canonical scan of all the branches of the structure. The value of the "Processed" attribute is evident here. When an entity marked as processed and "not scissored" is encountered later in the scan, only its identification need be recorded.

The next step, actually performed when the entity is being extracted, is to convert all the floating point values in its definition to fixed point, with scissoring performed according to the drawing mask and with appropriate scaling according to the relationship of the drawing mask to the display window. Entities which are scissored are marked. Entities lying entirely outside of the mask are discarded, and their parent entity marked as scissored.

The resulting structure, in fixed point form, is then transmitted to the remote processor, where a display file is constructed and the image displayed. The organizational structure is retained in the remote processor for the purposes of relating this structure to attention handling. If a drum or disc is available to the remote processor, this *structure file* may be stored there, and retrieved as required.

Satellite data structure

The data structures in the satellite processor are of three types. First, there is a description of the picture to be displayed in each window in terms of groups, group uses and items. This is just the structure file, or picture organization which is sent from the central site, and does not contain anything which is interpreted by the display presentation hardware.

The second type of structure contains the actual vector and character commands to be displayed. Since the data to be displayed must be associated with an item use (actually a use of the group in which the item resides), the entities which contain the display commands are called item blocks. These blocks are essentially freely positionable graphic subroutines, and are part of the display file.

The ordering, positioning for display, and determination of eligibility for light pen-detection of the item blocks is controlled by the third type of structure. Called the ordered display list, it is constructed by extracting the group and item uses implied by the hierarchical organization of the picture.

Group occurrences within the ordered display list are accomplished through control blocks which contain a jump instruction (interpreted by the display hardware) to lower structures (other groups or item control blocks), and then a jump to the next structure in the parent group. These jumps are not return jumps (subroutine calls), since the display list is explicit for all groups and uses. However, lower level structures contain jumps back to their parent structures, so that the display decoding hardware executes a complete pass through the picture organization on each refresh cycle. Control blocks also contain a command word for the control of the light-pen and trap interrupts, and a chain pointer to the ring of all repetitive occurrences of the entry. This pointer is located in the block so that it is not seen by the display decoding hardware.

Item control blocks contain positioning vectors and return jumps to item blocks. In addition they contain a control word for light pen and trap interrupts, and a chain pointer to the ring of all instances of this item. The jump command exiting from this block may be to another item or group control block or back to a parent group control block.

Item blocks themselves are the graphic subroutines containing the actual vector and character instructions which are displayed.

The descriptive structure is related to the display structure through the rings of groups and uses. Figure 6 gives an example of the organization of this data structure for the same graphic structures used previously in Figure 4.

Logic tables

Logic Tables provide a novel approach to the problem of defining and accomplishing the processing to be performed in the satellite computer. Most remote systems are limited in their flexibility since it is difficult, if not impossible, for the ordinary user to specify the program in the satellite computer. Besides the function of basic attention handling, it is often desirable for the satellite to be able to analyze and change both its data base and display file, and to be able to send and request information to and from the central computer as required.

Typical graphic applications vary widely in the type of response required of the satellite processor. For example, one application might be to display intermediate results of a large data reduction process, and interactively direct the course of the analysis¹⁰ while another common application is to provide the console user with a sketching facility.¹¹ In the first case, the satellite would be required to transmit key depression information back to the central computer to direct the calling of different subroutines, while the second case requires that the satellite be programmed to analyze the actions of the console operator, define new structures, alter existing structures, and perhaps notify the central computer of such additions and deletions.

The Logic Tables are a hardware independent, interpretively executed, interactively oriented language in which processing to be performed by the satellite can be easily programmed by the user. The appli-

cations program initially specifies the Logic Tables as strings of mnemonic commands, and associates them with use or item structures in the data base. Not all structures must have Logic Tables associated with them, except that the master or universal group must have a Logic Table associated, since it is always the first Logic Table interpreted. During execution, Logic Tables are translated and transferred to the satellite with their associated structure. The Logic Table interpreter portion of the satellite resident monitor will execute one pass through the master Logic Table whenever one of the following occurs: the light-pen picks a display target, the tracking symbol is moved, a software timer expires, or a key is depressed on the alphanumeric or function keyboard. Based on its analysis of the action to which it responds, the master Logic Table may initiate execution of sub-Logic Tables; such nesting may continue arbitrarily deep. Logic Table commands include the following:

- 1) Arithmetic and logical capability using software registers for the retention of results.
- 2) Ability to transfer control conditionally within a Logic Table and to other Logic Tables.
- 3) Ability to decode keystrokes and to react to timer expirations.
- 4) Control of light pen tracking and picking identification and feedback.
- 5) Definitional capability for points, lines, text and conics.
- 6) Facilities for obtaining the complete genealogy

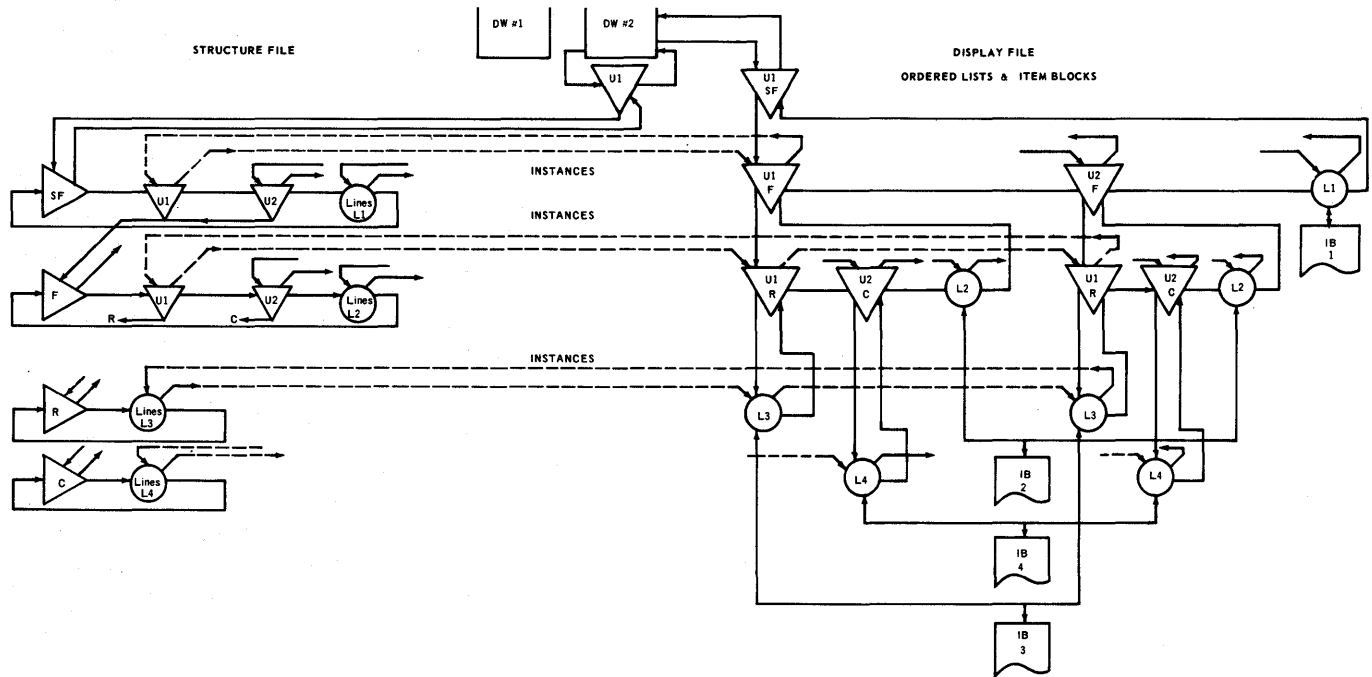


FIGURE 6—Example of the data structure in the satellite computer for the same picture used in Figures 3 and 4

(hierarchical identification) of any entity in the data structure, as well as the identification of its type.

- 7) Ability to search the data structure either horizontally (within a group) or vertically (from parent to dependent entity or vice versa).
- 8) Ability to define new entities and alter existing entities, including the ability to reposition entities and to turn them on and off.
- 9) Ability to record data into report blocks for transmission to the central computer, and to initiate this transmission.

Some examples of the interaction of the Logic Tables with the system's data structures, both in the satellite and in the central computer, follow.

Light-pen identification

In the past, light-pen picking was limited to a one-to-one relationship of the item block hit to a single level of identification. While simple in approach, this did not always satisfy the needs of the graphic system. The data structure design allows the Logic Tables to identify for the scope user either an item, particular use of a group, or all uses of a group.

When the light produced by an item block which is enabled for pen detection is sensed by the light-pen, an interrupt is triggered by the hardware, causing the resident monitor to activate the identification sequence. This sequence may vary depending on the processor in which the system is implemented. For the processor in the present implementation, only the address in core of the display command which generated the light which was sensed is available. Starting with this address, the item block is scanned to find its end, where may be found a pointer back to its head. Just as subroutines may be traced back to back to a calling program by return addresses, so may item blocks and group uses be traced back through this parent structure to the master group.

A Logic Table command is provided to make this genealogy available by entering it on a push-down list which may be manipulated by other Logic Table commands. The push-down is constructed so that dependent structures are on the top of the stack, and parent structures deeper in the push-down. Push and Pop commands are provided for searching the data structure in the vertical sense. A Ring command is also provided for traversing the data structure in the horizontal sense. Feedback (blinking) is also under the control of the Logic Table, which may select a particular instance or all instances of entities at any level in the structure for feedback, depending on the contents of the top entry in the push-down list at the time of the request for feedback.

Light-pen tracking

Light-pen tracking presents a somewhat different problem. While the display is ordinarily refreshed at a rate somewhere between 30 and 60 times per second, this is too slow to detect all but the slowest movements of the light-pen across the tube face. Provision must therefore be made to display the tracking symbol more often—about 200 times per second has proved to be satisfactory.

This is accomplished by the cooperative utilization of trap commands in the display file and logical interrupts occurring every five milliseconds from a real-time clock in the satellite processor. Clock pulses stimulate immediate display of the tracking symbol if display file presentation is not currently in progress. If display file presentation is in progress, display of the tracking symbol is delayed until the next trap command, in order that the display file be resumed at the proper location. Traditional tracking methods¹² can thus be applied with no detriment to the picture being displayed.

Logic Table commands are not required to insert the trap commands, to monitor the real-time clock, to perform the actual tracking calculations, nor to reposition the tracking symbol. The logical effects of pen tracking, however, are all under the control of the Logic Table. An approach is taken which seeks to provide sufficient tracking information without overloading the system. The approach is to require the Logic Table to establish a range around the tracking symbol prior to enabling tracking. The Logic Table will be executed as a result of tracking only when the tracking symbol is moved outside this range. Thus, the user may obtain either very fine or very coarse tracking, depending on his particular requirements.

When a Logic Table is invoked as a result of tracking, the new X and Y position of the tracking symbol is made available. The Logic Table may do with these as it wishes. The rubberbanding of lines and the inclusion of vertical or horizontal constraints are particularly easy with other Logic Table commands which allow lines to be defined on the basis of data in software registers. Thus, Logic Tables represent a very powerful tool for the construction of sketching facilities.

Local modification of structures

Since the Logic Tables provide the ability to modify the data structure in the satellite computer both by altering existing structures and by defining new entities, the satellite's data file may be used as a scratch-pad for changes to be made to the central site data file. All changes made by the Logic Tables are strictly local to

the satellite, and have no effect on any structures in the central site. In fact, the communications link could be disconnected between central site and satellite during the period in which the console operator wished to try out his modifications, without degrading the console response. Of course, the link would have to be re-established before these changes could be incorporated into the central site data base. The facilities of the report block would be used to inform the application program of console operator actions or of modification performed by the Logic Tables in response to console operator actions. The application program would then instruct that changes be made to the Entity Table paralleling the changes made in the satellite.

Report block

Report blocks provide for the transmission of information about the console user's actions from the satellite to the central computer. Report blocks are built in response to Logic Table commands, and only transmitted to the application program at the direction of the Logic Table. Report blocks allow accumulation of data to form a logical, comprehensive set of parameters for application program execution. By deferring application program activation until a complete message is ready, this approach should keep conversational overhead to a minimum.

Actually, report block information is not simply allowed to accumulate in the satellite until the report block is terminated and sent. Since core is at a premium in the satellite, and in order to decrease the amount of transmission which must actually be performed when the completed report block is requested to be sent, a report block buffer is provided in the satellite which is flushed and transmitted to the central site whenever it fills. A complete report block, however, implies a report block header and terminator, indicating a complete logical message. While the report block may be transmitted in fragmented physical segments as it accumulates, it is not made available to the application program until a complete logical report block is built and sent by the Logic Table.

The following types of information may be recorded into a report block by the Logic Tables: constant values, software register contents, push-down list contents (genealogy of an entity), coordinates of a particular point, and new use and item entries. This information becomes available to the application program in a general parametrized form which the worker can employ to reconstruct the actions of the console user which the Logic Tables were programmed to record. Based on this information, the application program may alter the data structure in the central computer to reflect the current status of the image on the tube face.

Scissoring

As has been described, all picture elements are defined within a user universe defined in floating point user coordinates. Scissoring first became necessary when a segment of this universe was selected to be displayed on the tube face. When the call to display an image was given, the data structure was searched to determine what lays within the defined limits. Items which lay completely outside the limits were discarded, but their parent structures marked as scissored. Items which lay partially out were converted to a scissored representation. This may have involved discarding certain components and altering others (for example, shortening a line). In this case the parent structure was also marked as scissored. The scissored picture was converted to fixed point representation so as to map onto the display window, and transmitted to the satellite for display.

Scissoring (or unscissoring) may be required at the satellite processor when entities are moved under the direction of the Logic Tables. These are two possible cases: entities may be moved outside the defined limits so as to require scissoring, or scissored entities may be moved so as to lie more with the limits. The first case the satellite processor can handle by itself it has an unscissored version with which to work. Sufficient power is built into the resident monitor to perform the required scissoring of the entity as it is moved outside the limits. The second case will require the assistance of the central computer. Recall that entities which lay entirely outside the limits were not transmitted to the satellite. The satellite is not aware of even which these entities are; all it knows is that the parent structure was scissored. When the parent structure is moved so that more of it may be displayed, a request must be made back to the central computer to provide the additional data required to display structures coming into view.

Since the central computer is aware of which entities are stored in the satellite's data base, it may be able to save on transmission by only sending identification codes for entities which it knows are available at the satellite, and permitting the satellite to perform the appropriate scissoring. In the case of partially scissored entities, it may be that the satellite has an unscissored version in its data base. If it can be determined that only this entity is required to be moved (Logic Table commands provide the ability to reposition particular entities), then the satellite can use the unscissored version without the necessity of a request back to the central computer.

Logic table extensions

Once the basic interpretation cycle for Logic Tables

has been implemented new Logic Table commands may be added (by a systems programmer) without great difficulty. Since the full power of the satellite computer is available, rather elaborate Logic Table commands may be designed to satisfy particular user requirements. For example, a problem in the effective use of the light-pen has been the inability to determine the exact position of the pen when a detect is recorded on a line generated by a single vector instruction. A Logic Table command has been designed which can provide this information by automatically displaying the locus of addressable points on the line in a search for the pen.

Another example of the imaginative use of Logic Table capabilities are in the display of labeled *construction points*. These are points displayed under Logic Table control which are not part of the data structure. Other Logic Table commands may associate such points with the parameters of particular graphic structures, such as conics, for their automatic definition and inclusion into the data structure.

Finally, Logic Table commands may be designed which make advantageous use of the computing power and information available at the central site. For example, in mapping user coordinates onto raster coordinates, a certain degree of resolution is usually lost. Re-converting raster coordinates back to user coordinates then results in values differing from the original. Facilities can be included in the graphics system to insure that reconversion results in values within the original locus. Such facilities may also be employed to insure that a newly defined point which lies visually on a structure lies actually on it.

CONCLUSION

In summary, the system described is a synthesis of the already popular methods for hierarchical picture definition and the technique of interpretive specification of programs to be performed in a small satellite processor. This approach results in a system with a higher degree of flexibility than that found in most graphics systems. The system is designed to satisfy the requirements of dynamically defining and altering graphical images, within a remotely accessed environment. It is an interactively oriented system, such as would be required for design automation. It is only fair to point out that providing this capability occasionally results in additional overhead beyond that which might be required for systems oriented towards output only. Certainly there must be an output-oriented system embedded in the interactive system, but this system may at times be less efficient than it might need to be, were display its only function. In a sense, the display of an

image is an afterthought; the system is primarily concerned with the description organization of the picture. As has been pointed out, this approach lends itself well to a hardware independent interface.

APPENDIX

A representative set of service routines for the central site are outlined. Only routines required by the system are included. Other routines could be included in a working system for application program convenience or economy.

- WINDOW—define a CRT window and associated mask delimiting the display of a given data structure.
- GROUP —create a group-header entry in the data structure.
- USE —place a use-header for a specified group into a specified parent group.
- ITEM —place an item-header entry into a specified group.
- COMPE —add components to the specified item or use header.
- DELETE —delete the specified entity from the data structure.
- DISPLAY —display any specified graphic structure contained in the Entity Table, subject to attribute logic plus mask and window limitations.
- ASSOC —enters arrays of managerial information into the Entity Table.
- GET —retrieves any specified data from the Entity Table.
- LOGIC —performs Logic Table conversion and enters the Logic Table into the Entity Table associated with a specified entity.
- WAIT —suspends worker program activity until a report block is sent from the remote.
- REPORT —retrieves report blocks entries.

ACKNOWLEDGMENT

The authors wish to acknowledge R. Ladson, N. Fritchie, and G. Halliday of UNIVAC, who were responsible for initiating and directing the work described, and D. Cohen and R. Baust of Adams Associates who made significant contributions to the system design.

The design of the Graphics System was developed by Adams Associates, under contract to UNIVAC. The implementation currently in progress employs UNIVAC 1557 Display Controllers and 1558 Display Consoles remotely accessing UNIVAC 1108 computers.

REFERENCES

- 1 D T ROSS et al
*The design and programming of a display interface system
integrating multi-access and satellite computers*
ACM/SHARE 4th Annual Design Automation Workshop
Los Angeles California June 26-28 1967
- 2 C CHRISTENSEN E N PINSON
Multi-function graphics for a large computer system
Proc of the FJCC Vol 31 Thompson Book Co Wash DC 1967
pp 697-711
- 3 W H NINKE
*A satellite display console system for a multi-access central
computer*
To be presented at 1968 IFIP Congress
- 4 W R SUTHERLAND
The CORAL language and data structure
PhD Dissertation MIT Cambridge Mass 1966
- 5 A VAN DAM D EVANS
*A compact data structure for storing retrieving and manipulating
line drawings*
Proc of the SJCC Vol 30 Thompson Book Co Wash DC 1967
pp 601-610
- 6 M H LEWIN
An introduction to computer graphic terminals
Proc IEEE Vol 55 No 9 Sept 1967 pp 1544-1552
- 7 J C GRAY
Compound data structure for computer aided design
Procedures of the ACM National Conference 1967 pp 355-365
- 8 L G ROBERTS
*Homogeneous matrix representation and manipulation of
n-dimensional constructs*
Notes for the Engineering Summer Conf Univ of Michigan
Ann Arbor June 14-18 1963
- 9 A VAN DAM
*Computer driven displays and their use in man/machine
interaction*
Advances in Computers Academic Press New York 1966
- 10 BOWMAN & LICKHALTER
Graphical data management in a time-shared environment
Proc of the SJCC Vol 30 Thompson Book Co Wash DC 1967
pp 353-362
- 11 I E SUTHERLAND
Sketchpad a man-machine graphical communication system
Proc of the SJCC Vol 32 Thompson Book Co Wash DC 1968
pp 329-346
- 12 ADAMS ASSOCIATES
Computer Display Review
Bedford Mass

Graphical systems communication: An associative memory approach *

by EDGAR H. SIBLEY, ROBERT W. TAYLOR**
and DAVID G. GORDON

University of Michigan
Ann Arbor, Michigan

INTRODUCTION

This paper describes the design and preliminary implementation of a graphics system using a large and small computer configuration connected through conventional voice lines. The small computer is driven by both graphical and conventional input/output devices. In this type of system it is extremely important that the communication line carry only messages with high information content, since low speed is one of the major system constraints. Thus, the design of the software system depends heavily on the communication support devices, as well as on the relative power of the computers.

Before continuing, it is obviously necessary to define the meaning of some relative terms. Thus, *large computer* implies, in this article, any one of many high-speed, large-memory, timeshared computers, but more specifically, an IBM 360/67 working under a general timesharing system developed at the University of Michigan. The graphic system is just another user (i.e., it has no privileged status). The *communication link* consists of conventional telephone lines using standard transmission equipment; this means that the transfer rate is in the audible range, which is abysmally slow for intercomputer communication. For those unfamiliar with these transmission rates, a picture of average TV quality would normally require about five to ten

minutes of transmission time. Specifically, we are using 201 modems (modulator-demodulators) which have a transmission rate of two-thousand baud (or audible bits per second); normal telegraph rate is, of course, substantially lower than this, and uses different modems. By *small computer* we imply one with a relatively small memory, but coupled to a CRT input/output device, with low-speed extension to the memory by use of secondary storage. Specifically, we are using a Digital Equipment Corporation PDP-338 system. Although this represents the present configuration, further work is under way on the use of other small computers, either with greater or lesser capability, in an attempt to calibrate the potential problems of more or less sophisticated graphic terminals.

Because the major power resides in the large computer, the small computer may be looked on as either the slave of the main computer, or as a device which supports the graphical input/output equipment. In either case, we must assume that the large computer stores most of the information, and presumably makes most of the computations, whereas the small computer acts as a combined message switcher and data compression device. Thus, the major use of this experiment will be the development of software programs in the small computer which simulate future potentially hardware macros, while at the same time allowing research on data structures in the large computer, the development of graphical languages, and the ascribing of meaning to these graphics.

The rest of this article may therefore be logically divided into seven parts:

*The work reported in this paper was supported in part by the Concomp Project, a University of Michigan Research Program, sponsored by the Advanced Research Projects Agency, Department of Defense, under Contract Number DA-49-083 OSA-3050.

**Mr. Taylor received support from the National Aeronautics and Space Administration under their Traineeship Program.

A brief description of the hardware features of the system and its components, to bring out the advantages and disadvantages of the particular configuration, rather than to provide an extract of manufacturers' literature.

Examples of the communication language, and the software macros required for its implementation in the small computer.

A brief explanation of the software-implemented data structure of the large computer and the language used to access it.

A description of the communication generator language.

An example of the abstraction of meaning from graphical data, using the associative and relational structure.

Simple examples of the system in operation.

A summary of present and future work.

System hardware considerations

This section is a brief description of system hardware from the programmer's standpoint. Thus, we shall not discuss any particular hardware features unless it affects the design of system software.

Although much could be said about the IBM 360/67 computer, its reference literature speaks for itself. It is obviously a highly sophisticated, large-memory computer with many special hardware features, specifically designed to encourage timeshared operation. Its present software support does not include IBM's timesharing system (TSS) because of many delays, but consists of the Michigan Terminal System (MTS)¹ developed as an interim measure for both education of system programmers as well as systems support for early timesharing users. The major problems of the present timesharing system are ones of optimization, where the multitude of users and proliferation of disc files have tended to cause slow operation, due to build-up of input/output queues at the disc support routines. Thus to a day-time user the system is somewhat slow, but, as proven at other installations, it really flies at night. The constraint of the large computer is mainly one of access, the graphics terminal being a non-privileged user of a multi-user, large timesharing system.

The communication link may be either through 201 or 103 dataphones over telephone lines. The message switcher between the IBM 360 and the modems is either a standard IBM 2702 interface, or the higher speed "Data Concentrator," which is

a modified PDP-8 developed at the University of Michigan.² The Data Concentrator treats all users (including the IBM 360) in similar fashion, routing messages to the designated devices and users; most of the routines written for it may be used by the PDP-8 part of the 338 configuration for device support. Thus, the small computer operates as a message switcher between 201 lines, the teletype, card-reader, tape-reader, disc, Grafacon, light pen, and display. Any input to the PDP-8 causes job scheduling, followed by calls to special device support routines. Thus the software task in the PDP-8 becomes one of writing support routines, graphics language interpreting routines, and macros to communicate between the various parts.

The DEC 338 operates in cycle-stealing mode, with the display counter running through a display file in its PDP-8 core. The PDP-8 program sets up or modifies the display file, which the display controller then executes. The special hardware feature of the 338 is therefore a display register which steps sequentially through the display file, picking up point or vector commands; the point command gives an absolute position of X Y coordinates for locating the beam, whereas vector is always incremental from the previous position. Although it is available, our configuration does not include a character generator.

In order to allow self-refreshing of the display file, the display register can be loaded with a new location by using a jump command; the simplest display file could therefore be a series of position and vector commands followed by a single jump to the beginning of the display file. A further jump, called a push jump, may be used in conjunction with a "pop" instruction to allow sub-routining.

A push jump stores the present location of the display file in a pushdown stack, and jumps to the given location. A series of instructions at this location may contain a particular graphic grouping, which may be used several times in the same picture without redrawing, provided that all its instructions are relative rather than absolute. As an example, the character strings are normally implemented by a push jump to the incremental vector characters each of which ends with a pop, therefore requiring only two instructions per character rather than several.

Unfortunately from the software programmer's standpoint, the 338 controller has only primitive logical capabilities. The ability to make a logical

comparison of locations in the display file from the display file and to add or subtract a number (even 1) from within the display file would be a great advantage over the present configuration.

The light-pen capabilities involve enabling the light pen so that it will send back an interrupt whenever it sees light. This allows the PDP-8 to examine the status of the display file, and read the display register or X Y coordinates to determine the message to be sent to the 360.

The actual configuration and major system specifications are given in Figure 1.

The graphic macros

The DEC 338 used as a terminal in this experiment interprets the graphic macros sent from the 360 and sends messages concerning user actions to the large computer. It has 12K of core (three core banks of 4K). A Type DF32 Random Access Disc File provides another 32K of back-up storage, approximately half of which is available for scratch use by the system. We currently use one core bank for our program; display files occupy the second core bank; the third is currently about half full of tables; and the rest is dynamically allocated for buffer and display file use.

The display files make extensive use of the subroutine jump (called a push jump) and the pop features of the 338 display to give a ring struc-

ture to the display program. Specifically, the highest logical level in the display program is a list of push jumps terminated by a normal jump back to the beginning of the list. Each push jump causes execution of a section of code which is a graphic description of the entity involved. This section may, of course, contain other push jumps. Each of these drawing sections is terminated by a pop; the ring is thereby closed.

Each entity in a display file is given an internal name. This is accomplished by associating the display file location of the highest-level push jump with the item drawn. It is thus possible to identify a specific item for further processing by examining the *bottom* of the pushdown stack when a light-pen interrupt occurs, i.e., the first return address placed in the pushdown stack will yield for each graphic entity a unique number. It might also be noted that later entries in the pushdown stack correspond to sub-parts of the graphic entity identified by the first entry in the stack. These later entries can be helpful in resolving ambiguities in some situations.

The set of graphic macros used in the construction and manipulation of display files can be divided into several parts. These are the basic drawing commands, the display file editing commands, and the figure commands.

The five basic commands used in the construction of graphic entities are: POSITION (PN), POINT (PT), LINE (LE), NON-INTENSIFIED LINE (NL), and CHARACTER STRING (CH). While it is clear that this set of primitive drawing commands is by no means exhaustive—circular arc is an obvious addition—it was felt that they formed a useful subset and would suffice in our early investigations.

POSITION, POINT, LINE, and NON-INTENSIFIED LINE all take two decimal numbers, in the range 0 to 1023, as arguments. (There are 1024 points in the basic 338 display, but this is an easily adjustable parameter at the command language level.) CHARACTER STRING takes a variable number of characters from the ASCII set, with the exception of ' (apostrophe), which is used as a delimiter. POSITION and POINT are similar commands; both position the beam to the coordinates X,Y given in the arguments. POINT intensifies the beam once positioning is accomplished. Clearly the only difference between LINE and NON-INTENSIFIED LINE is the beam status; both draw a line from the current beam position to the X,Y coordinates given as an argument in the command. CHARACTER STRING takes

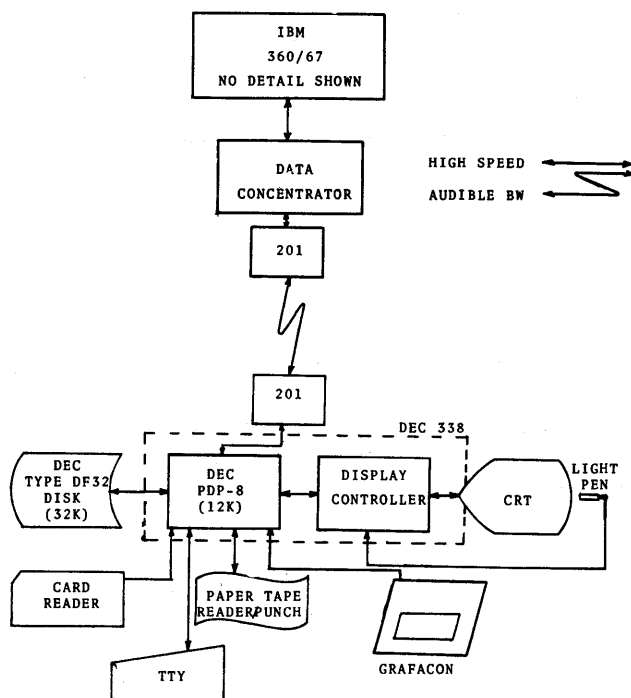


FIGURE 1—Configuration and major system specifications

the string which follows and displays it at the current beam position.

One of the design decisions that even this limited subset of commands raises is: Should the commands refer to absolute locations on the screen at all times, or should there be some facility for relative or incremental coordinates? An example of a relative positioning command would be "RP 10, - 3" which would position the beam to a point 10 raster units to the right and three raster units below the current beam position. Relative commands for lines could be similarly designed. Although, with a little computation, we have some facility for relative behavior,* it is clear that we took the more absolute approach. A complete set of primitives would probably have facilities for both types of commands.

Editing of drawings can be carried out by using the commands: DELETE (DL), BLANK (BL), and UNBANK (UB). Each of these commands takes a display file location as an argument, and either permanently deletes the item from the 338 display file, temporarily blanks the item without removing it, or unbanks a previously blanked item. A sketch may also be repositioned using the command MOVE (MV), which will translate the named item to a new position on the screen. There is currently no provision for rotation, though this would be an obvious extension of the command set. In addition, the command MODIFY (MY) allows a displayed entity (except for a figure, see below) to be changed.

Two other commands are provided which allow one to manipulate drawings. These are REPLOT (RP) and COPY (CP). The first of these plots a previously displayed entity at the current beam position. Internally, this means that another subroutine jump (push jump) to a previously existing section of code is generated and another so-called instance will appear. Instances thus have the property that if one changes, all change. COPY, on the other hand, generates a copy of the previously existing code in addition to creating a subroutine jump to this copy of the code. Thus a copy can be changed without modifying the entity from which it originally came. Both facilities were included in our command set so that we might gain some insight about the situa-

tions when each method of creating new entities is useful.

The next step is to allow a user to build up a library of drawings, each with a name, which he can call up for use in other drawings. To this end, we have defined the commands DEFINE FIGURE (DF), END FIGURE (EF), PLOT FIGURE (PF), MODIFY FIGURE (MF), and KILL FIGURE (KF).

DEFINE FIGURE places the system in a special mode whereby all succeeding actions with the light-pen are made relative to a starting point, and are placed in a special block until the END FIGURE command is given. At this time, the block is transferred to the disc storage at the 338 under the BCD name given by the user. When a user wishes to include a previously defined figure in a new drawing, he issues a PLOT FIGURE command, which plots the named figure at the current beam position. Because of the subroutine feature in the 338, only one plot of a figure need be kept in core, no matter how many times the figure is used.

Figures, like any displayed entity, are subject to the editing commands DELETE, BLANK, and UNBLANK, as well as to MOVE. To remove or change the definition of a figure, however, the commands KILL FIGURE and MODIFY FIGURE are provided. MODIFY FIGURE brings a previously defined figure into core, displays it, and places the system in DEFINE FIGURE mode so that DELETE, MOVE, etc., will permanently change the definition. An END FIGURE command will, of course, initiate the storage process once again. KILL FIGURE removes the definition of a figure from the appropriate tables and disc files.

The principal means by which a user communicates the above commands to the system is through use of the light-pen. Thus the two final commands of this admittedly incomplete set are POINTING MODE (PM), which places the light-pen in a mode whereby the name of an item pointed at will be sent to the 360, and LIGHT PEN TRACK MODE (LT), which causes a tracking symbol to appear and transmits the X,Y coordinates at which the user finally loses tracking to the 360.

The instruction set also contains several other commands which will not be described here because of their detailed nature. Complete documentation of the graphic macros may be found elsewhere.³

A brief description of the communication proc-

*Relative position is simply a non-intensified line from the current position to the desired position; the distance must be computed before the command is given.

ess from the viewpoint of the 338 can now be given (see Section V for a more complete treatment). When a user initiates program operation, a list of commands which he can issue will appear on the edge of the screen (Figure 2). These commands, the menu, are set up by the system at initialization, and each one has associated with it a name which is the display file location of the highest-level push jump. Thus, when a user points at an item in the menu, the display file location is transmitted to the 360 and the 338 awaits further instructions. Two options are open to the 360 at this point: it may either enable the 338 in POINTING MODE, in which case the name of the next entity pointed at will be returned, or it may enable the 338 in LIGHT PEN TRACKING MODE, in which case the X,Y coordinates of the point at which tracking is lost will be returned. The 360 can then use these two pieces of information to generate one of the display commands described above, or it can wait for more information to be provided, either from the light-pen or, perhaps, from the teletype in the case of characters. When the 360 has completed its command generation it re-enables the 338 in POINTING MODE and waits for the user to point at another item in the menu. The process thus repeats.

The present process reflects little sophistication on the part of the 338. Adding further capabilities at the terminal will clearly improve system performance, which even now is not intolerable. We plan to build this sophistication into several different terminals thereby providing one practical indication of the computing power necessary (at the terminal end) to achieve a given

level of system performance. For example, the use of auxiliary tables in the terminal would enable the 338 to act immediately on certain standard commands like LINE rather than having to wait for a message similar to the one just sent to return via the 201 line. This will be our first upgrading of the terminal's capabilities and should cut down the traffic on the 201 significantly. However, we feel that ultimate control of terminal characteristics—the state of the light-pen, for example—will still have to reside in the 360, because new items, whose meaning in terms of actions is unknown to the 338, may be added to the menu at any time.

Data structure and its access language

The data structure is fully described elsewhere,⁴ but a short description at this point may be necessary for full understanding of this paper. The acronym TRAMP stands for Timeshared Relational Associative Memory Program, and it refers to two distinct types of subroutines used to store and retrieve data. The first of these is used to enter, retrieve, and manipulate data in an associative (content-addressable) fashion by using a hash-coded scheme first suggested by Feldman.⁵ The second part is a logical compiler, which places an artificial structure on the associative memory: the relational package allows the user to make logical statements about the relations between objects, which may already be stored in the associative data structure. As an example, the associative structure may have stored the fact that "object A" is "to the left of" "object B"; this is an associative triple which may be stored or retrieved in the first data structure. If we now define the fact that the relation "to the left of" is the converse of "to the right of," then questions about right position of objects may be asked without requiring explicit storage of extra information. The second package will search the data structure for converses (as well as many logically defined relationships).

Although the machine-coded functional packages described above were developed for embedding in the UMIST* interpreter, the first package relies on it only for input/output, and therefore could simply be modified for use elsewhere. The particular use of the interpreter will be obvious in later discussion, but at this point it is worth

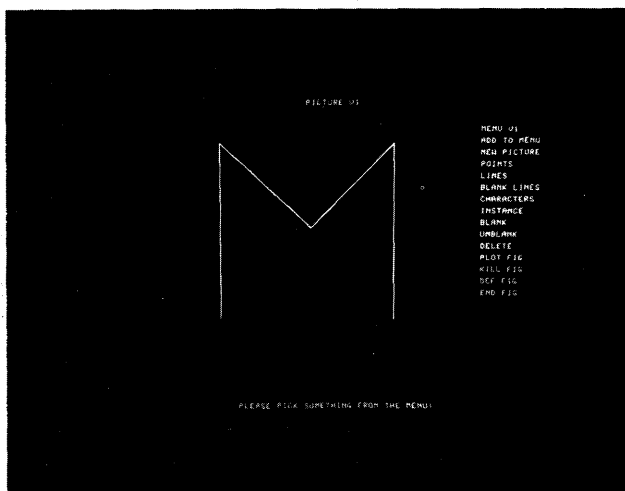


FIGURE 2—The original menu

*UMIST is closely patterned after the TRAC-T-64 language, and was implemented at the University of Michigan with the cooperation of Mr. C. N. Mooers, creator of the TRAC language.

noting that the interpreter is fully recursive and has excellent debugging features which make it possible to write and modify programs rapidly, yet with an efficient set of algorithms.

The initial work of Feldman was a strong motivation for the initial design of the associative memory. His notation will be adopted in this explanation. The generic entity is:

$$A(O) = V$$

<Attribute> of <Object> equals <Value>

Each of the three components is a non-empty set. By appropriately designating the three components as being constant or variable, there are eight "questions" that may be asked of the data structure:

- F0 A (O) = V
- F1 A (O) = x
- F2 A (x) = V
- F3 A (x) = y
- F4 x (O) = V
- F5 x (O) = y
- F6 x (y) = V
- F7 x (y) = z

where [A,O,V] represent constants, and [x,y,z] are variables. Question F7 is not a question at all but a request for a dump of the associative memory. Question F0 simply asks: Does $A(O) = V$? And the answer is a kind of truth value. In the case where A, O, and V are all singletons, the truth value is a straightforward 1 or 0 denoting whether or not the specified association can be verified by the data. The interpretation is slightly ambiguous, however, when one or more of the three sets has cardinality greater than one. To illustrate, assuming that the association ENDS (L1) = P1;P2 has been stored, the following questions have the defined truth values:

- 1) ENDS (L1) = P3 0
- 2) ENDS (L1) = P1 1
- 3) ENDS (L1) = P1;P3 ?

The interpretation which seemed most natural, and the one adopted, gives:

if ALL associations implied by the question are resident in memory, or derivable thereof, the value is 1;

if none, the value is 0;
if some, but not all, the value returned is ?

Questions F1 to F6 simply ask the system to fill in the blank(s), i.e., to replace the variable with the set that is the answer to the question.

For example, Question F1, $A(O) = x$, asks for the set (x) of all values (v) which satisfy the conditions A (O). Thus the question: ENDS (L1) = x produces the result $x = P1;P2$. Question F3, $A(y) = x$, asks for the set (y) of all Os and the set (x) of all Vs that have the attribute A. Thus in our example:

$$y = L1$$

$$x = P1;P2.$$

Of course, for a normal picture there is more than one line, and the normal result of this operation might well be:

$$y = L1;L2;L3 \dots$$

$$x = P1;P2;P3;P4;P1;P3;etc.$$

Because of the recursive nature of TRAC, Questions F1 to F6 may be nested in any way, to any desired depth.

For those totally unfamiliar with the TRAC language, for this paper it is necessary only to know the syntax of a function call. The sharp sign (#) signals the start of a function call, with the call itself enclosed in an immediately following pair of parentheses. The arguments are separated by commas, and the first argument is the name of the function. # (sub,ARG) is therefore analogous to the FORTRAN: CALL SUB(ARG). One of the minor additions of UMIST is to allow implicit calls of functions, i.e., when the normal call might be # (cl, FUNC) in TRAC, the UMIST call may be either the same, or else # (FUNC).

The name of the storage function is *dr* and the syntax of the call is # (dr,A,O,V). None of the three arguments to *dr* may be an empty set. Each point in the cartesian product of the three sets is stored using hash-coding techniques, i.e., each element of each set is grouped with each pair of elements of the other two sets, and the resulting triple is stored. Thus a single call on *dr* stores as many associations as the product of the cardinalities of the three sets.

The primary retrieval function has the name *rl*. The syntax of the function call is identical to

that of *dr* except for variable specification. A variable in TRAMP is denoted by enclosing a name, possibly null, within asterisks (*). Thus, $\#(rl,A,O,V)$ has no variables and asks whether $A(O) = V$; $\#(rl,A,O,*X*)$ asks: What does $A(O)$ equal? Place the answer in X. When the variable is not given, the answer is returned into the calling string.

rl generates the union of the answer sets. That is, the question: $\#(rl,ENDS,L1;L2,**)$ has two answer sets: the ENDS of L1 and the ENDS of L2. *rl* simply forms the *union* of however many sets there might be, however *int* is a function which generates the *intersection* of the several answer sets. Thus, $\#(int,ENDS,L1;L2,**)$ generates the set of all end-points common to L1 and L2. $\#(rl,ENDS,L1;L2,**)$, on the other hand, would generate the set of all points at the ends of either L1 or L2.

Throughout this article the UMIST delimiter of arguments is the *comma*; the element delimiter for TRAMP cannot be the same, we therefore use the *semicolon*.

The communication generator language

The large timeshared computer is used for storage, retrieval, and manipulation of the description of the graphical structure as well as communication of display commands to the smaller display computer. Essentially, the information is stored in a hierarchical structure built up in TRAMP, whereas the communication between the large and small computers is under the command of UMIST, which also resides within the large computer. This intercomputer communication is kept as small as possible because of the low channel capacity of the data lines.

At each of its transmissions, the small computer sends either the coordinates of a tracking-cross position or the reference name of an entry in the small computer's display file. The large computer interprets this message in terms of what it was expecting to receive and sends back the necessary position, line, character string, or command (as described in a previous section).

In addition to sending commands to the small computer, the large computer stores picture information so that it may either alter pictures (upon command), or perform computations on picture structure, or redisplay pictures at a later time.

The quiescent state of the large computer is

“procedure check,” which corresponds to pointing mode in the small computer. In this state, the critical items on the display are the list of operations which can be performed, the menu. Because each item in the menu is a character string at the display tube, it has a unique name, a four-digit number, corresponding to its location in the data file. If the light-pen is pointed at an item, for example, the word POINT, the display file location corresponding to this entity is transmitted over the data line to the large computer, which can determine the procedure corresponding to this name and execute this procedure. This is possible because, at the time that the name item was created, its display file location was associated with the corresponding procedure, i.e., the association was defined by executing: $\#(DR,PROC,\#(XIN),\#(TEST))$, where $\#(XIN)$ is an implicit call on an input string buffer. The *call* is replaced by the last input string, and $\#(TEST)$ is replaced by the value of TEST, in this case the string POINT, which was previously written as the procedure which defines a point. The input string, stored in XIN, is the display file location corresponding to the character string POINT. Let us assume this location is 0027; then we have $\#(DR,PROC,0027,POINT)$.

Now, whenever the light-pen is in pointing mode, and POINT is picked, the name 0027 will be transmitted over the data line (see Figure 3). The larger computer is in the procedure check state, which means it is waiting to read an input string, viz: $\#(\#(RL,PROC,\#(RS),**))$. If POINT is picked, this becomes first $\#(\#(RL,PROC,0027,**))$, then $\#(POINT)$, which then initiates a string of commands to define points.

Figures 3 and 4 show the communication sequence between the machines for several operations. Those lines prefixed with “<<” are messages from the smaller to the larger computer while those with “>>” are from the larger to the smaller. The unmarked lines are either messages from the computers to the user, or typed messages from the user. Ordinarily very little is printed out at the teletype once the menu has been built up.

In addition to PROC(EDURES), there are several other attribute names of paramount importance in the large computer; among these are CLAS(S), COOR(DINATES), characters (CHRS), and HIST(ORY).

For example, to store the description of a

```

>>PM
<<0029
>>LT
<<0422 0842
>>PN 0422 0842
* <<0037
>>LP
<<0584 0710
>>LE 0584 0710
* <<0039
>>LP          Light pen
<<2000        Stop
>>PM          Pointing mode
<<0027        Data file location of POINT
>>LT          Light pen tracking
<<0370 0426   Coordinates
>>PT 0370 0426 Point Command
* <<0041        Data file location
>>LP          Light pen (continue)
<<0530 0432   Coordinates
>>PT 0530 0432 Point command
* <<0043        Data file location
>>LP          Light pen (continue)
<<2000        Stop
>>PM          Pointing mode
<<0035        Data file location of CHARACTERS
>>LT
<<0360 0244
>>PN 0360 0244
* <<0045
CHARACTERS?   Computer message on teletype
NOW WE HAVE ONE LINE, TWO POINTS AND A CHARACTER STRING.
>>CH NOW WE HAVE ONE LINE, TWO POINTS AND A CHARACTER STRING.
* <<0047
>>LP
<<0350 0201
>>PN 0350 0201
* <<0049
CHARACTERS?
THIS OUGHT TO GIVE AN IDEA OF COMMUNICATION
>>CH THIS OUGHT TO GIVE AN IDEA OF COMMUNICATION.
* <<0051
>>LP
<<2000
>>PM

```

* At this point the 360/67 is defining associations CLAS(S), COOR(DINATES), and HIST(ORY).

FIGURE 3—Drawing a picture

```

>>SN          Start again
PICTURE NAME? Computer message on teletype
VI           User's answer on teletype
>>PM          Pointing mode
MENU        User types on teletype
>>PN 850 900 Automatic start position of menu
<<0017        Data file location
* <<0019        Character string
>>CH MENU VI  Data file location
<<0021        Position
>>PN 850 870 Data file location
<<0021        Character string
>>CH ADD TO MENU Character string
* <<0023        Data file location
DONE.        End of automatic part of menu
>>PM          Pointing Mode
<<0023        Light pen picks procedure
ENTRY IN..MENU;PROGRAM? Computer message on teletype
POINT;PINT   User's answer on teletype
>>PN 850 840 Next line of menu
<<0025        Name
>>CH POINT'   Character string
* <<0027        Data file location. (of POINT)
ENTRY IN..MENU;PROGRAM?
LINE;LIN
>>PN 850 810
<<0029
>>CH LINE'
* <<0031
ENTRY IN..MENU ,PROGRAM?
CHARACTERS;CHR
>>PN 850 780
<<0033
>>CH CHARACTERS
* <<0035
ENTRY IN..MENU;PROGRAM?
'
DONE.        User types ' to stop
>>PM

```

Note: Message direction: ">>" - larger to smaller computer
 "<<" - smaller to larger computer

* At this point the 360/67 is defining associations CLAS(S), COOR(DINATES), PROC(EDURE), and HIST(ORY).

FIGURE 4—Defining the menu

point, P1, which is in Picture VI, the following set of TRAMP statements is executed:

```

# (DR,CLAS,P1,VI,)
# (DR,CLAS,POINT,P1)
# DR,COORD,P1,0370:0426)
# (DR,HIST,0041,P1:VI).

```

This involves defining P1 in the class of V1; POINT in the class of P1; the coordinates of P1 as 370/426; and P1's display file entry (41) is then given the history of the dendrite* of P1.

The information can be retrieved by asking questions. For example, in Figure 3, where many pictures, etc. are defined within other pictures: # (RL,CLAS,*T*,VI) will define T to be L1;L2;P1;P2;L3;C3;L4;C4.

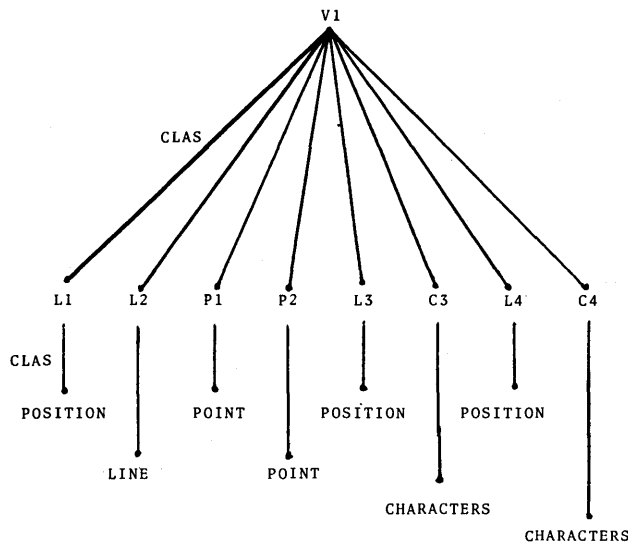
Please note that these entities are ordered, being retrieved in the same order as they were defined. This is important since drawing lines from coordinate pairs L7 to L8 to L9 will not, in general, produce the same picture as drawing from coordinates L8 to L7 to L9.

We now have the hierarchical structure shown in Figure 5.

The heart of the display file generator in the large computer is a recursive routine which climbs down this structure from the top going down the left-most dendrite until it finds a terminal element, or the end of the string. It communicates any displayable element to the small computer, deletes it from the string (but not from the memory), and climbs back up the structure until it finds the next non-terminal element. It then repeats the previous steps. When the whole structure has been examined, and all the terminal elements communicated to the display, the larger computer sends a pointing mode command and enters the quiescent state.

As the routine goes down the structure it saves the path (dendrite) in order to know the HISTORY of each entity transmitted to the display. This history is used by many routines, for example, to change or delete items in the associative memory as the graphical entities that they represent are moved or deleted from the display. The histories shown in Figures 3 and 4 are very short; however, since pictures can be defined within pictures to any depth short of infinite, histories can be quite long. An obvious example of infinite re-

*A dendrite is a unique path down from the topmost element of the hierarchy.



POSITION, LINE, POINT, and CHARACTERS are all at the same heirarchical ply (level) in this figure. In general this may not be the case.

FIGURE 5—Hierarchical structure of Figure 3.

ursion occurs if we define a picture within itself (the barbershop-mirror problem).

HISTORY is essentially a cross-reference table between the display file in the small computer and the associative memory in the large computer, yet HISTORY is itself part of the associative memory. This is in sharp contradistinction to several other graphic systems which either use a sequential cross-reference table, or some mechanism other than the main storage itself.

Ascribing meaning to graphical data

Early graphics systems, such as the GM DAC⁶ system and Sketchpad,⁷ were little better than automated drafting boards. This statement is not intended in any way to belittle their efforts, but merely to underline the fact that there was very little that could be done with a picture once it had been generated. Certain of Sutherland's illustrations are quite startling in their apparent sophistication, but generally return to the use of constraints (which were satisfied using least squares fit, which is an energy constraint in engineering). In endeavoring to ascribe meaning to pictures, later investigators were forced to use data structures in a more sophisticated manner, and it became obvious that associations should be much more complex than the original ring structures, etc. The CORAL⁸ language, APL,⁹ and AL, the language described by Feldman, are all outgrowths

of the need to ascribe extra associations and meanings to a picture. Many are now working on this problem but information in technical literature is relatively sparse. To illustrate the techniques being developed at the University of Michigan, and to show the power of the associative language, a detailed example will now be given.

The problem chosen is that of finding a list of all points, joined by lines to a given point. This may seem relatively trivial, since it requires only a knowledge of the end-points of the lines, a selection of those lines which end on the given point, and continuation by asking for all points now joined to the new set of end-points. There are two problems with this solution. The first is that we are potentially in an infinite recursion, where we must exclude all points previously found from our next search. The second is that the picture does not necessarily use the same name for points occupying the same space. As an example, in constructing the letter Y we draw two lines in a continuous fashion through the center point, and then draw the third line to intersect these two at the center point. Unless special software has been produced to check for this, there is no reason why the data structure should describe the third line as terminating at the same point *name* as that of the first two-line intersection.

This is really another example of the synonym in keyword searches, or common node points of graph theory. The problem occurs, because although the coordinates of the end-points may be the same, they have been defined at different times with different external or internal names. Figure 6 is in three parts, it fully describes the process of finding all points. Part A is a description in a normal language; Part B is a solution in meta-language, using the associative language of an

<p>1. Name Program POICON Read input on teletype</p>	<pre>FUNCTION POICON READ INTO P1 ANSWER: PANS TEMPORARY: NEWP,HOLDS CURRENT POINTS GENERATED</pre>	<pre> #(DS,POICON, # (DS,P1,#(RS)) # (DS,PANS) # (DS,NEWP,#(P1)) # (TWO))</pre>
<p>2. Find all synonyms of new points, remove all previously found points (relative complement) find union of new points with answer set, and with latest new points to give newest set of points.</p>	<pre>COORD(NEWP) = X,FIND X COORD(Y) = X,FIND Y RELCOM(Y,PANS)=Z UNION(Z,PANS)=PANS UNION(Z,NEWP)=NEWP</pre>	<pre> #(DS,TWO,(# (RL,COORD,#(NEWP),*X*) # (RL,COORD,*Y*,#(X)) # (ROOM,#(Y),#(PANS),Z) # (UN,#(Z),#(PANS),PANS) # (UN,#(Z),#(NEWP),NEWP) # (THREE))</pre>
<p>3. Find all lines emanating from the new points, and then all ends of these lines. Remove all previously found points from this set (relative complement). Union the new points with the answer set. If there are no new points found, exit, printing the answer on the teletype.</p>	<pre>END(X) = NEWP,FINDX END(X) = NEWP,FIND NEWP RELCOM(NEWP,PANS) NEWP UNION(NEWP,PANS) PANS IF NEWP IS NULL,EXIT AND PRINT PANS, OTHERWISE REPEAT FROM STEP 2.</pre>	<pre> #(DS,THREE,(# (RL,END,*X*,#(NEWP)) # (RL,END,#(X),*NEWP*) # (ROOM,#(NEWP),#(PANS),NEWP) # (UN,#(NEWP),#(PANS),PANS) # (EQ,#(NEWP),, # (PANS)), # (TWO))</pre>

a. NORMAL LANGUAGE DEFINITION b. META-LANGUAGE PROGRAM c. TRAMP PROGRAM

FIGURE 6—Point connectivity: Definition in an associative structure

earlier section; Part C is a set of TRAMP language statements describing the same process. It is important to note that the transformation from one language to the next is relatively simple, because of the similarity between the original language statement, the meta-language, and the TRAMP statements.

Some simple examples

The photographs in this section provide some simple examples of the system. Figure 2 shows some basic options available to the user. The various items were defined through communication similar to that shown in Figure 4. Items may be added to the menu at any time. An example of this is given in Figure 7 where the user has explicitly associated the word appearing on the screen with a UMIST procedure in the 360. Figure 8 shows several lines and character strings, and Figure 9 shows that pictures may be called back from 360 storage for further processing.

SUMMARY OF THE PRESENT SYSTEM AND FUTURE PLANS

The illustrations of the last two sections are not intended to be particularly erudite or sophisticated, but to show that the present use of an associative data structure gives the programmer extreme generality. Although the original PDP-8 macro language was developed to facilitate drawing, the associative data structure allowed us to ignore such problems as connectivity or geometrical interpretation until later in the implemen-

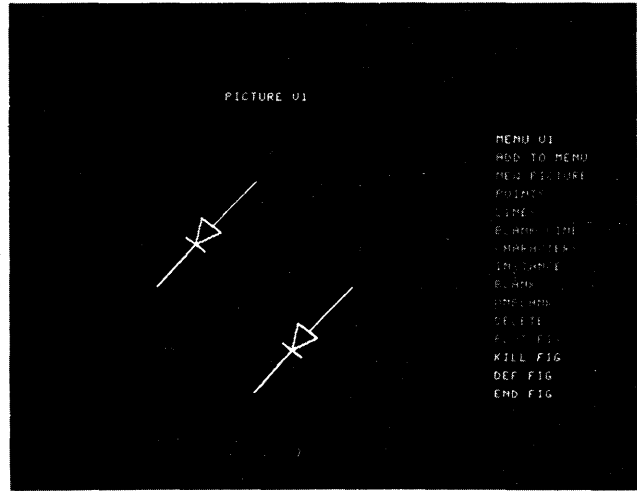


FIGURE 8—Two diodes

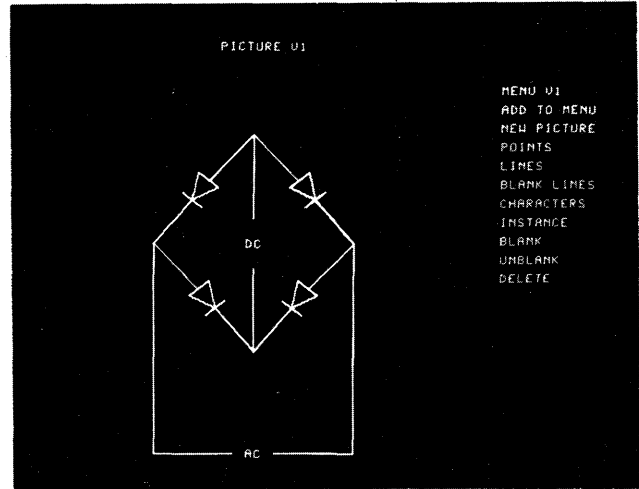


FIGURE 9—Combination of pictures

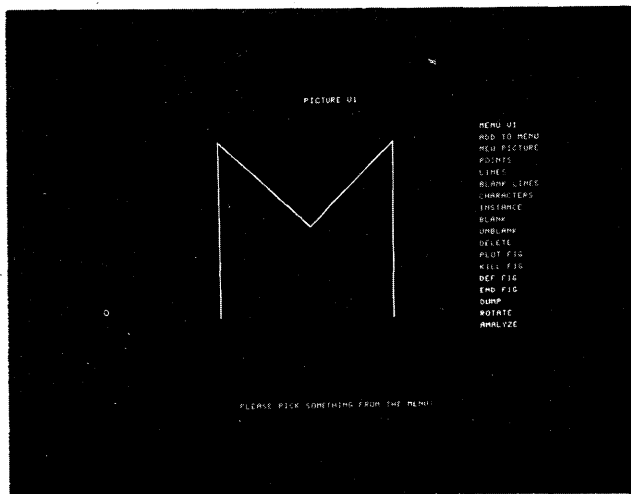


FIGURE 7—User-augmented menu

tation: the information was not lost, but merely not being used.

Obviously the relational aspects described in a previous section allow for even more sophisticated programming. For we do not have to say that if A is connected to B, and B is connected to C, that A is connected to C, and make recursive searches, but merely state that the "connected to" relation is transitive, and allow the relational programs to generate the necessary calls to give the complete solution set.

A further illustration of the use of associative and relational data structures includes the ability to recognize primitive objects from their description. Thus we may define a triangle by drawing three lines, and then ask for this picture to be

"analyzed." The analysis program may require man-machine interaction, in which case the user gives the word "triangle" and the program checks that the picture does, in fact, satisfy the definition of a triangle, or alternatively checks that the three lines to which the user points are indeed connected together at their end-points to form a triangle. In another type of analysis, the data structure could be heuristically searched to find the primitive structure of the various parts, and come up with the attributes of the various parts, such as triangle, square, rectangle, etc. This last mode of operation is obviously time-consuming, and will not probably prove to be an economical way of operation in the near future.

A further extension will allow the user to ask for computation of the various angles of the triangle given the sides, or allow generation of algebraic formulae from the data structure of the picture. This requires the association of the shape of the triangle with its formulae, i.e., trigonometric or geometric properties of triangles.¹⁰ This latter concept allows us to look toward the future where an interactive graphic system may involve the drawing of pictures which are later analyzed geometrically, for use in engineering formulae to calculate stresses, voltages, volumes, or temperature of the physical objects represented in a drawing.

In this article we have endeavored to show that an associative approach to graphics design gives advantages both to the communication of information over slow-speed transmission lines, as well as to the ascribing of meaning to pictures. Other work is presently under way on better input/output devices, or higher band-width transmission devices. We hope to determine the design parameters of the man-machine interface, such as the effect on a user working within a timesharing system not committed totally to him. We also hope to determine design aspects for better hardware (e.g., would floating point hardware on the small computer significantly affect the system?). Finally, we intend to build up the complexity of the small terminal until the economies of size are balanced by diseconomies of committed use; for although the larger computer generally provides

lower computer cost per instruction, (economies of size), a committed system, although small, allows maximum human interaction. Graphics systems require so much interaction that it could be an unwarranted burden on a large computer, which immediately forces us to a diseconomy of size; but an unsophisticated terminal could be a frustration to the user, impairing his productivity or creativity. The question of the relative power of the terminal is therefore one of major significance in the next few years.

REFERENCES

- 1 MTS: *Michigan Terminal System*
2nd ed Computing Center University of Michigan December 1967
- 2 D E MILLS
The data concentrator
Project Comcomp Technical Report 8 May 1968
- 3 D G GORDON E H SIBLEY R W TAYLOR
GAMES: A graphical associative memory system
Concomp Project University of Michigan in press October 1968
- 4 W L ASH E H SIBLEY
TRAMP: an interpretive associative processor with deductive capabilities
Proc ACM 1968 pp 143-156 Brandon Systems Press Princeton N J
May 1968
- 5 J A FELDMAN
Aspects of associative processing
Technical Note 1965-13 Lincoln Laboratory Lexington Mass
April 1965
- 6 E L JACKS
A laboratory for the study of graphical man-machine communications
Proc AFIPS 1964 FJCC Vol 26 Part 1 p 343
- 7 I E SUTHERLAND
Sketchpad, a man-machine graphical communication system
Proc AFIPS 1963 SJCC Spartan Books Baltimore Md pp 329-346
- 8 L G ROBERTS
Graphical communications and control languages
Second Congress on Information System Sciences Spartan Books Baltimore Md 1964
- 9 G G DODD
APL—A language for associative data handling in PL/I
Proc AFIPS 1966 FJCC pp 677-684
- 10 E H SIBLEY
The engineering assistant: Design of a symbol manipulation system
Technical Report Concomp Project University of Michigan
August 1967

Description of a set-theoretic data structure

by DAVID L. CHILDS

University of Michigan
Ann Arbor, Michigan

INTRODUCTION

The overall goal, of which this paper is a part, is the development of a machine-independent data structure allowing rapid processing of data related by arbitrary assignment such as: the contents of a telephone book, library files, census reports, family lineage, graphic displays, information retrieval systems, networks, etc. Data which are non-intrinsically related have to be expressed (stored) in such a way as to define the way in which they are related before *any* data structure is applicable. Since any relation can be expressed in set theory as a set of ordered pairs and since set theory provides a wealth of operations for dealing with relations, a set-theoretic data structure appears worth investigation.

A Set-Theoretic Data Structure (STDS) is a storage representation of sets and set operations such that: given any family of sets η and any collection S of set operations an STDS is any storage representation which is isomorphic to η with S . The language used with an STDS may contain any set-theoretic expression capable of construction from η and S . Every stored representation of a set must preserve all the properties of that set and every representation of a particular set must behave identically under set operations.

General storage representation

An STDS is comprised of five structurally independent parts:

- 1) a collection of set operations S .
- 2) a set of datum names β .
- 3) the data: a collection of datum definitions, one for each datum name.
- 4) a collection of set names η .
- 5) a collection of set representations, each with a name in η .

The storage representation is shown schematically in

Figure 1. In order for an STDS to be practical the set operations must be executed rapidly. If any two sets can be well ordered (a linear order with a first element) such that their union preserves this well-ordering, then the subroutines needed for set operations just involve a form of merge or, at worst, a binary search of just one of the sets. It was shown in another paper¹ that *any* set defined over β could be so ordered. Sets are represented by blocks of contiguous storage locations with η containing names of all the sets. The set β is the set of all datum names, and is represented by a contiguous block of storage locations; the address of a location in the β -block is a datum name and an element of β . The content of a location in the β -block is the address of a stored description of that datum (see Figure 1). The contents of the β -block and the η -block are the *only* pointers needed for the operation of an STDS. The storage representations of the individual sets *do not* contain pointers to other sets, but contain information about datum names. Since each set representation has only one pointer associated with it, the set representation can be moved throughout storage without affecting its contents or the contents of any other set representation—only the one pointer in η is affected. Updating set representations is virtually trivial. Elements to be deleted are replaced by the last element in the set. Elements to be added are added to the end of the set representation as space allows. When contiguous locations are no longer available a new set is formed and the element in η that referenced the set before it was extended now references a location that indicates that the set is now the union of two set representations. (In a paging structure such sets could be kept on the same page.) This demonstrates two different kinds of sets in η : *generator* sets and *composite* sets. Only the generator sets have storage representations, the composite sets are unions of generator sets, and the generator sets are mutually disjoint. Since no duplication of storage of sets is necessary and since the set representa-

tions are kept to a minimum by containing just the elements of the sets and no pointers, an STDS is intrinsically a minimal storage representation for arbitrarily related data.

Operation of an STDS

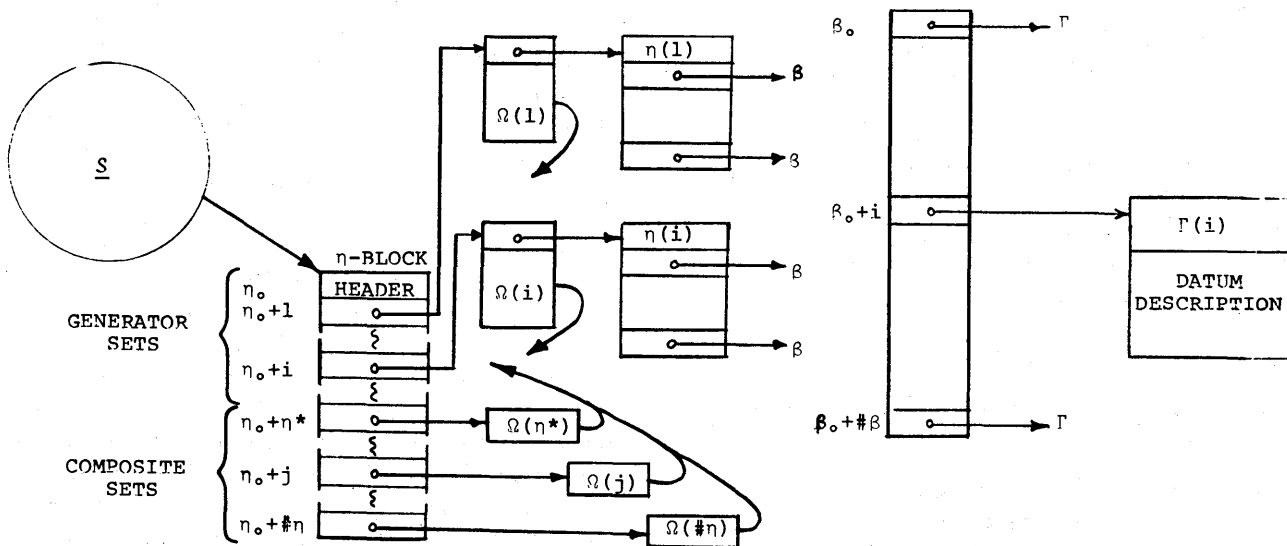
An STDS relies on set operations to do the work usually allocated to pointers or hash-coding as in list structures, ring structures, associative structures, and relational files. A set operation of S is represented by a subroutine which accesses sets through pointers in η . Again it should be stressed that no pointers exist between sets, hence the set operations S act as the *only* structural ties between sets. Since S will allow any set-theoretic operation, S will be rich enough that all information between sets may be expressed by a set-theoretic expression generated from the operations of S . Any expression establishes which sets are to be accessed and which operations are to be performed within and between these sets; therefore all pages needed for completion of an expression are known before the expression is executed. Complementing the set operation subroutines are some strictly storage manipulation subroutines. These, however, are not reflected in any

set-theoretic expression. These routines change storage modes and perform sorts and orderings. A fast sort routine has been programmed with execution times as a linear function of the number of words to be sorted. (On an IBM 7090 this sort ordered 1000 words in 0.35 seconds and 10,000 words in 3.3 seconds. The nature of this sort is such that on an IBM 360/67 it may sort up to 60,000 bytes per second. This routine is presently being programmed.) Another subroutine which is crucial to the operation of an STDS is the tau-ordering routine.¹ This routine gives a well-ordering which is preserved under union.

Details of β -block

The β -block is a section of contiguous storage locations with β_0 as the address of the head location. The first location containing a datum-pointer has the address β_0+1 , and the location of the i -th datum-pointer is β_0+i . Let $\#\beta$ represent the total number of datum-pointers, then the last address of the β -block would be $\beta_0+\#\beta$. β is the set of datum-names or locations of datum-pointers in the β -block. Since all datum-pointers are located between β_0+1 and $\beta_0+\#\beta$, let β be the set of integers $\{1,2,\dots,\#\beta\}$. Therefore any integer i such

SET OPERATIONS: S SET NAMES: η SET REPRESENTATIONS DATUM NAMES: β DATA STORAGE



$\Omega(1)$ through $\Omega(n^*-1)$ are sets of pointers to COMPOSITE SETS in n -BLOCK
 $\Omega(n^*)$ through $\Omega(\#n)$ are sets of pointers to GENERATOR SETS in n -BLOCK
 $\Gamma(i)$ are sets of pointers to GENERATOR SETS in n -BLOCK

FIGURE 1

that $1 \leq i \leq \# \beta$ is the datum-name for the i -th datum-pointer. The i -th datum-pointer locates a block of storage containing a description of the i -th datum and all the generator set names (elements of η) for which the i -th datum name is a constituent.

Details of η -block

The η -block is similar to the β -block with η_0 and $\# \eta$ as the address of the head location and cardinality respectively. The contents of the η -block are pointers. These pointers are of two types and are distinguished by an integer η^* such that $1 < \eta^* \leq \# \eta$. For all $1 \leq i < \eta^*$, i is the name of a generator set, and for all $\eta^* \leq i \leq \# \eta$, i is a composite set. A generator set has a set representation while a composite set does not since it is the union of some generator sets. For $i \geq \eta^*$ the pointer-in $\eta_0 + i$ locates a section of storage containing names of generator sets. For $i < \eta^*$ the pointer in $\eta_0 + i$ locates a section of storage containing all composite set names that use i , and a pointer to the set representation of i . Since all generator sets are mutually disjoint and since only generator sets have a storage representation, there is no duplication of storage in an STDS.

Set representation

In order to insure fast execution times for the set operations in S , the sets involved must be isomorphic to a *unique* linear representation of their elements. Unique is used here to mean unique relative to some predefined well-ordering relation, such that independently of how the set is presented to a machine the ordering of its elements will always be the same. This well-ordering must be preserved under union. Any ordering satisfying the above conditions is adequate for the efficient operation of an STDS.¹

Since the set representatives must be isomorphic to the sets they represent, every set representation must reflect the rank and preserve the order (if any) of the sets and their elements. Let $A = \langle a, b, c \rangle$, $B = \{a, b, c\}$, and $C = \{c, b, a\}$ then B and C must have the same set representation while A must have a completely different representation. For simple sets like these, adequate representations are trivial, such is not always the case, however.

Complexes and n -tuples

If an STDS is to be general then it will have to accommodate more imaginative sets than the ones above. Let $W = \{a, b, \{\{c\}\}, \langle a, \{b, d\}, c \rangle, \langle \langle a, b \rangle, c \rangle\}$ and $V = \{\langle a, b, c \rangle, \langle \langle a, b \rangle, \langle c, d \rangle \rangle, \langle d, a, \rangle, \{\{c\}\}, b\}$. In order for set operations on these sets to fall within the allotted time bounds, the storage

representations of W and V must satisfy the well-ordering condition. Such a representation is not immediately obvious. Two problems arise. (1) The first problem is machine oriented in that an ordered set in set theory is defined through nesting and repetition of the elements of the set. For example the Kuratowski definition of ordered pair gives $\langle a, b \rangle = \{\{a\}, \{a, b\}\}$. Since any machine representation will induce an order on the elements of a set by their location in storage, this may be utilized instead of relying on redundancy of storage. This in turn may present problems in preserving the isomorphism between sets and their set representations, since an unordered set must have a unique representation and no ordering on its elements. (2) The second problem is much allied with the first except that it is more biased towards the foundations of set theory. There seems to be a general lack of precision in set theory when ordering beyond a pair is involved. No set representation of ordered triples, ordered quadruples, quintuples, sextuples, etc., is given save for an arbitrary assignment in terms of ordered pairs. (This problem is discussed by Skolem³). For example $\langle a, b, c, d \rangle$ has no set equivalent independent of ordered pairs, it is given one of the following as its canonical form: $\langle \langle a, b \rangle, \langle c, d \rangle \rangle$; $\langle a, \langle b, \langle c, d \rangle \rangle \rangle$; $\langle a, \langle \langle b, c \rangle, d \rangle \rangle$; $\langle \langle \langle a, b \rangle, c \rangle, d \rangle$; $\langle \langle a, \langle b, c \rangle \rangle, d \rangle$; or $\{\langle 1, a \rangle, \langle 2, b \rangle, \langle 3, c \rangle, \langle 4, d \rangle\}$. Clearly each of these sets has independent stature and assigning one as a canonical form of the other precludes the use of the others. The problem with ordered tuples is compounded in that though they are defined as sets they are excluded from meaningful set operations. The intersection between quadruples $\langle a, b, c, d \rangle$ and $\langle x, b, c, d \rangle$ is always empty unless $a = x$, and even then it depends on which assignment is used. In another paper⁴ the definition of a "complex" is presented which preserves the distinction between different nestings of ordered pairs, does not require order to be defined by repetition, and does not arbitrarily exclude certain sets from being operated on by set operations. The formal definition of a complex is given by the following, where N is the set of natural numbers.

DEFINITION OF A COMPLEX: Any two sets A and B form a complex $(A; B)$ if and only if $(\exists X)$

$$(\exists Y)(X \in \{A, B\})(Y \in \{A, B\})[(\forall x \in X)(\exists i \in N)$$

$$(\{\{x\}, i\} \in Y) \& (\forall y \in Y)(\exists j \in N)(\exists x \in X)(\{\{x\}, j\} = y)]$$

This definition is stated in such a way as not to presuppose any ordering in $(A; B)$ of A before B , insuring that a complex be an unordered coupling of two sets, each bearing a mutual dependence on the other. The defi-

dition states that for every element x of one of the sets, X , the other set, Y , contains an element containing a natural number and a set whose only element is x ; and that Y is such that every element of Y contains only a natural number and a singleton set containing an element of X (either $X=A$ and $Y=B$, or $X=B$ and $Y=A$, but not both). Let $A = \{a, b, c\}$, $B = \{\{a\}, 1, \{b\}, 3, \{c\}, 963, \{b\}, 6\}$ and let $C = \{a, b, \{b\}, 3, \{a\}, 1, \{d\}, 6\}$ then $(A;B)$, $(B;A)$ and $(A \cap C; B \cap C)$ are complexes, while $(A;A)$, $(A;C)$, $(A;B \cap C)$ and $(A \cap C; B)$ are not complexes. From the definition it should be noticed that if $(A;B)$ is a complex then $(B;A)$ is the same complex and $A \neq B$. Without giving a formal definition here let $x \in_i A$ be understood to mean that x is in the i -th position of the complex A , then a notational schema for a complex is given by:

DEFINITION SCHEMA: $\{x^i: \Psi(x,i)\} = A$ iff $[(\forall x)$
 $(\forall i \in \mathbb{N}) (x \in_i A \leftrightarrow \Psi(x,i)) \& A$ is a complex]

These results allow a set theoretic foundation for the following equivalent notations:

set	$\{a, b, c\} = \{a^1, b^1, c^1\}$
ordered pair	$\langle a, b \rangle = \{a^1, b^2\}$
ordered triple	$\langle a, b, c \rangle = \{a^1, b^2, c^3\}$
ordered quadruple	$\langle a, b, c, d \rangle = \{a^1, b^2, c^3, d^4\}$
ordered pairs of ordered pairs	$\langle \langle a, b \rangle, \langle c, d \rangle \rangle = \{\{a^1, b^2\}^1, \{c^1, d^2\}^2\}$
	$\langle a, \langle b, \langle c, d \rangle \rangle \rangle = \{a^1, \{b^1, \{c^1, d^2\}^2\}^2\}$
	$\langle a, \langle \langle b, c \rangle, d \rangle \rangle = \{a^1, \{\{b^1, c^2\}^1, d^2\}^2\}$
	$\langle \langle \langle a, b \rangle, c \rangle, d \rangle = \{\{\{a^1, b^2\}^1, c^2\}^1, d^2\}$
	$\langle \langle a, \langle b, c \rangle \rangle, d \rangle = \{\{a^1, \{b^1, c^2\}^1\}^1, d^2\}$
	$\langle \langle 1, a \rangle, \langle 2, b \rangle, \langle 3, c \rangle, \langle 4, d \rangle \rangle = \{\{1^1, a^2\}, \{2^1, b^2\}, \{3^1, c^2\}; \{4^1, d^2\}\}$

and from the beginning of this section,

$$W = \{a^1, b^1, \{\{c^1\}\}, \{a^1, \{b^1, d^1\}^2, c^3\}, \{\{a^1, b^2\}, c^1\}$$

$$V = \{\{a^1, b^2, c^3\}, \{\{\{a^1, b^2\}, \{c^1, d^2\}\}, \{d^1, a^2\}^2\}, \{\{c^1\}\}, b^1\}$$

Since for all a , $\{a^1\} = \{a\}$, the exponent '1' is optional. It should be stressed that the symbol 'x' has no meaning apart from being enclosed by set brackets. mean-
 $\{a^6, b^8\}$, then $a \in_i A$ and $b \in_i A$ are true, but $a^6 \in_i A$ is meaningless. For examples of set operations between complexes see Figure 2.

- 1) $\langle a, b, c \rangle \cap \langle x, b, y \rangle = \{b^2\}$
- 2) $\langle a, b, c \rangle \cup \langle x, y \rangle = \{a^1, x^1, b^2, y^2, c^3\}$
- 3) $\{a, b, c\} \cap \langle a, x, y \rangle = \langle a \rangle = \{a^1\} = \{a\}$
- 4) $\cup \{a^1, b^2, (x^1, c^3)^3, (y^2, d^4)^4\} = (x^1, c^3) \cup (y^2, d^4) = \langle x, y, c, d \rangle$
- 5) $\langle a, b, z \rangle \Delta \langle a, y, c \rangle \Delta \langle x, b, c \rangle = \langle x, y, z \rangle$
- 6) $\langle a, b, c, d \rangle \sim \langle x, y, c, d \rangle = \langle a, b \rangle$

FIGURE 2—Set operations between complexes

Set operation subroutines

The viability of an STDS rests not only on the speed of the set operations, but also on their scope. Table 1 presents some available set operations for constructing questions in any way compatible within a parent language. (For those who are not familiar with the set-theoretic definitions or are not accustomed to the notation preferred in this monograph, the definitions are given in Appendix I.) These subroutines are presented in a format compatible with FORTRAN, and with MAD if periods are added as in the examples to follow. The argument represented by C in the subroutines can be deleted. This default case assigns a temporary storage block whose location is returned in D , as if it were a permanent storage location, i.e., $D = UN(A, B)$. Since all subroutines operate on the name of a storage block representing a set, then for all subroutines that return a name, any degree of nesting of these subroutines within subroutines is allowable (see examples). Since the only restriction on a set representation is that it be isomorphic to the set and have a predefined well-ordering on its elements, there are many storage configurations available. MODE allows a choice of different storage configurations for non-set-theoretic needs. Though all the subroutines appear to be defined just for sets, they are defined for any complex as well. However, to make use of complexes that are not sets since they allow the extension of binary relation properties (e.g., domain, image, relative product, restriction, etc.) to sets of arbitrary length n -tuples, further delimiters must be included. For example using 'Q' and an extra argument the I -th relative produce of A with B could be QRP (I, A, B, C), and the I -th domain of A could be QDM (I, A, C), and QELM (I, A, B) could represent the question "is A an I -th element of B ?"

Some applications

This section will be devoted to examples demonstrating the applicability of set-theoretic questions. For a germane reference on computer graphics see Johnson.² The first two examples are to give some indication of execution times. The two examples were run on an IBM 7090, the times may or may not be characteristic of the potential speeds in an STDS. With just two examples no claims can be made other than that two examples were run with the following results:

EXAMPLE 1: Given a population of 24,000 people and a file F containing a ten-tuple for each person such that each ten-tuple is of the form <age, sex, marital status, race, political affiliation, mother tongue, employment status, family size, highest school grade completed, type of dwelling>, the following four questions were asked:

- a. Find the number of married females:
Answer: 6,015 Time: 0.50 seconds
- b. Find the number of people of Spanish race whose mother tongue is not Spanish.
Answer: 1,352 Time: 0.48 seconds
- c. Find the number of people aged 93 or 94.
Answer: 46 Time: 0.73 seconds
- d. Find the number of males and unmarried females.
Answer: 17,985 Time: 0.55 seconds
- e. Find the number of males between the ages of 20 and 40.
Answer: 588 Time: 0.62 seconds.

EXAMPLE 2: Given a population of 3,000 people and given two collections, A and B, of subsets from this population such that: A contains 20 sets of 500 people, and B contains 500 sets of 20 people. Find the set of people belonging to *some* set in A, to *all* sets in A, and to an *odd* number of sets in A; and similarly for B.

Results	A-Times	B-Times
a. people in some set	0.73 sec	0.76 sec
b. people in all sets	0.48 sec	0.05 sec
c. people in odd no. of sets	0.76 sec	0.78 sec

A point to notice is that where every element has to be accessed, as in (a) and (c), the times are dependent on the total number of elements included ($\xi(A) = \xi(B) = 10,000$) and not the number of sets involved (20 for A and 500 for B).

Examples three and four are presented with MAD as the parent language, therefore all the subroutine names must end with a period.

EXAMPLE 3: Let six sets A,B,C,D,E, and F be the membership lists of six country clubs. For each male resident of Ann Arbor, let there be a datum in β for a data-block containing: person's name, address, phone number, credit rating, age, golf handicap, wife's name (if any), political affiliation, religious preference, and salary. The set η will contain the names of the sets, namely: A(0), B(0), C(0), D(0), E(0), F(0). This along with the collection \mathcal{S} of set operations allows answering the following questions.

- 1) How many members belong to club A or B but not C?
- 2) Find the phone numbers of members in an odd number of clubs.
- 3) Get addresses of members belonging to one and only one club.
- 4) Get addresses and phone numbers of people not in any club.
- 5) Find members of A that are not also in B but who may be in C only if they are not in D, or in E if they are not in F.
- 6) Get the average credit rating of members belonging to exactly three clubs.

The possible questions may become ridiculously involved and may interact with any spontaneously constructed sets. For example of the latter, let X be the set of Ann Arbor males born in Ann Arbor.

- 7) Find the average age of members born in Ann Arbor and compare with average age of members not born in Ann Arbor.

The answers to (1) through (7) formulated in an STDS are expressed below, with N and M representing real numbers, and with BB for β and NN for η .

- 1) N = C. (RL. (UN. (A,B),C))
ans: N
- 2) ACC. (1,SD. (1,NN),Q)
ans: Q Format 1 gives phone numbers (see Table 1, #25)
- 3) ACC. (2,EX. (1,NN),Q)
ans: Q Format 2 gives addresses
- 4) ACC. (3,RL. (BB,UN.(1,NN)),Q)
ans: Q Format 3 gives phone numbers and addresses
- 5) RL.(RL.(A,B),UN.(RL.(D,C),RL.(F,E)),Q)
ans: Q
- 6) ACC.(4,EX.(3,NN),Q)
N = 0
THROUGH LOOP,
FOR I=1,1,I.G.C.(Q)

LOOP N = N + Q(I)
 N = N/C.(Q)
 ans: N Format 4 gives credit rating

7) N = 0
 M = 0
 ACC.(5,X,T)
 THROUGH LOOP1,
 FOR I=1,1,I.G.C.(T)

LOOP1 N = N + T(I)
 ACC. (5,RL.(BB,X),P)
 THROUGH LOOP2,
 FOR I=1,1,I.G.C.(P)

LOOP2 M = M + P(I)
 N = N/C.(T)
 M = M/C.(P)

ans: N and M are the respective average ages
 Format 5 gives ages

EXAMPLE 4: Family lineage is easily expressed in STDS. With just five initial relations defined over a population U, all questions concerning family ties may be expressed.

Let U be a population of people and let

$M = \{ \langle x,y \rangle : y \text{ is the mother of } x \}$

$F = \{ \langle x,y \rangle : y \text{ is the father of } x \}$

$S = \{ \langle x,y \rangle : y \text{ is a sister of } x \}$

$B = \{ \langle x,y \rangle : y \text{ is a brother of } x \}$

$H = \{ \langle x,y \rangle : y \text{ is a husband of } x \}$

Let X be any subset of the population U, find

- 1) the set G of grandfathers of X.
 $G = F[(FUM)_X]$ set notation
 $IM.(F,IM.(UN.(F,M),X),G)$ in an STDS
- 2) the set GF of grandfathers of X on the father's side.
 $GF = F[F[X]]$ set notation
 $IM.(F,IM.(F,X),GF)$ STDS
- 3) the set GM of grandfathers of X on the mother's side
 $GM = G \sim GF$ set notation
 $RL.(G,GF,GM)$ STDS
- 4) the set GR: the grandfather relation over U.
 $GR = (FUM)/F$ set notations
 $RP.(UN.(F,M),F,GR)$ STDS
- 5) the general relation: $P = \{ \langle x,y \rangle : y \text{ is a parent of } x \}$
 $P = F \cup M$ set notation
 $UN.(F,M,P)$ STDS
- 6) the general relation: Sibling, L.
 $L = S \cup B$ set notation
 $UN.(S,B,L)$ STDS

7) the general relation: Children, C.
 $C = \overline{M \cup F} = \overline{P}$ set notation
 $CV.(P,C)$ STDS

8) the general relation: Aunt, A.
 $A = (P/S) \cup (P/B/\overline{H})$ set notation
 $UN.(RP.(P,S),RP.(P,RP.(B,CV.(H))),A)$ STDS

9) the general relation: Wife, W.
 $W = \overline{H}$ set notation
 $CV.(H,W)$ STDS

10) the general relation: Cousin, K.
 $K = P/L/C$ set notation
 $RP.(P,RP.(L,C),K)$ STDS

11) the general relation: Half-sibling, HS.
 $HS = P/C \sim (M/\overline{M} \cup F/\overline{F})$ set notation
 $RL.(RP.(CV.(C),C),IN.(RP.(M,CV.(M))),$
 $RP.(F,CV.(F))),HS)$ STDS

12) people in X with no brothers or sisters.
 $Q = X \sim \mathfrak{D}(L)$ set notation
 $RL.(X,DM.(L),Q)$ STDS

13) find all relations of X to a set Y such that Y is equal to the image of X.
 $Q = \{ A : (A \in \eta) (Y = A[X]) \}$ set notation
 $DC.(X,NN,T)$ STDS
 THROUGH LOOP, FOR I=1,1,I.G.C.(T)
 $B = IM.(T(I),X)$
 LOOP WHENEVER EQL.(Y,B).E.1,
 $UN.(Q,S.(T(I)),Q)$

Many more possibilities are available and might be tried by the reader.

CONCLUSION

The purpose of an STDS is to provide a storage representation for arbitrarily related data allowing quick access, minimal storage, generality, and extreme flexibility. With the definition of a complex, a predefined well-ordering, and the operations of set theory, such a storage representation can be realized.

Set-theoretic definitions

Conventions

The logical connectives 'and,' 'or,' 'exclusive-or' are represented by ' \wedge ,' ' \vee ,' ' Δ .' 'For all x,' 'for some x,' 'for exactly n x' will be represented by ' $\forall x$,' 'Ex', 'E(n)!x.' Parentheses are used for separation, and as usual the concatenation of parentheses will represent conjunction.

'A' will be a *set* if and only if (a) it can be represented formally by abstraction (i.e., $A = \{x:\theta(x)\}$ where $\theta(x)$ is a predicate condition specifying the allowable elements 'x'); (b) 'A' can be represented by $\{,\}$ enclosing the specific elements of 'A.'

Definitions

The symbol ' ϵ ' means 'is an element of'; $x \epsilon A$ reads: "x is an element of A."

1) UNION

a) binary union of two sets A and B

$$A \cup B = \{x:(x \epsilon A) \vee (x \epsilon B)\}$$

b) unary union of a family G of sets

$$\bigcup G = \{x:(\exists A \epsilon G) (x \epsilon A)\}$$

c) indexed union of a set f(A) over the family G

$$\bigcup_{A \epsilon G} f(A) = \{x:(\exists A \epsilon G) (x \epsilon f(A))\}.$$

2) INTERSECTION

a) binary intersection of A and B

$$A \cap B = \{x:(x \epsilon A) \wedge (x \epsilon B)\}$$

b) unary intersection of a family G

$$\bigcap G = \{x:(\forall A \epsilon G) (x \epsilon A)\}$$

c) indexed intersection of f(A) over the family G

$$\bigcap_{A \epsilon G} f(A) = \{x:(\forall A \epsilon G) (x \epsilon f(A))\}.$$

3) SYMMETRIC DIFFERENCE

a) binary symmetric difference of A and B

$$A \Delta B = \{x:(x \epsilon A) \Delta (x \epsilon B)\}^*$$

* even though the symbol ' Δ ' has two different meanings, no confusion is likely

b) unary symmetric difference of G

$$\Delta G = \{x:(\text{for an odd number of } A \epsilon G) (x \epsilon A)\}$$

c) indexed symmetric difference of f(A) over G

$$\Delta_{A \epsilon G} f(A) = \{x:(\text{for odd no. of } A \epsilon G) (x \epsilon f(A))\}.$$

4) RELATIVE COMPLEMENT

$$A \sim B = \{x:(x \epsilon A) \wedge (x \notin B)\}.$$

5) EXACTLY N!

the set of elements common to exactly 'n' elements of a given set G is represented by:

$$E_n G = \{x:(E(n)!A \epsilon G) (x \epsilon A)\}.$$

6) DOMAIN of a set A

$$\mathfrak{D}(A) = \{x:(\exists y) (\langle x,y \rangle \epsilon A)\}^*.$$

* $\langle x,y \rangle$ represents an ordered pair

7) RANGE of a set A

$$\mathfrak{R}(A) = \{y:(\exists x) (\langle x,y \rangle \epsilon A)\}.$$

8) IMAGE of B under A

$$A[B] = \{y:(\exists x \epsilon B) (\langle x,y \rangle \epsilon A)\}.$$

9) CONVERSE IMAGE of B under A

$$[B]A = \{x:(\exists y \epsilon B) (\langle x,y \rangle \epsilon A)\}.$$

10) CONVERSE of A

$$\bar{A} = \{\langle y,x \rangle : \langle x,y \rangle \epsilon A\}.$$

11) RESTRICTION

$$A|B = \{\langle x,y \rangle : (\langle x,y \rangle \epsilon A) \wedge (x \epsilon B)\}.$$

12) RELATIVE PRODUCT of A and B

$$A/B = \{\langle x,y \rangle : (\exists z) (\langle x,z \rangle \epsilon A) \wedge (\langle z,y \rangle \epsilon B)\}.$$

13) CARTESIAN PRODUCT of A and B

$$A \times B = \{\langle x,y \rangle : (x \epsilon A) \wedge (y \epsilon B)\}.$$

14) DOMAIN CONCURRENCE of X relative to A

$$\mathfrak{D}(X:A) = \{B:(B \epsilon A) \wedge (X \subset \mathfrak{D}(B))\}.$$

15) RANGE CONCURRENCE of X relative to A

$$\mathfrak{R}(X:A) = \{B:(B \epsilon A) \wedge (X \subset \mathfrak{R}(B))\}.$$

16) SET CONCURRENCE of X relative to A

$$\mathfrak{S}(X:A) = \{B:(B \epsilon A) \wedge (X \subset B)\}.$$

17) CARDINALITY of A

#A = n iff there are exactly n elements in A.

18) A is a *SUBSET* of B iff every element of A is an element of B: $CBB \leftrightarrow (\forall x)(x \epsilon A \rightarrow x \epsilon B)$.

19) A is *EQUAL* to B iff A is a subset of B, and B is a subset of A: $A=B \leftrightarrow (ACB \& BCA)$.

20) A and B are *DISJOINT* iff the intersection of A and B is empty: $A \cap B = \emptyset$.

21) A is *EQUIVALENT* to B iff A and B contain the same number of elements: #A = #B.

GLOSSARY OF SYMBOLS

Symbol	Symbol Definition
iff	if and only if
=	Identity
\wedge	Conjunction
\vee	Disjunction
Δ	Exclusive or
\rightarrow	Implication (if . . . then)
\leftrightarrow	Equivalence
$\forall x$	Universal quantifier (for all)
$\exists x$	Existential quantifies (for some)
$E!x$	Uniqueness quantifier (for exactly one)
Θx	Odd quantifier (for an odd number of)
$(E_n)!x$	Exact number quantifier
ϵ	Set membership
ϕ	Empty set
\notin	Non-membership
\subset	Set inclusion

$A \cap B$	Intersection	$A \times B$	Cartesian product
$A \cup B$	Union	$\mathfrak{D}(A)$	Domain of A
$A \Delta B$	Symmetric difference	$\mathfrak{R}(A)$	Range of A
$A \sim B$	Relative complement	\bar{A}	Converse of A
$\langle x,y \rangle$	Ordered pair	A/B	Relative product of A and B
$\{x:\theta(x)\}$	Definition by abstraction	$A X$	A restricted to X
xAy	Ordered pair $\langle x,y \rangle$ contained in A	$A[X]$	Image of X under A
UG	Union or sum of G	$[X]A$	Converse-image of X under A
$\cap G$	Intersection of G	$\mathfrak{D}(X)$	Domain-concurrence of X
ΔG	Symmetric difference of G	$\mathfrak{R}(X)$	Range-concurrence of X
$E_n G$	Elements contained in exactly n elements of G	$\mathfrak{G}(X)$	Set-concurrence of X
		$\xi(A)$	Total cardinality of A

The last column contains an executable expression of the set-theoretic expression preceding it. D is an indirect name for the permanent storage with name C, or for temporary storage if the argument C is deleted, (see text).

1) UNION	$C = A \cup B$	$D = UN(A, B, C)$
	$C = UA$	$D = UN(1, A, C)$
2) INTERSECTION	$C = A \cap B$	$D = IN(A, B, C)$
	$C = \cap A$	$D = IN(1, A, C)$
3) SYMMETRIC DIFFERENCE	$C = A \Delta B$	$D = SD(A, B, C)$
	$C = \Delta A$	$D = SD(1, A, C)$
4) RELATIVE COMPLEMENT	$C = A \sim B$	$D = RL(A, B, C)$
5) EXACTLY N ELEMENTS OF A	$C = E_n A$	$D = EX(N, A, C)$
6) DOMAIN of A	$C = \mathfrak{D}(A)$	$D = DM(A, C)$
7) RANGE of A	$C = \mathfrak{R}(A)$	$D = RG(A, C)$
8) IMAGE of B under A	$C = A[B]$	$D = IM(A, B, C)$
9) CONVERSE IMAGE under A	$C = [B]A$	$D = CM(A, B, C)$
10) CONVERSE of A	$C = \bar{A}$	$D = CV(A, C)$
11) RESTRICTION of A to B	$C = A B$	$D = RS(A, B, C)$
12) RELATIVE PRODUCT of A and B	$C = A/B$	$D = RP(A, B, C)$
13) CARTESIAN PRODUCT of A and B	$C = A \times B$	$D = XP(A, B, C)$
14) DOMAIN CONCURRENCE of A to B	$C = \mathfrak{D}(A:B)$	$D = DC(A, B, C)$
15) RANGE CONCURRENCE of A to B	$C = \mathfrak{R}(A:B)$	$D = RC(A, B, C)$
16) SET CONCURRENCE of A to B	$C = \mathfrak{G}(A:B)$	$D = SC(A, B, C)$
17) CARDINALITY of A $N = \#A, (N \text{ is an integer})$		$N = C(A)$
<p>BOOLEAN OPERATIONS I = 1 if the statement is true. I = 0 if the statement is false.</p>		
18) A is a subset of B		I = SBS(A, B)
19) A is equal to B		I = EQL(A, B)
20) A and B are disjoint		I = DSJ(A, B)
21) A is equipollent to B		I = EQP(A, B)
22) A is an element of B		I = ELM(A, B)

SPECIAL CONTROL OPERATIONS

23) SET CONSTRUCTION	$C = \{A, B, X, \dots\}$	$D = S(C, A, B, X, \dots)$
24) MODE of A (see text) N is an integer		$N = M(A)$
25) ACCESS DATA in A by format N		$D = ACC(N, A, C)$

(each format is written in the parent language and given an integer name)

TABLE 1—Some set operations expressed as subroutines

REFERENCES

1 D L CHILDS
Feasibility of a set-theoretic data structure—a general structure based on a reconstituted definition of relation
IFIP Congress 68

2 T E JOHNSON
A mass storage relational data structure for computer graphics and

other arbitrary data stores
MIT Department of Architecture Report October 1967

3 T SKOLEM
Two remarks on set theory
MATH SCAND 5 43–46 1957

4 P SUPPES
Axiomatic set theory
Van Nostrand Princeton 1960

Application of functional optimization techniques for the serial hybrid computer solution of partial differential equations*

by HIROSHI H. HARA

Lockheed-California Co.
Burbank, California

and

WALTER J. KARPLUS

University of California at Los Angeles
Los Angeles, California

INTRODUCTION

Since its introduction in 1961, the serial or CSDT (continuous-space-discrete-time) method for solving nonlinear parabolic partial differential equations in one space-dimension has received a considerable amount of attention.¹⁻⁸ This effort has been justified by the engineering importance and the abundance of problems characterized by one-dimensional diffusion equations. Although initially introduced as a pure analog technique, interest in the CSDT method was stimulated particularly by the increasing availability and capabilities of hybrid computing systems. Utilizing the hybrid computer, a closed loop of analog elements is employed to integrate a second-order ordinary differential equation at successive time levels. The digital computer serves as a function memory and to control the iterative determination of an initial condition at each time level.

The major advantages of the CSDT approach over the more conventional DSCT (discrete-space-continuous-time) approach include the following:

- 1) The analog hardware requirements of the CSDT method are very small compared to those of the DSCT method. This consideration is particularly important in the treatment of equations with non-

linear and time-varying parameters, for in this case a DSCT simulation requires separate nonlinear function generators at each finite-difference grid point in the space domain.

- 2) The high-speed computational capabilities of modern iterative analog computers can be utilized to full advantage.
- 3) Problems involving moving boundaries can be solved readily by controlling the analog computer's integration interval, since the problem space variable is represented by the computer time variable.

In practical applications, however, considerable difficulty is encountered in obtaining dependable results using the CSDT method. The reason for this lies in the fact that in addition to the errors normally encountered in hybrid simulations (errors due to truncation and analog inaccuracies), the mechanization of the CSDT technique requires the instrumentation on the analog computer of an inherently-unstable ordinary differential equation. This in turn causes errors, which normally can be tolerated in hybrid work, to accumulate and grow excessively as the solution progresses, leading to highly unreliable results.

It is the purpose of the present paper to introduce a method which retains the advantageous features of the serial CSDT method but which involves the solution on the analog computer of a stable rather than an unstable differential equation. In the following section the

*Research in hybrid computation in the Department of Engineering, University of California, Los Angeles is supported by the National Science Foundation under grant GK-1758.

classical CSDT method is briefly reviewed in order to provide a point of departure for the description of the new method. The third section is devoted to a general description of the new technique, while the computational algorithms handled on the digital computer are presented in the fourth section. The last section includes general comments on the new method as well as a description of the extension of the method to parabolic differential equations in two space-dimensions.

The classical CSDT method

We consider a linear one-dimensional diffusion equation

$$\frac{\partial^2 u(x, t)}{\partial x^2} = \frac{1}{\alpha} \frac{\partial u(x, t)}{\partial t} \tag{2.1}$$

where

$$x \in (0, x') \text{ and } t \in (0, t')$$

with the boundary conditions

$$u(x = 0, t) = u_0(t) \tag{2.2}$$

$$u(x = x', t) = u_f(t) \tag{2.3}$$

and the initial condition

$$u(x, t = 0) = u^0(x) . \tag{2.4}$$

Applying the classical CSDT approximation to equation (2.1),

$$\frac{d^2 u^i}{dx^2} \cong \frac{u^i - u^{i-1}}{\alpha \Delta t}, \quad i = 1, \dots, N \tag{2.5}$$

where

$$u^i = u(x, t^i)$$

$$t^i = i \Delta t$$

$$\Delta t = \frac{t'}{N}$$

Equations (2.2) and (2.3) are, respectively, transformed into

$$u^i(x = 0) = u_0^i = u_0(t^i) \tag{2.6}$$

and

$$u^i(x = x') = u_f^i = u_f(t^i), \tag{2.7}$$

while equation (2.4) is transformed into

$$u^{i=0} = u^0(x) . \tag{2.8}$$

Since equation (2.5) is now an ordinary differential equation, an analog computer can be utilized for its solution. That is, the independent variable x can be represented by the analog computer-time variable and therefore integrations can be carried out in a continuous manner. In solving for u^i , the solution u^{i-1} of the previous step in time acts as a driving function. The generated function u^i then must be "remembered" so as to be used as the driving function for the next time step.

Furthermore, we now need to solve a two-point boundary value problem because the boundary condition (2.3) in the original problem has been converted to the final value of u^i . That is, while the boundary condition (2.2) is used as one of the initial conditions (i.e., $u^i(x = 0)$), the other initial condition, $\frac{du^i}{dx}(x = 0)$, which is unknown must be found such that (2.3) is satisfied at the end of the computer run. This is usually accomplished by an iterative technique.

A typical hybrid computer mechanization for the implementation of classical CSDT method is shown in Figure 1. Briefly, the computational steps are:

- 1) Initialize the analog integrators with one known initial condition u_0^i and a trial initial condition $\frac{du^i}{dx}(x = 0)$.
- 2) Place the analog subsection in the COMPUTE mode during which the solution u^{i-1} obtained during the preceding solution cycle is played back via a DAL (digital-to-analog linkage), and the new solution u^i is recorded via an ADL (analog-to-

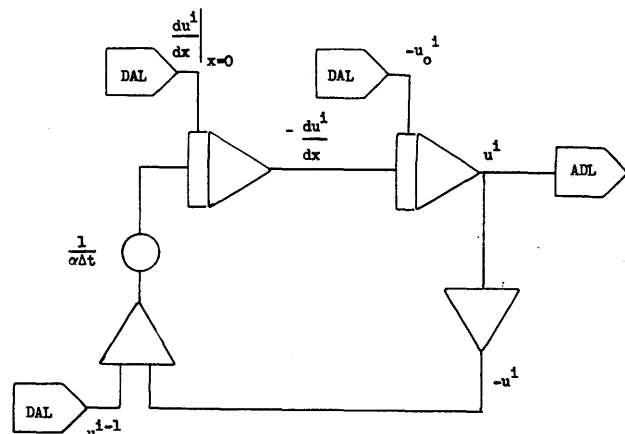


FIGURE 1—Hybrid computer mechanization of the classical CSDT method

digital linkage). The ADL usually includes a sample-hold amplifier which is connected to an analog-to-digital converter through a multiplexor switch. The DAL can include a hybrid interpolator which transforms discrete staircase functions appearing at digital-to-analog converters into smooth functions.

- 3) At the end of the run corresponding to $x = x'$, compare the final value of u^i to the known boundary value u_f^i . If the difference is within a specified error bound, increment i and go back to step 1. If not, improve the trial initial condition and repeat the above process.

As can be seen from Figure 1, the analog computer circuit contains a closed loop comprised of four operational amplifiers. Such a circuit is inherently unstable and can be expected to exhibit marked sensitivities to errors in initial conditions and component inaccuracies. The loop gain is seen to be inversely proportional to Δt ; hence the smaller Δt the more pronounced the instability of the analog loop. It is therefore not feasible with a mechanization of this type to reduce the truncation error inherent in the finite difference approximation of $\frac{\partial u}{\partial t}$ by reducing Δt . The loop-gain is also affected by the time scale factor employed in the analog loop, so that the faster the solution speed the larger the instability. It should be noted that the reversal of the computing direction (i.e., backward computation) still keeps the unstable loop intact. These considerations have severely limited the applicability of the serial CSDT method.

A new CSDT method

It is the objective of the present method to retain the advantageous features of the serial CSDT method while obviating the need for an even number of operational amplifiers in the analog loop. The method is applicable to nonlinear as well as to linear partial differential equations. For simplicity in exposition the discussion below makes reference to a linear parabolic partial differential equation in one space-dimension. It should be recognized, however, that most practical applications will involve equations with nonlinear and time-varying parameters. The key step in the derivation of this method involves the definition of a control function, which is generated digitally and imposed as a forcing function upon the analog circuit. Denoting this control function as \hat{u}^i and introducing this term in equation (2.5) results in

$$\frac{d^2 u^i}{dx^2} = \frac{1}{\alpha \Delta t} (2\hat{u}^i - u^i - u^{i-1}). \quad (3.1)$$

If $\hat{u}^i(x)$ can be found such that

$$\hat{u}^i(x) = u^i(x) \text{ for all } x \in (0, x'), \quad (3.2)$$

equation (3.1) reduces to equation (2.5). Note that u^i is the only function for which to solve—with \hat{u}^i and u^{i-1} appearing as external driving functions. Of course, we must find or "optimize" one of the external functions, namely the control $\hat{u}^i(x)$ such that the equality (3.2) is satisfied.

We can now re-formulate our problem as that of the optimal control: find a control policy \hat{u}^i in order to minimize (in our case, we would like to make it zero) the criterion function θ which is defined by

$$\theta = \int_{x=0}^{x'} (\epsilon^i)^2 dx \quad (3.3)$$

where

$$\epsilon^i = \hat{u}^i - u^i \quad (3.4)$$

subject to the constraints

$$\frac{d^2 u^i}{dx^2} = \frac{1}{\alpha \Delta t} (2\hat{u}^i - u^i - u^{i-1})$$

with initial conditions

$$u^i(x=0) = u_0^i \quad (3.5)$$

$$\frac{du^i}{dx}(x=0) = u_{x_0}^i \quad (3.6)$$

and also satisfying the terminal constraint equation

$$\Psi(u^i) = u^i(x=x') - u_f^i = 0. \quad (3.7)$$

Note that the initial condition in (3.6) is still unknown, but it turns out that $u_{x_0}^i$ is automatically determined once a control policy \hat{u}^i is selected.

In order to simplify subsequent derivations, we introduce new notation:

$$\begin{aligned} u_1^i &= u^i \\ u_2^i &= \frac{du^i}{dx} = \frac{du_1^i}{dx} \end{aligned} \quad (3.8)$$

$$\frac{du_3^i}{dx} = (\epsilon^i)^2 \quad (\text{i.e., } u_3(x=x') = \theta).$$

Equations (3.1) through (3.9) now become

$$\begin{aligned} \frac{du_1^i}{dx} &= u_2^i \\ \frac{du_2^i}{dx} &= \frac{1}{\alpha \Delta t} (2\hat{u}^i - u_1^i - u^{i-1}) \\ \frac{du_3^i}{dx} &= (\epsilon^i)^2. \end{aligned} \quad (3.9)$$

with initial conditions

$$\begin{aligned} u_1^i(x=0) &= u_0^i \\ u_2^i(x=0) &= u_{20}^i \\ u_3^i(x=0) &= 0 \end{aligned} \quad (3.10)$$

and

$$\epsilon^i = \hat{u}^i - u_1^i \quad (3.11)$$

$$\theta = u_3^i(x=x') \quad (3.12)$$

$$\Psi = u_1^i(x=x') - u_f^i = 0. \quad (3.13)$$

In matrix form, (3.9) and (3.10) become, respectively

$$\frac{dU^i}{dx} = F^i(x, U^i, u^{i-1}, \hat{u}^i) \quad (3.14)$$

$$U^i(x=0) = U_0^i \quad (3.15)$$

where

$$U^i = \begin{bmatrix} u_1^i \\ u_2^i \\ u_3^i \end{bmatrix}$$

and

$$F^i = \begin{bmatrix} f_1^i \\ f_2^i \\ f_3^i \end{bmatrix} = \begin{bmatrix} u_2^i \\ \frac{1}{\alpha \Delta t} (2\hat{u}^i - u_1^i - u^{i-1}) \\ (\epsilon^i)^2 \end{bmatrix} \quad (3.16)$$

The gradient method or the steepest-descent method⁹ is now applied.

We consider the system equation (3.14) and the corresponding perturbation equation

$$\frac{d}{dx} (\delta U^i) = \frac{\partial F^i}{\partial U^i} \delta U^i + \frac{\partial F^i}{\partial \mu^i} \delta \hat{u}^i \quad (3.17)$$

where the partial derivatives are evaluated along the nominal trajectory and

$$\frac{\partial F}{\partial U} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \frac{\partial f_1}{\partial u_3} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \frac{\partial f_2}{\partial u_3} \\ \frac{\partial f_3}{\partial u_1} & \frac{\partial f_3}{\partial u_2} & \frac{\partial f_3}{\partial u_3} \end{bmatrix} \quad (3.18)$$

The adjoint equation to (3.17) is defined to be

$$\frac{d\lambda^i}{dx} = - \left[\frac{\partial F^i}{\partial U^i} \right]^T \lambda^i \quad (3.19)$$

where the superscript T denoting the transpose of the matrix and $\lambda^i = [\lambda_1^i \lambda_2^i \lambda_3^i]^T$. If equation (3.17) is pre-multiplied by λ^T and the transpose of equation (3.19) post-multiplied by δU , and the sum is integrated over the interval $0 \leq x \leq x'$, we obtain

$$\lambda^{iT} \delta U^i \Big|_{x=0}^{x=x'} = \int_{x=0}^{x'} \lambda^{iT}(x) G^i(x) \delta \hat{u}^i(x) dx \quad (3.20)$$

where

$$G(x) = \frac{\partial F}{\partial \mu} = \left[\frac{\partial f_1}{\partial \hat{u}} \frac{\partial f_2}{\partial \hat{u}} \frac{\partial f_3}{\partial \hat{u}} \right]^T \quad (3.21)$$

Since we are still free to choose λ , we let

$$\lambda^T(x=x') = \frac{\partial \theta}{\partial U}(x=x') = \frac{\partial u_3}{\partial U}(x=x'). \quad (3.22)$$

Then, $\lambda^T \delta U(x=x') = \delta \theta(x=x')$ and equation (3.20) becomes

$$\begin{aligned} \delta \theta^i &= \lambda^{iT} \delta U^i(x=0) \\ &+ \int_{x=0}^{x'} \lambda^{iT}(x) G^i(x) \delta \hat{u}^i(x) dx. \end{aligned} \quad (3.23)$$

The λ^i 's are the influence functions or Green's functions which give the effect of small changes in the control function \hat{u}^i and the initial state variables, on the

criterion function θ^i . The influence functions are obtained by equation (3.19) with the boundary (terminal) condition (3.22). In view of our objective to minimize θ (i.e., to make $\delta\theta$ as negative as possible), $\delta\hat{u}^i$ should be chosen such as to make the integral in (3.23) always negative regardless of $\delta U^i(x=0)$. This can be realized if we choose

$$\delta\hat{u}^i = -K_\phi G^{iT} \lambda^i \quad (3.24)$$

where K_ϕ is a positive constant to be determined later. This is the "steepest descent" direction to the minimum θ^i .

In order to satisfy the constraint equation (3.13) we further need to obtain another set of influence functions, say ξ^i , which will indicate the effects of $\delta\hat{u}^i(x)$ as well as $\delta U^i(x=0)$ upon the constraint function Ψ . Following an identical derivation as above, we obtain

$$\begin{aligned} \delta\Psi^i(x=x') &= \xi^{iT} \delta U^i(x=0) \\ &+ \int_{x=0}^{x'} \xi^{iT}(x) G^i(x) \delta\hat{u}^i(x) dx \end{aligned} \quad (3.25)$$

where ξ^i is the solution of

$$\frac{d\xi^i}{dx} = - \left[\frac{\partial F^i}{\partial U^i} \right]^T \xi^i \quad (3.26)$$

with the boundary (terminal) condition

$$\xi^{iT}(x=x') = \frac{\partial\Psi}{\partial U}(x=x'). \quad (3.27)$$

Now, if the nominal trajectory approximately satisfies the constraint (3.13), then by setting $\delta\Psi^i$ equal to the negative of $\Psi^i(x=x')$, the next trajectory should satisfy $\Psi = 0$. The steepest descent direction to minimize θ and satisfy a given $\delta\Psi$ is

$$\delta\hat{u}^i(x) = - (K_\phi G^{iT} \lambda^i + K_\Psi G^{iT} \xi^i) \quad (3.28)$$

where K_Ψ is another constant to be determined.

In equation (3.10), while two of the initial conditions are fixed, we still are faced with the problem of determining the correct initial condition on u_2^i . Fortunately, both equations (3.23) and (3.25) also relate the effect of the change in the initial values of the state variable upon the criterion function and the constraint function, respectively. Furthermore, the choice of $\delta U^i(x=0)$ is directly dictated by the choice $\delta\hat{u}^i$ as shown below. From equation (3.15),

$$\delta U^i(x=0) = [0 \ \delta u_2^i \ 0]^T_{x=0} \quad (3.29)$$

where

$$\delta u_2^i(x=0) = \delta \left[\frac{du_1^i}{dx} \right]_{x=0} \quad (3.30)$$

Since we are seeking \hat{u}^i such that $\hat{u}^i = u_1^i$, equation (3.30) can be written as

$$\delta u_2^i(x=0) = \delta \left[\frac{d\hat{u}^i}{dx} \right]_{x=0} \quad (3.31)$$

Due to the fact that \hat{u}^i is to be updated by $\delta\hat{u}^i$, we can rewrite (3.31) as

$$\delta u_2^i(x=0) = \left[\frac{d}{dx} (\delta\hat{u}^i) \right]_{x=0} \quad (3.32)$$

Therefore, once $\delta\hat{u}^i$ is chosen according to (3.28), $\delta U^i(x=0)$ in equations (3.23) and (3.25) is automatically determined. Combining of equations (3.23), (3.25), (3.28), (3.29), and (3.32) yields

$$\begin{aligned} \delta\theta(x=x') &= - [\lambda_2^i \frac{d}{dx} (K_\phi G^{iT} \lambda^i \\ &+ K_\Psi G^{iT} \xi^i)]_{x=0} - \int_{x=0}^{x'} \lambda^{iT} G^i (K_\phi G^{iT} \lambda^i \\ &+ K_\Psi G^{iT} \xi^i) dx \end{aligned} \quad (3.33)$$

and

$$\begin{aligned} \delta\Psi(x=x') &= - [\xi_2^i \frac{d}{dx} (K_\phi G^{iT} \lambda^i \\ &+ K_\Psi G^{iT} \xi^i)]_{x=0} \end{aligned} \quad (3.34)$$

$$- \int_{x=0}^{x'} \lambda^{iT} G^i (K_\phi G^{iT} \lambda^i + K_\Psi G^{iT} \xi^i) dx.$$

For desired changes of $\delta\theta(x=x') = \Delta\theta$ and $\delta\Psi(x=x') = \Delta\Psi$, we can solve linear equations (3.33) and (3.34) for K_ϕ and K_Ψ and obtain

$$K_\phi = \frac{Q_{22}\Delta\theta - Q_{12}\Delta\Psi}{Q_{12}Q_{21} - Q_{11}Q_{22}} \quad (3.35)$$

$$K\psi = \frac{Q_{11}\Delta\Psi - Q_{21}\Delta\theta}{Q_{12}Q_{21} - Q_{11}Q_{22}} \quad (3.36)$$

where

$$Q_{11} = [\lambda_2^i \frac{d}{dx} (G^{iT} \lambda^i)]_{x=0} + \int_{x=0}^{x'} \lambda^{iT} G^i G^{iT} \lambda^i dx$$

$$Q_{12} = [\lambda_2^i \frac{d}{dx} (G^{iT} \xi^i)]_{x=0} + \int_{x=0}^{x'} \lambda^{iT} G^i G^{iT} \xi^i dx \quad (3.37)$$

$$Q_{21} = [\xi_2^i \frac{d}{dx} (G^{iT} \lambda^i)]_{x=0} + \int_{x=0}^{x'} \xi^{iT} G^i G^{iT} \lambda^i dx$$

$$Q_{22} = [\xi_2^i \frac{d}{dx} (G^{iT} \xi^i)]_{x=0} + \int_{x=0}^{x'} \xi^{iT} G^i G^{iT} \xi^i dx$$

Computational algorithm of the new CSDT method

In the previous section, basic equations necessary to perform automatic updating of the control policy were derived, but some of the equations were left in a general form. In this section, the key equations will be first written in more specific terms, and then the hybrid computational algorithm will be summarized.

The state equations to be solved are

$$\frac{du_1^i}{dx} = u_2^i$$

$$\frac{du_2^i}{dx} = \frac{1}{\alpha\Delta t} (2\hat{u}^i - u_1^i - u^{i-1}) \quad (4.1)$$

$$\frac{du_3^i}{dx} = \epsilon_i^2$$

with the initial conditions

$$u_1^i(x=0) = u_0^i$$

$$u_2^i(x=0) = \frac{d\hat{u}^i}{dx}(x=0) \quad (4.2)$$

$$u_3^i(x=0) = 0$$

where

$$\epsilon^i = \hat{u}^i - u_1^i \quad (4.3)$$

and \hat{u}^i is the nominal control.

The two sets of adjoint equations to be solved are

$$\frac{d\lambda_1^i}{dx} = \frac{1}{\alpha\Delta t} \lambda_2^i + 2\epsilon^i \quad (4.4)$$

$$\frac{d\lambda_2^i}{dx} = -\lambda_1^i$$

with the terminal conditions

$$\lambda_1^i(x=x') = \lambda_2^i(x=x') = 0 \quad (4.5)$$

as obtained from equations (3.19) and (3.22), and

$$\frac{d\xi_1^i}{dx} = \frac{1}{\alpha\Delta t} \xi_2^i \quad (4.6)$$

$$\frac{d\xi_2^i}{dx} = -\xi_1^i$$

with the terminal conditions

$$\xi_1^i(x=x') = 1, \quad \xi_2^i(x=x') = 0 \quad (4.7)$$

as obtained from equations (3.26) and (3.27).

It can be shown from (3.28) that the updating equation for the control policy is

$$\delta \hat{u}^i = - [K_\phi \left(\frac{2}{\alpha\Delta t} \lambda_2^i + 2\epsilon^i \right) + K_\psi \left(\frac{2}{\alpha\Delta t} \xi_2^i \right)] \quad (4.8)$$

where K_ϕ and K_ψ are given by equations (3.35) and

(3.36). Furthermore the equations in (3.37) simplify to

$$\begin{aligned}
 Q_{11} &= [\lambda_2^i \frac{d}{dx} (\frac{2}{\alpha \Delta t} \lambda_2^i + 2 \epsilon^i)]_{x=0} \\
 &\quad + \int_{x=0}^{x'} \left(\frac{2}{\alpha \Delta t} \lambda_2^i + 2 \epsilon^i \right)^2 dx \\
 Q_{12} &= - [\lambda_2^i \frac{2}{\alpha \Delta t} \xi_1^i]_{x=0} \\
 &\quad + \int_{x=0}^{x'} \left(\frac{2}{\alpha \Delta t} \lambda_2^i + 2 \epsilon^i \right) \left(\frac{2}{\alpha \Delta t} \xi_2^i \right) dx
 \end{aligned} \tag{4.9}$$

$$\begin{aligned}
 Q_{21} &= [\xi_2^i dx \left(\frac{2}{\alpha \Delta t} \lambda_2^i + 2 \epsilon^i \right)]_{x=0} \\
 &\quad + \int_{x=0}^{x'} \left(\frac{2}{\alpha \Delta t} \lambda_2^i + 2 \epsilon^i \right) \left(\frac{2}{\alpha \Delta t} \xi_2^i \right) dx
 \end{aligned}$$

$$\begin{aligned}
 Q_{22} &= - [\xi_2^i \frac{2}{\alpha \Delta t} \xi_1^i]_{x=0} \\
 &\quad + \int_{x=0}^{x'} \left(\frac{2}{\alpha \Delta t} \xi_2^i \right)^2 dx .
 \end{aligned}$$

Actual updating of the control policy is accomplished by

$$\hat{u}^i (\text{NEW}) = \hat{u}^i (\text{OLD}) + \delta \hat{u}^i . \tag{4.10}$$

The initial estimate of the control policy can be obtained as follows. If we apply the forward CSDT approximation to equation (2.1) we obtain

$$\frac{d^2 u^i}{dx^2} = \frac{1}{\alpha \Delta t} (u^{i+1} - u^i) \tag{4.11}$$

Equating this equation with (2.5) and simplifying,

$$u^{i+1} = 2u^i - u^{i-1} . \tag{4.12}$$

This equation provides a rough estimate on the state function for the next time increment. Therefore, it is reasonable to use this equation to compute the first nominal control policy as

$$\hat{u}^{i+1} = 2u^i - u^{i-1} = 2u^i - u^{i-1} \tag{4.13}$$

The overall hybrid computational algorithm then becomes:

1) With a nominal control policy \hat{u}^i , solve equation

(4.1) on the analog subsection of the hybrid computer with the initial conditions (4.2). Both \hat{u}^i and u^{i-1} are played back via DAL's from the digital subsection into the analog subsection. Meanwhile, read ϵ^i into the digital subsection via an ADL.

At the end of the run corresponding to $x = x'$,
 2) record the values of u_3^i (i.e., θ) and u_1^i . These values are used digitally to compute $\Delta\theta$ and $\Delta\Psi$ to be used in step 5. ($\Delta\theta = -\beta u_3^i(x')$, $\Delta\Psi = -\beta\Psi^i$ where $0 < \beta \leq 1$; the more non-linear the problem the smaller the value of β). If a convergence has been attained (e.g., $\text{Max} |\epsilon_i| < \text{specified bound}$), go to step 6.

3) Next, solve the adjoint equations (4.4) and (4.6) backward in analog computer time, t_c , (where $t_c = x' - x$) since we are given the terminal conditions instead of the usual initial conditions. Note that this process changes the signs of the lefthand sides of these equations. ϵ^i which was recorded in step 1 is played back via a DAL in the reverse direction. The terms required by equation (4.8) are recorded via ADL's. At the same time compute in the analog subsection the values of the integrals which appear in equations (4.9).

4) At the end of the adjoint run, record the final values of λ_2^i , ξ_1^i , and ξ_2^i which actually are $\lambda_2^i(x=0)$, $\xi_1^i(x=0)$, and $\xi_2^i(x=0)$, respectively.

5) Digitally compute Q_{11} , Q_{12} , Q_{21} , and Q_{22} . The derivatives which need to be evaluated in (4.9) at $x = 0$ can be digitally approximated. Using these values and $\Delta\theta$ and $\Delta\Psi$ which were computed in step 2, obtain K_θ and K_Ψ according to (3.35) and (3.36). Update the control policy according to (4.8) and (4.10), and go to step 1.

6) Compute a new nominal control policy by equation (4.13). (For the very initial prediction of u^{i+1} where $i = 0$, $u^1 \cong u^0(x)$ can be used.) Increment i by one and go to step 1.

7) Repeat steps 1 through 6 until $i = N$. A typical hybrid mechanization diagram for the new method is shown in Figure 2.

Discussion of the new method and its extension

With the application of the functional optimization technique, the unstable analog loop in the classical CSDT method has been replaced by a stable loop. Although the parameter (initial condition) optimization problem involved in the classical method has been replaced by a more complicated functional optimization problem in the new method, the high-speed computational capabilities of the hybrid computer make this

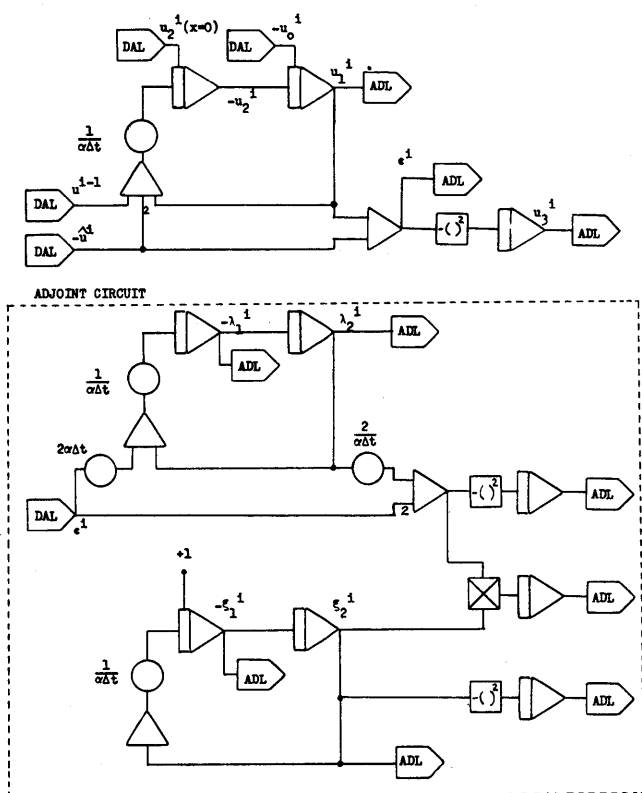


FIGURE 2—Hybrid computer mechanization of the new CSDT method

method practical. Furthermore, with removal of the unstable loop, it is now possible to reduce the size of Δt in order to minimize the truncation error of the CSDT approximation.

A similar technique can be applied to the CSDT version of the well-known Crank-Nicholson numerical approximation

$$\frac{1}{2} \left[\frac{d^2 u^i}{dx^2} + \frac{d^2 u^{i-1}}{dx^2} \right] = \frac{u^i - u^{i-1}}{\alpha \Delta t} \quad (5.1)$$

In order to avoid the explicit computation of $\frac{d^2 u^{i-1}}{dx^2}$ as well as the unstable computational loop, introduce a new variable w^i defined by

$$w^i = \frac{1}{2} (u^i + u^{i-1}) \quad (5.2)$$

from which

$$u^{i-1} = 2w^i - u^i \quad (5.3)$$

Combining of (5.1), (5.2), and (5.3) yields

$$\frac{d^2 w^i}{dx^2} = \frac{2}{\alpha \Delta t} (u^i - w^i) \quad (5.4)$$

Equation (5.4) is now a stable equation in state variable w^i , and u^i becomes the external control policy to be optimized. The criterion function θ to be minimized becomes (from 5.3)

$$\theta = \int_{x=0}^L [(2w^i - u^i) - u^{i-1}]^2 dx \quad (5.5)$$

The new CSDT and Peaceman-Rachford ("alternating direction implicit")¹⁰ methods can be combined to solve two dimensional diffusion equations. As an illustration, the procedure is explained for the simple constant diffusivity case:

$$\frac{\partial^2 u(x, y, t)}{\partial x^2} + \frac{\partial^2 u(x, y, t)}{\partial y^2} = \frac{1}{\alpha} \frac{\partial u(x, y, t)}{\partial t} \quad (5.6)$$

where $x\epsilon(0, x')$, $y\epsilon(0, y')$, $t\epsilon(0, t')$, with boundary and initial conditions

$$\begin{aligned} u(0, y, t) &= u_a(y, t) \\ u(x', y, t) &= u_b(y, t) \\ u(x, 0, t) &= u_c(x, t) \\ u(x, y', t) &= u_d(x, t) \\ u(x, y, 0) &= u^0(x, y) \end{aligned}$$

The basic process in passing from t^i to t^{i+1} has two parts; each part involving the solution of a set of ordinary differential equations by the CSDT technique.

We will need to find u and $\frac{\partial^2 u}{\partial x^2}$ along the lines $y = y_m$, $m = 1, \dots, M-1$, for the first part (note that $y = 0$, $y = y_M$ are boundaries). The second part will consist of finding u and $\frac{\partial^2 u}{\partial y^2}$ along the lines $x = x_\ell$, $\ell = 1, \dots, L-1$. At the end of the first part we will have passed to $t^{i+1/2}$. Thus, u will be defined on a $(M + 1) \times (L + 1)$ grid at t^{i+1} (the analog solutions are digitized only at these discrete points). During the process, there will be similar tables of $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$.

With the foregoing in mind, we proceed to explain the procedure. At t^i , u and $\frac{\partial^2 u}{\partial y^2}$ are defined on the $(M + 1) \times (L + 1)$ grid in the interior of the region $x\epsilon(0, x')$, $y\epsilon(0, y')$. First, we find the set of functions

$$u_m^{i+1/2}(x) = u(x, y_m, t^{i+1/2})$$

on the lines $y = y_m$ from the equations

$$\frac{d^2 u_m^{i+1}(x)}{dx^2} + \frac{d^2 u_m^i(x)}{dy^2} = \frac{u_m^{i+1}(x) - u_m^i(x)}{\alpha \frac{\Delta t}{2}},$$

$$m = 1, \dots, M-1 \quad (5.7)$$

with analog integration. Because y is fixed, the spacial derivatives of (5.6) reduce to total derivatives. The term $\frac{d^2 u_m^i(x)}{dy^2}$ is formed by interpolation from the previously stored $\frac{d^2 u}{dy^2}$ data on the $(M+1) \times (L+1)$ grid at t^i .

The functional optimization method (as previously described) is used to solve equations (5.7) from

$$\frac{d^2 u_m^{i+1}(x)}{dx^2} = \frac{2\hat{u}_m^{i+1}(x) - u_m^{i+1}(x) - u_m^i(x)}{\alpha \frac{\Delta t}{2}} - \frac{d^2 u_m^i(x)}{dy^2} \quad (5.8)$$

where the $\hat{u}_m^{i+1}(x)$ are the external control policies to be optimized (i.e., \hat{u}_m^{i+1} converges to u_m^{i+1}). When equations (5.8) have been solved, $u_m^{i+1}(x)$ and $\frac{d^2 u_m^{i+1}(x)}{dx^2}$ (which is also available from the analog program) are digitized and stored over the grid at t^{i+1} , and the first part is complete. The second part of the procedure consists of finding the set of functions

$$u_\ell^{i+1}(y) = u(x_\ell, y, t^{i+1})$$

on the lines $x = x_\ell$ from the equations

$$\frac{d^2 u_\ell^{i+1}(y)}{dx^2} + \frac{d^2 u_\ell^{i+1}(y)}{dy^2} = \frac{u_\ell^{i+1}(y) - u_\ell^{i+1}(y)_x}{\alpha \frac{\Delta t}{2}},$$

$$\ell = 1, \dots, L-1.$$

As before, the term $\frac{d^2 u_\ell^{i+1}(y)}{dx^2}$ is obtained by interpolation of the $\frac{d^2 u_m^{i+1}(x)}{dx^2}$ values stored on the grid at

t^{i+1} . These equations are solved with the same functional optimization procedure. The solutions $u^{i+1}(y)$ and $\frac{d^2 u^{i+1}(y)}{dy^2}$ are digitized and stored over the grid at t^{i+1} . This completes the second part and also one step of the CSDT procedure.

The values of u over the grid for successive t^i , $i = 1, 2, \dots$ are the solution to the problem defined by (5.6).

A few words regarding the details of generating certain of the second derivatives at grid boundaries may be helpful. The values of $u_m^{i+1}(x)$ for $m=0, M$ are automatically available as boundary values. But, the values of the second derivatives $\frac{d^2 u_m^{i+1}(x)}{dx^2}$ for $m=0, M$ must be approximated digitally for each x_ℓ . This is necessary because during the subsequent half-step (i.e., proceeding from t^{i+1} to t^{i+2}) they are both required for external driving functions. The same requirement is imposed on the second derivatives $\frac{d^2 u_\ell^{i+1}(y)}{dy^2}$ for $\ell = 0, L$. Of course, the second derivatives required to start the process at $t = 0$ can be calculated from $u^0(x, y)$.

REFERENCES

- 1 SH JURY
Solving partial differential equations
Industrial and Engineering Chemistry Vol 53 pp 177-180 1961
- 2 D R ELLIS
Analog solution of the diffusion equation with continuous space and discrete time variables
MS Thesis Dept of Elec. Engr MIT Aug 1963
- 3 R MORONEY
Hybrid computer solution of parabolic partial differential equations via the continuous space discrete time method
MIT Electronic Systems Lab DSR 9128-wp-16 Jan 1964
- 4 H S WITSENHAUSEN
Hybrid solution of initial value problems for partial differential equations
MIT Electronic Systems Lab DSR 9128 Memo No 8 Aug 1964
- 5 R BENZ
A hybrid computer solution of certain field problems
Boeing Company Internal Memo Jan 1965
- 6 SHU-KWAN CHAN
A study on certain hybrid techniques for the solution of the heat equation
MIT Electronics Systems Lab DSR 9128 Memo No 11 Dec 1965
- 7 I M MANLEY
Hybrid computer solution of the diffusion equation with continuous space and discrete time variables using sensitivity equations
MS Thesis Dept of Engr Univ of California at Los Angeles 1966
- 8 R VICHNEVETSKY
A new stable computing method for the serial hybrid computer

integration of partial differential equations

Proceedings of the Spring Joint Computer Conference 1968
pp 143-150

9 A E BRYSON W R DENHAM

A steepest-ascent method for solving optimum programming

problems

Journal of Applied Mechanics Vol 29 pp 247-257 1962

10 J TODD

Survey of numerical analysis

New York McGraw-Hill Book Co Chap 11 1962

Hybrid assumed mode solution of non-linear partial differential equations

by DONALD J. NEWMAN and JON C. STRAUSS

Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION

Economical solution of partial differential equations (PDEs) is necessary for the solution of many pressing optimization, identification, design and simulation problems involving spatially continuous systems. The hybrid computer with its parallel organization promises to provide this necessary economy through a combination of increased solution speed and reduced equipment cost with respect to stand alone digital computer methods. This paper presents a hybrid computer oriented assumed mode solution method for non-linear PDEs which are initial value problems in a time-like independent variable. (To facilitate discussion, PDEs with these characteristics are referred to as Dynamic PDEs.) Several examples illustrating the efficiency of the method are included.

Hybrid PDE solution techniques

Most hybrid methods for Dynamic PDE solution involve the reduction of the PDE to a system of ordinary differential equations (ODEs). The ODEs are then solved on the parallel, analog subsection and the results are monitored and combined on the serial digital subsection to obtain the hybrid computer solution. The techniques commonly used for the reduction of a PDE to a system of ODEs consider all but one of the independent variables at a set of discrete points, and an ODE in the remaining independent variable is determined at each point. Extensive bibliographies of such techniques can be found in references 1, 2, 3, and 4. However, in a few situations a solution may be found that is separable in the independent variables.^{3,5,6} The resulting solutions in the separate independent variables are termed modes. In the many cases that do not have separable solutions, modes can be assumed for all but one of the independent variables, and ODEs for the remaining variable can be obtained by approximation techniques.⁷

The techniques involving discretization of all but one of the independent variables divide into two groups. The Discrete Space-Continuous Time (DSCT) techniques give one ODE in the time independent variable for each discrete point in the space independent variables.¹ To provide for accurate derivative approximations in the discretized independent variables, a large number of ODEs must be solved simultaneously especially in two or three space dimension problems. Methods that circumvent this large hardware requirement generally involve time sharing of the parallel analog hardware by the digital.² These methods therefore tend to exhibit the serial speed characteristics of all digital methods; moreover, they can lead to convergence problems.

The Continuous Space-Discrete Time (CSDT) techniques are applicable in one space dimension problems and involve discretization in the time variable; they generally require the solution of an unstable two-point boundary value problem in the space dimension at each discrete value of time.^{1,6} Techniques that eliminate or reduce this difficulty have been suggested but no experimental results have been reported.^{8,9} In addition, the iterative procedures necessary to solve the boundary value problems generally require that the solution of the ODEs be effected serially in time. The resulting serial method does not take full advantage of the parallel computational capability of the hybrid computer.

"Assumed mode" techniques avoid many of these problems, but their use to date has been restricted to the digital computer¹⁰ and to linear PDEs on the analog.¹¹ The number of ODEs to be solved in the assumed mode method is generally much smaller than that required by the DSCT method for equivalent accuracy. (Only one equation is needed for each mode.) These ODEs are stable initial value problems which are well suited for hybrid computer solution.

The assumed mode technique requires a great deal of preliminary work before the problem is ready for the

hybrid computer. The selection of the modes is not easy and is usually subjective. After the modes are determined, a large amount of algebraic computation is required. Much of this computation must be done in closed form and is so extensive that symbolic manipulation is required.

Scope

Of primary concern is the determination of appropriate modes for various boundary conditions. A review of the theory of the assumed mode techniques is presented with a view to applying the technique to non-linear equations. Problems with differential equation boundary conditions are studied for homogeneous and non-homogeneous cases. A sample problem is developed for demonstration, and numerical results are shown to compare the suggested techniques. These results are obtained by numerical integration on the digital computer because the CMU Hybrid Computer is not currently available.

The intent of the work is to demonstrate method, not to build a theoretical structure. No theorems are given that "prove" the merit of the suggestions. The formulation is restricted to simple, specific problems although an attempt is made to use notation that is readily extended to a broader class of problems.

The Dynamic PDEs discussed in this work have two independent variables (x and t) and the operator forms of the PDEs are polynomials in the partial derivatives. The formulation is often applicable only to the second order case in x , but the extension to higher order is generally straightforward.

The large quantity of symbolic computation required would appear to be a serious disadvantage of the technique. However, a digital computer program has been written to do the computation required for any of the class of PDEs discussed in this work and for PDEs with two and three space dimensions as well. The result of this programming effort is that the determination of the ODE system from the PDE and the assumed modes is reduced to a trivial task of preparing approximately ten input statements. This work discusses this aspect only briefly in order to point out where the symbolic computation is employed.

Assumed mode solution

The approach is described and applied to a set of modes to determine the ODEs for hybrid solution.

Non-linear dynamic partial differential equation

The form of the Dynamic PDE of interest is given in (2-1) where u is the dependent function of independent

variables x and t , P is a non-linear partial differential operator with respect to x and f is a forcing function.

$$\frac{\partial}{\partial t} u(x, t) = P[u(x, t)] + f(x, t) \quad (2-1)$$

The solution to this problem must satisfy an initial condition in t and homogeneous boundary conditions in x on the interval $[0, 1]$. (The restriction to homogeneous boundary conditions is removed in Section 3. The $[0, 1]$ interval is chosen for notational convenience only; the solution so obtained may be scaled to any other interval. Brackets are used to denote "operates on," and parentheses are used to denote that the value "depends on.") Thus it is an initial value problem in t , and retention of this character in the system of ODEs to be obtained is desirable.

Assumed modes

An approximate solution v to (2-1) is proposed in the separable form of (2-2).

$$v(x, t) = \sum_{i=1}^n H_i(x) \phi_i(t) \quad (2-2)$$

The assumed spatial modes $H_i(x)$ are pre-selected to satisfy the orthogonality conditions of (2-3) and the spatial homogeneous boundary conditions on the solution to (2-1).

$$\int_0^1 H_i(x) H_j(x) dx = \begin{cases} 0 & i \neq j \\ h_i & i = j \end{cases} \quad (2-3)$$

Since the boundary conditions are homogeneous, v also satisfies the spatial boundary conditions. Only modes that satisfy the boundary conditions are considered although other approaches are possible.^{7,11} The $\phi_i(t)$ functions are weighting functions for the assumed modes.

Subject to the conditions stated above, the selection of the modes depends on the problem, knowledge of the solution and computational convenience. Usually continuity of derivatives of even higher order than the order of the PDE is desirable. If specific regions of the space differ in such a way that the solution has different characteristics there, the assumed modes can accommodate this effect. However, simple algebraic forms for the modes greatly reduce the required algebraic computation.

Ritz-Galerkin approach

The $\phi_i(t)$ functions must be determined to give (in some sense) the best solution to the PDE in (2-1) for the

given modes of (2-2). Ritz suggested minimizing the criterion (2-4) where R is the residual in the PDE when the proposed solution, $v(x, t)$ in (2-2), is substituted into (2-1).

$$\int_0^1 R^2(x, t) dx \tag{2-4}$$

Equation (2-4) gives a measure of the satisfaction of the PDE. To determine a "best" solution, it is necessary to specify in addition that the solution correspond to the initial condition. Galerkin suggested an approximation method based on orthogonalizing the residual with respect to the assumed modes; this is equivalent to the Ritz suggestion for the PDE of (2-1) if the optimization is performed with respect to the time derivatives of $\theta_i(t)$.

The $\theta_i(t)$ are determined by $\frac{d}{dt} \theta_i$ and a $\theta_i(0)$. Substitution of (2-2) into (2-1) yields (2-5).

$$\sum_{i=1}^n H_i \frac{d}{dt} \phi_i = P \left(\sum_{i=1}^n H_i \phi_i \right) + f + R \tag{2-5}$$

The $\frac{d\theta}{dt_i}$ are chosen at each value of t to minimize the criterion on the residual; this implies that the first variations of (2-4) with respect to $\frac{d\theta}{dt_i}$ are necessarily zero as shown in (2-6).

$$\int_0^1 R(x, t) H_i(x) dx = 0 \quad i = 1, 2, \dots, n \tag{2-6}$$

(The result in (2-6) is the orthogonality condition on R and H_i as suggested by Galerkin.) Substituting (2-5) into (2-6) and employing the orthogonality conditions in (2-3) yields the ODEs in $\theta_i(t)$ given by (2-7).

$$h_i \frac{d}{dt} \theta_i = \int_0^1 \left\{ P \left[\sum_{i=1}^n H_i \phi_i \right] + f \right\} H_i dx \tag{2-7}$$

In this work the $\theta_i(0)$ are chosen to give a least squares fit of $v(x, 0)$ in (2-2) to the initial condition on $u(x, t)$ in (2-1). Thus the $\theta_i(t)$ functions are determined from ODE initial value problems.

Differential boundary conditions

A technique to solve a non-linear Dynamic PDE with non-linear differential boundary conditions is developed.

Linear homogeneous case

The problem with homogeneous linear boundary

conditions (one constant, one differential) is posed as follows: Find the solution $u(x, t)$ for $0 \leq x \leq 1$ and $t > 0$ of (3-1).

$$\frac{\partial}{\partial t} u(x, t) = P[u(x, t)] + f(x, t)$$

$$L[u(x, t)] \Big|_{x=0} = 0 \tag{3-1}$$

$$u(1, t) = 0$$

$$u(x, 0) = u_0(x)$$

In (3-1) L is a linear partial differential operator with respect to x, P is a non-linear partial differential operator with respect to x, f is a forcing function and x, t are independent variables.

As an example, the thermal conduction problem with Newton's law of cooling as one boundary condition can be posed in the form of (3-1). If T is the temperature, P[T] is $k \frac{\partial^2 T}{\partial x^2}$ and L[T] is $\frac{\partial T}{\partial x} - \alpha T$, where k and α are physical constants.

A set of assumed modes, $G_i(x)$, must be developed for (3-1) that satisfy the boundary conditions. These modes are developed from the solution to an auxiliary problem to (3-1) with only one spatial boundary condition. For this auxiliary problem, $H_i(x)$ denotes a set of auxiliary modes that are selected to solve the PDE of (3-1) for $u(0, t) = 0$ instead of $L[u(x, t)] \Big|_{x=0} = 0$.

The proposed solution of the auxiliary problem $v(x, t)$ is given by (2-2), where the $H_i(x)$ modes are selected on the basis of knowledge of the system and so that they satisfy the boundary condition in (3-2).

$$H_i(0) = 0 \tag{3-2}$$

The new modes G_i are determined from (3-3) which is an initial value problem.

$$L[G_i(x)] = H_i(x) \tag{3-3}$$

$$G_i(1) = 0$$

G_i can be found in closed form for many $H_i(x)$ functions. This closed form property is required so that $G_i(x)$ may be substituted into (3-1) as an assumed mode. If $w(x, t)$ is defined by (3-4), $L[w] = v$ since L is a linear operator.

$$w(x, t) = \sum_{i=1}^n \phi_i(t) \dot{G}_i(x) \tag{3-4}$$

Also $L[w(x,t)]|_{x=0} = 0$ because $v(0,t) = H_i(0) = 0$ and

$w(1,t) = 0$ because $G_i(1) = 0$. Therefore $w(x,t)$ satisfies the boundary conditions and has the separable form necessary for a proposed solution to (3-1).

The functions $G_i(x)$ are not orthogonal. In order to apply the Ritz-Galerkin method conveniently, $G_i(x)$ should be orthogonalized, possibly with a Gram-Schmidt Orthogonalization Process. The orthogonal functions $\bar{G}_i(x)$ are defined by (3-5).

$$\bar{G}_j(x) = \sum_{i=1}^n c_{ij} G_i(x) \quad (c_{ij} \text{ constant}) \tag{3-5}$$

$$\int_0^1 \bar{G}_i \bar{G}_j dx = \begin{cases} 0 & i \neq j \\ g_i & i = j \end{cases}$$

Again the linearity of L insures that each $\bar{G}_i(x)$ satisfies the boundary condition.

Linear non-homogeneous case

A more general problem with non-homogeneous linear boundary conditions can be reduced to the homogeneous problem considered above. The problem is as stated before except (3-1) is replaced by (3-6).

$$\begin{aligned} \frac{\partial}{\partial t} u(x, t) &= P[u(x, t)] + f(x, t) \\ L[u(x, t)] \Big|_{x=0} &= b(t) \\ u(1, t) &= a(t) \\ u(x, 0) &= u_0(x) \end{aligned} \tag{3-6}$$

A new variable $u' = u - B(x, t)$ is defined where $B(x, t)$ is a function satisfying $L[B(x, t)]|_{x=0} = b(t)$ and $B(1, t) =$

$a(t)$. Since L is a linear differential operator, a B exists and can be determined from $L[B(x, t)] = b(t)$ which is an initial value problem in x, parametric in t. The result of substitution of u' into (3-6) is given in (3-7).

$$\begin{aligned} \frac{\partial}{\partial t} u'(x, t) &= P[u' + B] + f - \frac{\partial}{\partial t} B \\ L[u'(x, t)] \Big|_{x=0} &= 0 \\ u'(1, t) &= 0 \\ u'(x, 0) &= u_0(x) - B(x, 0) \end{aligned} \tag{3-7}$$

This problem is equivalent to the homogeneous case discussed previously. The effect of the non-homogeneous boundary condition is to require the solution of a modified PDE with homogeneous boundary conditions.

Non-linear case

The problem with non-linear differential boundary conditions can be solved as an extension of the technique for the non-homogeneous linear case if the problem is properly stated. The problem statement is as follows: Find the solution $u(x, t)$ for $0 \leq X \leq 1$ and $t > 0$ of (3-8).

$$\frac{\partial}{\partial t} u(x, t) = P[u(x, t)] + f(x, t) \tag{3-8}$$

$$L[u(x, t)] \Big|_{x=0} = N[u(x, t)] \Big|_{x=0}$$

$$u(1, t) = a(t) \quad u(x, 0) = u_0(x)$$

In (3-8) N is a non-linear partial differential operator with respect to x, L is a linear partial differential operator with respect to x containing a derivative of at least as high an order as the highest derivative in N, P is a non-linear partial differential operator with respect to x, f is a forcing function and x, t are independent variables. The form of the differential boundary condition is completely general and L may be selected for convenience subject to the restrictions above.

As an example, the thermal conduction problem with a radiation boundary condition can be posed in the form of (3-8). If T is the temperature, $P[T]$ is $k \frac{\partial^2 T}{\partial x^2}$, $L[T]$ is $\frac{\partial T}{\partial x}$ and $N(T) = \alpha T^4 + \beta$, where k, α and β are physical constants.

The development now proceeds in parallel with the non-homogeneous linear case. In order that the notation reflect the parallelism, $\eta(t)$ is defined by (3-9).

$$\eta(t) = N[u(x, t)] \Big|_{x=0} \tag{3-9}$$

Replacing b(t) with $\eta(t)$ in (3-6) and performing the indicated operations, the equations of (3-7) are derived where $B(x,t)$ is defined by (3-10).

$$\begin{aligned} L[B(x, t)] &= \eta(t) \\ B(1, t) &= a(t) \end{aligned} \tag{3-10}$$

This completes the reduction of the non-linear differential boundary conditions to linear homogeneous boundary conditions.

Since $B(x, t)$ depends on $\eta(t)$ and $\eta(t)$ depends on the solution of the differential equation which depends on $B(x, t)$, the situation is more complex than the linear cases. The $\eta(t)$ function is explicitly defined initially but only implicitly defined for later times as show in (3-11).

$$\eta(0) = N[u(x, 0)] \Big|_{x=0} = N[u_0(x)] \Big|_{x=0} \quad (3-11)$$

$$\eta(t) = N[u'(x, t) + B(x, t)] \Big|_{x=0}$$

Since the functional dependence of u' and B on x is known explicitly, (3-11) is always an algebraic system defining η with t as a parameter. In many cases of practical interest, $\eta(t)$ can be found explicitly, but an explicit solution is not necessary. Furthermore, since $\frac{\partial}{\partial t} B(x, t)$ is contained in the differential equation, the differential equation contains the time derivatives of its solution evaluated at the boundary. This complexity is reflected as an implicit set of ordinary differential equations.

Sample problem

A sample problem is solved to demonstrate the application of methods developed in the previous section, and numerical results are presented to compare the effect of some of the suggested alternatives.

The "free" aquifer surface

The aquifer is a layer of porous rock that is saturated with water and bounded below by an impervious layer. It is "free" when not confined by an impervious upper layer so that the upper surface of the saturated zone is floating. The object of this problem is to determine this surface as a function of time and space. In so doing it is necessary to solve a Dynamic PDE.

Since the flow of water is very slow in a natural aquifer, a form of Darcy's Law is applicable. That is $\bar{v} = -k \nabla h$ where \bar{v} is average velocity, k is a constant of proportionality, ∇ is the spatial differential operator and h is the depth of the saturated zone (the height of the surface above a flat impervious lower layer of rock). This relation is determined by averaging the horizontal component of the velocity over the depth of the saturated zone. The flow density at a point is $h\bar{v}$ and the

increase in flow density is $\nabla \cdot (h\bar{v})$. The additions to the flow (source density) in the aquifer are $\gamma - \mu \frac{\partial h}{\partial t}$ where γ is a function describing input to the aquifer from external sources, and if μ is the porosity of the rock, $\mu \frac{\partial h}{\partial t}$ counts for dynamic changes in the water height. The differential equation is given by (4-1).

$$k \nabla \cdot (h\bar{v}) = \mu \frac{\partial h}{\partial t} - \gamma \quad (4-1)$$

The sample problem deals with a valley of fixed width bounded by impervious rock walls that is partially filled by porous rock. The fixed width justifies a one dimensional analysis, and precipitation in the small region of study is neglected. The lower part of the valley is dammed to form a reservoir as shown in Plan of Lower Valley. The object is to investigate the dynamic behavior of the water table near the shore line when the water level of the reservoir is changed linearly from level 1 to level 2 during the first ten days.

The static water table is determined by the reservoir level at the shore line and the quantity of water flowing down the aquifer. Since this quantity of water is ultimately determined by the precipitation in the upper valley, an adequate boundary condition for this problem would be constant flow at the upper end of the region under study. (Any simpler form of boundary condition would have to be imposed at a point higher in the valley. This would necessitate increasing the length of the region under study and accounting for the shape of the bottom of the aquifer and the distribution of precipitation.) The flow over the dam (q) is 1.5×10^6 ft³/day, the width of the dam (w) is 5000 ft., the porosity (μ) is .15 and the permeability constant (k) is 300 ft/day. The differential equation and boundary conditions are shown in (4-2) where $r(t)$ is reservoir level.

$$k \left(\frac{h \partial^2 h}{\partial x^2} + \left(\frac{\partial h}{\partial x} \right)^2 \right) = \mu \frac{\partial h}{\partial t} \quad (4-2)$$

$$h(0, t) = r(t)$$

$$kh \frac{\partial h}{\partial x} \Big|_{x=x_s} = q/w$$

The initial condition for the problem is the steady state distribution resulting from level 1. The steady-state equation can be solved analytically by integration. The steps of the solution are given in (4-3).

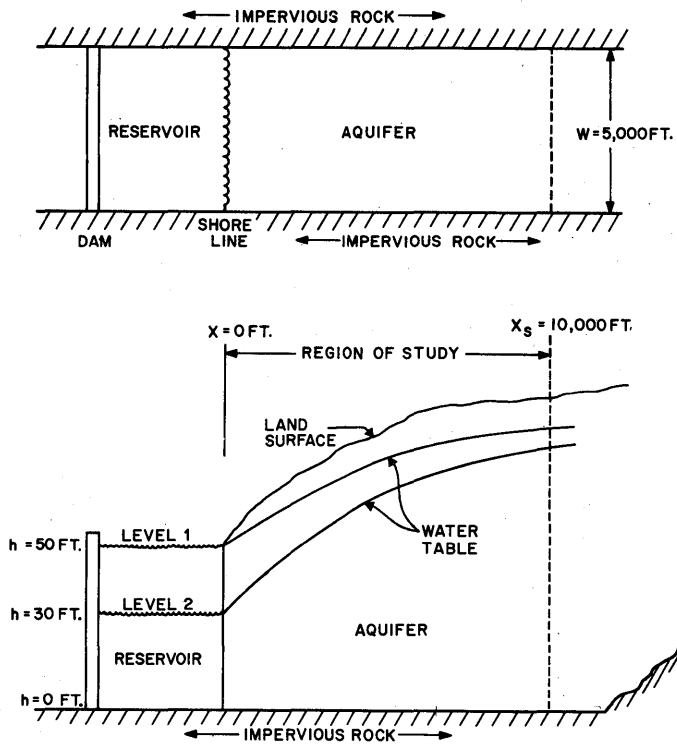


FIGURE 1—Plan of lower valley

$$k \left(h \frac{\partial^2 h}{\partial x^2} + \left(\frac{\partial h}{\partial x} \right)^2 \right) = k \frac{\partial}{\partial x} \left(h \frac{\partial h}{\partial x} \right) = 0$$

$$kh \frac{\partial h}{\partial x} = \frac{q}{w} \tag{4-3}$$

$$h = \sqrt{2qx/wk + h^2(0,0)}$$

The initial condition for the problem is the solution for the reservoir level 1 as shown in (4-4).

$$h(x, 0) = \sqrt{\frac{2q}{wk} x + r^2(0)} \tag{4-4}$$

Revising the boundary conditions

The problem contains a non-linear differential boundary condition which could be replaced by a linear differential boundary condition or rewritten in the form of (3-8)

By introducing the transformation $u = h^2$ the boundary condition becomes linear in u . If this is done, the differential equation (4-5) becomes deceptively simple in appearance. Unfortunately, the \sqrt{u} term is a source of difficulty when integrating to determine the weighting functions for the modes.

$$k\sqrt{u} \frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t} \tag{4-5}$$

This difficulty is so great as to make the transformation $u = h^2$ unsatisfactory for the assumed mode solution of this problem.

However, a linear approximation to the differential boundary condition in (4-2) does not produce this difficulty. The boundary condition in (4-2) is rearranged, and the expansion of the first two terms of the Taylor series for $1/h(x, t)$ about $h(x, 0)$ yields (4.6).

$$\left. \frac{\partial h(x, t)}{\partial x} \right|_{x=x_s} = \frac{q/wk}{h(x_s, t)} \approx \frac{q/wk}{h(x_s, 0)} \left(1 - \frac{h(x_s, t) - h(x_s, 0)}{h(x_s, 0)} \right) \tag{4-6}$$

Since $h(x_s, t)$ is nearly equal to $h(x_s, 0)$ throughout the solution, the three expressions in (4-7) are suggested as alternative boundary conditions.

1. $h = h(x_s, 0)$
2. $\frac{\partial h}{\partial x} = q/(wkh(x_s, 0))$ (4-7)
3. $\frac{\partial h}{\partial x} = q(2h(x_s, 0) - h(x_s, t))/(wkh^2(x_s, 0))$

The technique for non-homogeneous boundary conditions is applied to the problem (4-2) employing each of the boundary conditions above. The results are given in (4-8) where x is scaled by x_s to obtain a normalized form.

1. $2 \cdot 10^{-5} \left\{ (u + (150 - r)x + r) \frac{\partial^2 u}{\partial x^2} + \left(\frac{\partial u}{\partial x} + 150 - r \right)^2 \right\} - (1 - x) \frac{\partial r}{\partial t} = \frac{\partial u}{\partial t}$
 $u(0, t) = u(1, t) = 0$
 $u(x, 0) = -100x - 50 + \sqrt{2 \cdot 10^4 x + 50^2}$
2. $2 \cdot 10^{-5} \left\{ \left(u + \frac{10^8}{15} x + r \right) \frac{\partial^2 u}{\partial x^2} + \left(\frac{\partial u}{\partial x} + \frac{10^8}{15} \right)^2 \right\} - \frac{\partial r}{\partial t} = \frac{\partial u}{\partial t}$

$$u(0, t) = \frac{\partial u}{\partial x}(1, t) = 0 \tag{4-8}$$

$$u(x, 0) = -\frac{10^8}{15}x - 50 + \sqrt{2 \cdot 10^4 x + 50^2}$$

$$3. \ 2 \cdot 10^{-5} \left\{ (u + r \exp(x/2.25)) \left(\frac{\partial^2 u}{\partial x^2} + \right. \right.$$

$$\left. \frac{r}{5.0625} \exp(x/2.25) \right\} +$$

$$\left(\frac{\partial u}{\partial x} + \frac{r}{2.25} \exp(x/2.25) \right)^2 \Bigg\}$$

$$- \exp(x/2.25) \frac{r}{\partial t} = \frac{\partial u}{\partial t}$$

$$u(0, t) = \frac{\partial u}{\partial x}(1, t) - \frac{100}{15} u(1, t) = 0$$

$$u(x, 0) = -50 \exp(x/2.25) + \sqrt{2 \cdot 10^4 x + 50^2}$$

The non-linear boundary condition may be used by rewriting the boundary condition as shown in (4-9).

$$4. \ \frac{\partial h}{\partial x} = \eta(t) \quad \text{where} \quad \eta(t) = \frac{q}{wk} \frac{1}{h(x_s, t)} \tag{4-9}$$

Again the technique for a non-homogeneous boundary condition is applied to give the result in (4-10) where x is scaled by x_s .

$$4. \ 2 \cdot 10^{-5} \left\{ (u + \eta(t)x + r) \frac{\partial^2 u}{\partial x^2} \right.$$

$$\left. + \left(\frac{\partial u}{\partial x} + \eta(t) \right)^2 \right\}$$

$$-x \frac{\partial \eta}{\partial t} - \frac{\partial r}{\partial t} = \frac{\partial u}{\partial t} \tag{4-10}$$

$$u(0, t) = \frac{\partial u}{\partial x}(1, t) = 0$$

$$u(x, 0) = -\frac{10^8}{15}x - 50 + \sqrt{2 \cdot 10^4 x + 50^2}$$

The expression that determines $\eta(t)$ is found by substituting for h in (4-9). This expression and the solution

that matches the initial condition are given in (4-11).

$$\eta(t) = \frac{10^4}{u(1, t) + \eta(t) + r}$$

(4-11)

$$\eta(t) = \frac{-(u(1, t) + r) + \sqrt{(u(1, t) + r)^2 + 4 \cdot 10^4}}{2}$$

In the form presented in (4-7) and (4-8) case 3 requires the handling of very difficult algebra. If this case were formulated in the same way as a non-linear problem, the exponentials could be removed from the PDE and the modes simplifying the algebra considerably. Either way case 3 does not demonstrate any aspect that is not shown in the other three cases and will not be developed any further in this work.

Modes and the ODE system

To demonstrate the algebraic computation necessary to obtain the modes and the ODE system, the two most elementary orthogonal auxiliary modes are chosen. These modes are shown in (4-12) for each of the boundary conditions in (4-7) and (4-9).

$$1. \ H_1 = x^2 - x \quad H_2 = x^3 - \frac{3}{2}x^2 + \frac{1}{2}x$$

$$2. \ H_1 = x - 1 \quad H_2 = x^2 - \frac{5}{4}x + \frac{1}{4}x \tag{4-12}$$

$$4. \ H_1 = x - 1 \quad H_2 = x^2 - \frac{5}{4}x + \frac{1}{4}x$$

In case 1 two boundary conditions must be met since these are the assumed modes, and in cases 2 and 4 only one condition must be met since these are the auxiliary modes.

Determination of the ODE system for case 2 is carried out in detail here. (Case 1 would not demonstrate the techniques, and case 4 is similar to case 2 but more complicated.) The first mode for example is found by integrating $\frac{d}{dx} G_1 = H_1$ with $G_1(0) = 0$, and G_2 is determined similarly to give (4-13).

$$G_1 = \frac{1}{2}x^2 - x \quad G_2 = \frac{1}{3}x^3 - \frac{5}{8}x^2 + \frac{1}{4}x$$

(4-13)

$$\bar{G}_1 = 12x^2 - 24x \quad \bar{G}_2 = 8x^3 - \frac{61}{4}x^2 + \frac{13}{2}x$$

The orthogonal forms, \bar{G}_i , also shown in (4-13), are used in place of H_i in (2-7) to obtain the ODE system of (4-14) consistent with the method discussed in Section 3. The expression for h in (4-14) is obtained by incorporating the transformations made above into the solution (2-2) with H_i replaced by \bar{G}_i .

$$\begin{aligned} \frac{d\phi_1}{dt} &= \frac{.00002}{225} (2700 \phi_1^2 + \\ & (1125 r + 9375) \phi_1/2 + 4875 \phi_2^2/448 \\ & + (375 r + 49375) \phi_2/32 \\ & + 1171875 \frac{\partial r}{\partial t} - 312500/3) \\ \frac{d\phi_2}{dt} &= \frac{.00002}{2925} (907200 \phi_1^2 + (460080 \phi_2 \\ & + 37800 r - 7119000) \phi_1 - 43155 \phi_2^2/4 \\ & - (137655 r + 3073875) \phi_2/2 \quad (4-14) \\ & - 78750000 \frac{\partial r}{\partial t} + 7000000) \end{aligned}$$

$$h = \frac{1}{150} x + r + \sum_{i=1,2} \phi_i(t) \bar{G}_i \left(\frac{x}{10^4} \right)$$

$$\phi_1(0) = -2.94 \quad \phi_2(0) = 4.69$$

The algebra necessary for the determination of (4-14) is performed by the digital computer. The System for Algebraic Computation (SAC),¹² originally developed by G. E. Collins,¹³ is employed to do the orthogonalization, analytical evaluation of integrals and substitution of the proposed solution into the PDE. SAC is capable of manipulating polynomials with complex rational coefficients. The unlimited precision feature of SAC is invaluable in this application as the evaluation of the integrals often involves subtraction of large nearly equal numbers. The restriction to polynomials imposed by SAC is somewhat of a limitation, but not overly restrictive since relatively few other algebraic forms have the necessary closed form integrals.

Numerical results

The results of solving the PDE in (4-2) with the boundary conditions as specified by cases 1, 2 and 4 are presented graphically as water table profiles at different times from the initial condition.

For case 1 solutions at the initial condition, 50 days and 200 days are shown. The maximum deviation of the 200 day solution from the known steady state solution

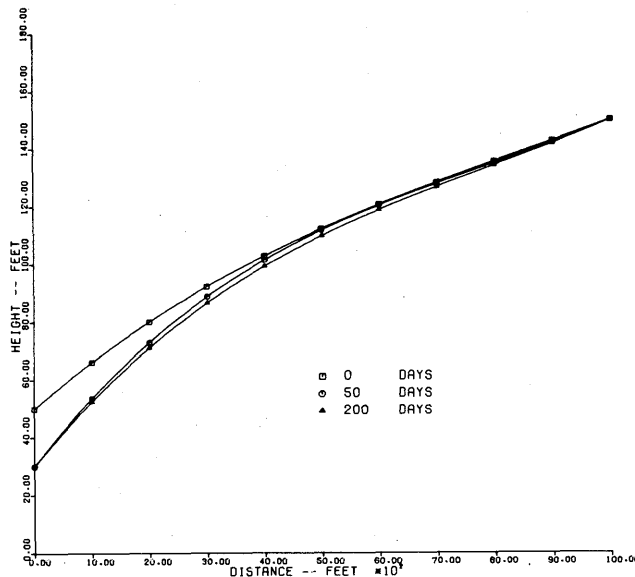


FIGURE 2—Water table profile—Case 1

in (4-3) with the constants chosen to match the boundary conditions for the case is 1.4%.

For cases 2 and 4 a solution at 1000 days is included because the time to reach steady state is much greater than case 1. The maximum deviation of the 1000 day solution from the appropriate steady state solution is 2.6% for case 2 and 2.2% for case 4.

In all cases the initial condition is within 1% of the desired condition in (4-4). Case 2 gives a reasonable approximation to the desired boundary condition when compared with the exact result in case 4. Case 1 differs

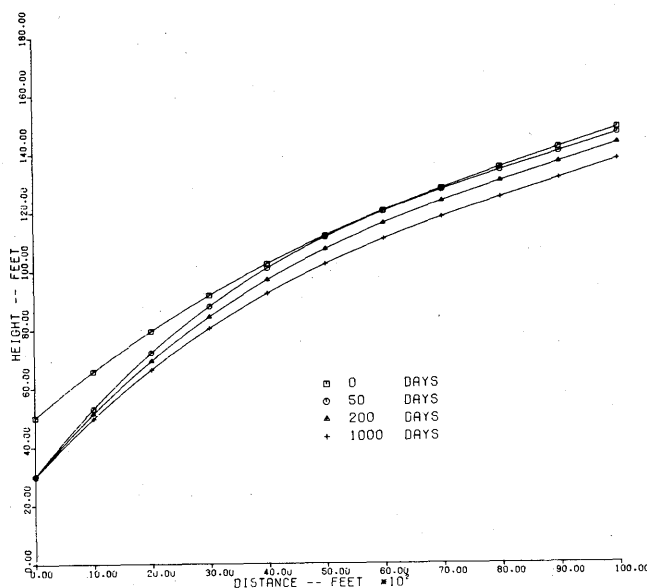


FIGURE 3—Water table profile—Case 2

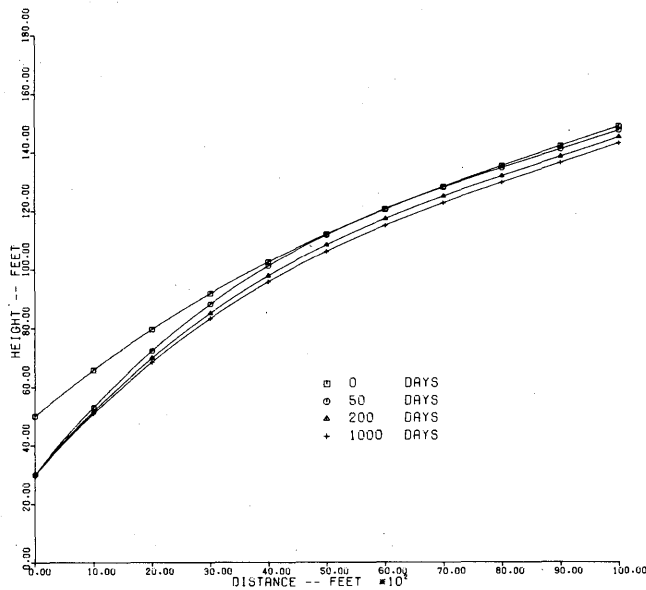


FIGURE 4—Water table profile—Case 4

not only at the end point but the dynamic behavior of the solution is also affected indicating that the floating boundary condition is essential to the overall character of the result. The comparison of the results is restricted to the analytic steady state solution because an analytic dynamic solution is not known.

Considering that the solution to the PDE is determined by only a second order ODE system, the results are very good. If more accurate description of the higher speed dynamics in the first 50 days is needed, more modes would be required in order to better represent the solution.

CONCLUSIONS

The feasibility of the assumed mode technique for hybrid solution of non-linear Dynamic PDEs is demonstrated. Even though the hybrid computer is not employed, the entire procedure is handled in such a way that the analog subsection of the hybrid can be directly substituted for the numerical integration routine employed to solve the ODE system. The power of the assumed mode technique has been increased by providing methods to obtain modes suitable for differential boundary conditions.

In addition to the computational ease of assumed modes, the method has some other valuable properties that may be advantageously exploited on a high speed hybrid computer. Since the SAC system handles all the algebra, an unknown parameter in the PDE or the boundary conditions could be handled without increased difficulty; the parameter would appear in the

ODE system for identification or optimization by the standard techniques for ODEs. Another feature is that the residual function can be calculated exactly, and iterative optimization techniques may be applied to minimize this function for any desired criterion. In this way improvement in the error of the solution may be obtained either by selecting weighting functions for the orthogonality conditions (2-6) or by selecting better modes.

REFERENCES

- 1 D M MacKAY M E FISHER
Analogue computing at ultra-high speed
Wiley New York 1962
- 2 R M HOWE S K HSU
Preliminary investigation of a hybrid method for solving partial differential equations
Aerospace Department University of Michigan Ann Arbor Michigan October 1967
- 3 G A KORN T M KORN
Electronic analog and hybrid computers
McGraw-Hill New York NY 1964
- 4 *Cumulative index*
Simulation Vol 9 No 6 December 1967
- 5 S KAPLAN G SONNEMANN
A generalization of the finite integral transform technique and tables of special cases
Proceedings of the Midwestern Conference on Solid and Fluid Mechanics, 1959
- 6 J E GODTS
Transient neutron distribution solutions by compressed and real time computer complexes
AFIPS 28 SJCC Spartan Books Washington DC 1966
- 7 W M STACEY JR
Modal approximations
MIT Press Cambridge 1967
- 8 R VICHNEVETSKY
A new stable computing method for the serial hybrid computer integration of partial differential equations
AFIPS Proceedings Vol 32 1968 SJCC Spartan Books Washington DC
- 9 H S WITSENHAUSEN
On the hybrid solution of partial differential equations
Proceedings of the IFIPS Conference 1965 Vol 2 Spartan Books Washington DC
- 10 P G CIARLET M H SCHULTZ R S VARGA
Numerical methods of high order accuracy for nonlinear boundary value problems
Num Math 9 pp 394-430 1967
- 11 R VICHNEVETSKY
Functional approximation methods applied to the simulation of field problems
Submitted for publication to IEEE Transactions on Electronic Computers 1967
- 12 D J NEWMAN
360 Reference guide SAC
Technical Memo CCD-98 Carnegie-Mellon Computation Center Pittsburgh Pa December 1967
- 13 G E COLLINS
The SAC-1 list processing system
University of Wisconsin Madison Wisconsin July 1967

Hybrid computer integration of partial differential equations by use of an assumed sum separation of variables

J. ROBERT ASHLEY

University of Colorado
Colorado Springs, Colorado

and

THOMAS E. BULLOCK

University of Florida
Gainesville, Florida

INTRODUCTION

The majority of numerical solution methods for partial differential equations by either analog or digital methods involve some form of finite differences technique,^{1,2} integral transforms,³ or Monto-Carlo methods.⁴ On the other hand, the most common classical analytical approach is based on some form of separation of variables and series expansions.⁵ The motivation for the research presented in this paper was to investigate the possibility of using the classical separation of variables approach as a basis for an efficient computational algorithm. The method studied was developed with a hybrid computer implementation in mind due to the ease in on-line operation in engineering design applications although it could be used for digital computation also.

The technique used in this paper differs from former all analog solutions in that a digital computer is used for function generation and storage. The present application also differs from the usual separation of variables approach in that the boundary conditions need not be given on a constant coordinate surface, e.g., $r = 0$. The numerical approximation used can be viewed either as an application of the method of axial potential expansions⁶ or a generalization of the classical method of an assumed sum separation of variables. The procedure is illustrated in detail by obtaining solutions to Laplace's equation; however, it holds promise for other types of equations. Laplace's equation was chosen as an example because it has been one of the most difficult for analog techniques based on finite difference approximations.

Development of the computing approximation

A familiar analytic approach for the solution of ordinary and partial differential equations is based on series expansions about some convenient nominal point, curve, or surface. The numerical technique to be developed in this section is also based on the idea of series expansion solutions to partial differential equations. The development will be for a specific problem, with the possible extension to other types of problems covered in the following section.

As an example, it is desired to determine the potential in a charge free region with cylindrically symmetric boundary conditions. The potential $V(r,z)$ will solve Laplace's equation in cylindrical coordinates:

$$\nabla^2 V(r,z) = \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial V(r,z)}{\partial r} \right) + \frac{\partial^2 V(r,z)}{\partial z^2} = 0 \quad (1)$$

A typical boundary condition encountered in practice is a specification of the potential $V_b(z)$ on a boundary surface $b(z)$. Such a specification is shown in Figure 1. Mathematically, the condition is

$$V(b(z), z) = V_b(z) \quad \text{for } 0 \leq z \leq a \quad (2)$$

In order to complete the boundary specifications, the potential along the end planes $z = 0$ and $z = a$ is also given.

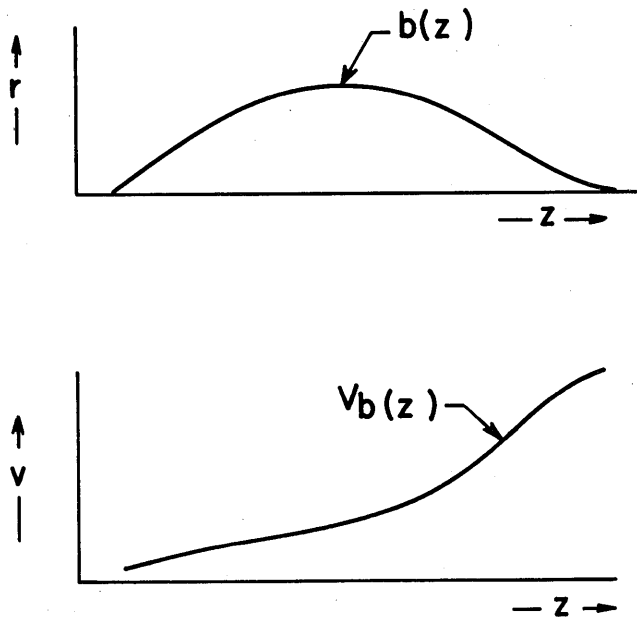


FIGURE 1—Boundary geometry $b(z)$ and boundary potential $V_b(z)$

$$\begin{aligned} V(r, 0) & \text{ specified for } 0 < r < b(0) \\ V(r, a) & \text{ specified for } 0 < r < b(a) \end{aligned} \quad (3)$$

An approximate solution which is accurate for small r may be obtained by expansion of the potential around the axis $(r,0)$ in a power series in r :

$$V(r, z) = a_0(z) + a_1(z) r + \frac{1}{2} a_2(z) r^2 + \dots \quad (4)$$

By substitution of the series into Laplace's equation and carrying out the differentiation, one obtains

$$\begin{aligned} \sum_{k=1}^{\infty} a_k(z) \frac{r^{k-2}}{(k-1)!} + \sum_{k=2}^{\infty} a_k(z) \frac{r^{k-2}}{(k-2)!} \\ + \sum_{k=0}^{\infty} a_k''(z) \frac{r^k}{k!} = 0 \end{aligned} \quad (5)$$

where the primes denote differentiation with respect to z . Since this equation must hold for all r if the series is to represent a solution, the coefficient of r^k must vanish. This leads to the set of equations

$$\begin{aligned} a_1(z) &= 0 \\ \left(\frac{k!}{(k+1)!} + 1 \right) a_{k+2}(z) + a_k''(z) &= 0 \\ k &= 0, 1, 2, \dots \end{aligned} \quad (6)$$

Since $a_1(z) = 0$, then all of the odd subscript coefficients must be zero from the equation above. The remaining equations represent an infinite set of differential equations which may be solved as follows

By definition

$$V(0, z) = a_0(z) \quad (7)$$

With $k = 0$, one obtains

$$a_2(z) = -\frac{1}{2} \frac{\partial^2}{\partial z^2} V(0, z) \quad (8)$$

And for $k = 2$,

$$a_4(z) = +\frac{3}{8} \frac{\partial^4}{\partial z^4} V(0, z)$$

We may continue in this manner to obtain the series solution for Laplace's equation

$$\begin{aligned} V(r, z) = V(0, z) - \frac{r^2}{4} \frac{\partial^2}{\partial z^2} V(0, z) \\ + \frac{r^4}{64} \frac{\partial^4}{\partial z^4} V(0, z) + \dots \end{aligned} \quad (10)$$

to as many terms as desired.

In order to fit the boundary condition at the boundary $b(z)$, the potential must equal $V_b(z)$ or

$$\begin{aligned} V_b(z) = V(0, z) - \frac{b^2(z)}{4} \frac{\partial^2}{\partial z^2} V(0, z) \\ + \frac{b^4(z)}{64} \frac{\partial^4}{\partial z^4} V(0, z) + \dots \end{aligned} \quad (11)$$

Assuming that the series converges rapidly enough so that only terms up to the q th derivative need to be retained, then the equation above becomes a q th order differential equation in the unknown axial potential $V(0, z)$. In order to solve the equation, q boundary conditions are needed. These boundary conditions are obtained by requiring that the solution (10) match the end boundary conditions (3) at a set of arbitrarily specified points $(r_1, 0), (r_2, 0), \dots, (r_{q/2}, 0)$ $(r_1, a), (r_2, a), \dots, (r_{q/2}, a)$. In the case of $q = 6$ the conditions may be found by solving the $q (= 6)$ equations

$$\begin{aligned} V(r_i, 0) = V(0, 0) - \frac{r_i^2}{4} \frac{\partial^2}{\partial z^2} V(0, 0) \\ + \frac{r_i^4}{64} \frac{\partial^4}{\partial z^4} V(0, 0) \end{aligned}$$

$$i = 1, 2, 3 \quad (12)$$

$$V(r_i, a) = V(0, a) - \frac{r_i^2}{4} \frac{\partial^2}{\partial z^2} V(0, a) + \frac{r_i^4}{64} \frac{\partial^4}{\partial z^4} V(0, a)$$

$$i = 1, 2, 3$$

for the 6 unknowns $V(0, 0)$, $\frac{\partial^2}{\partial z^2} V(0, 0)$, $\frac{\partial^4}{\partial z^4} V(0, 0)$, $V(0, a)$, $\frac{\partial^2}{\partial z^2} V(0, a)$, $\frac{\partial^4}{\partial z^4} V(0, a)$. Actually if $r_1 = 0$, $V(0, 0)$ and $V(0, a)$ are given so that only 4 equations need to be solved.

The boundary conditions obtained from the solution of (12) above and the differential equation for $V(0, z)$ as a function of z , equation (11), is a form of two-point boundary value problem which may be solved by iteration on the unknown derivatives at $z=0$, $\frac{\partial^i}{\partial z^i} V(0, 0)$ $i = 1, 3, \dots, q-1$ to match the known derivatives at $a=z$, $\frac{\partial^i}{\partial z^i} V(0, a)$, $i = 0, 2, 4, \dots, q-2$. The usual Dirichlet boundary conditions will specify $V(0, 0)$ and $V(0, a)$ but not $\partial V/\partial z|_{z=0}$. Thus, we must iterate the solution of Equation (10) or, since the problem is linear, obtain a solution directly.¹⁰ In either case, several solutions to the differential equations are involved which is a strong reason for using the analog portion of a hybrid computer for this task.

An alternate derivation of these results can be based on an assumed sum separation of variables solution in Laplace's equation (1). Assuming a solution $V(r, z) = R(r) + Z(z)$ and the usual separation of variables manipulations leads to two ordinary differential equations

$$\frac{d^2 Z}{dz^2} = T \quad (13)$$

$$\frac{1}{r} \frac{d}{dr} \left(r \frac{dR}{dr} \right) = -T \quad (14)$$

The solution of (14) with the boundary conditions $\partial V/\partial r = 0$ along $r = 0$ gives

$$R = -T \frac{r^2}{4} + c \quad (15)$$

Equation (13) could be integrated in a similar fashion to

yield terms in z and z^2 . Rudenberg⁷ has done this and applied the results to electron optics.

The usual separation of variables argument takes T as a constant independent of r or z . However, if we put $T = Vz''$, a function of z , the assumed form for $V(r, z)$ becomes

$$V(r, z) = V_z(z) - V_z''(z) \frac{r^2}{4} \quad (16)$$

where here and in the following we write the axial potential $V(0, z)$ as $V_z(z)$ for convenience.

The above relationship may be recognized as a special case of Eqn. (10) to second order terms in r and may be solved in the same manner.

If Eqn. 16 is differentiated twice with respect to both of the variables and combined as required by Laplace's equation, the result is

$$\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial V}{\partial r} \right) + \frac{\partial^2 V}{\partial z^2} = - \frac{d^4}{dz^4} V_z(z) r^2 \quad (17)$$

which differs from Laplace's equation by the term on the right hand side. Within computing error, the solution of Eqn. 17 does match the boundary conditions. If the solution for V_z can be closely matched by a polynomial in z with powers no higher than 3, then the solution obtained is a good approximation to the solution of Laplace's equation.⁸ We have checked three different solutions for Laplace's equation as used in practical electron lenses and found V_z to be sufficiently smooth that this is an excellent computing approximation.

Extension to other types of equations

The series method given in the previous section may be extended, at least formally, to many other types of partial differential equations in two variables. For example consider the equation

$$\nabla^2 V(r, z) + k_c V(r, z) = f(r, z) \quad (18)$$

If the function $f(r, z)$ has a power series expansion in the neighborhood of $(0, 0)$,

$$f(r, z) = \sum_{i=0}^{\infty} b_i(z) \frac{r^i}{i!} \quad (19)$$

then it is reasonable to look for a series solution for $V(r, z)$ of the form

$$V(r, z) = \sum_{i=0}^{\infty} a_i(z) \frac{r^i}{i!} \quad (20)$$

Substituting (19) and (20) into (18) and equating the coefficients of r^i to zero leads to the relationships

$$a_1(z) = 0$$

$$(1 + k_c)a_i''(z) + 2a_{i+2}(z) = f_i(z)$$

$$i = 2, 3, \dots \quad (21)$$

As before

$$V(0, z) = a_0(z) \quad (22)$$

The equations (16) and (17) may be iterated to solve for the remaining $a_i(z)$'s to obtain the formal series for $V(r, z)$

$$V(r, z) = V(0, z) + 1/4 \left[f_1(z) - (1 + k_c) \frac{\partial^2 V(0, z)}{\partial z^2} \right] r^2 + \frac{1}{12} f_2(z) r^3 + \frac{1}{48} \left[f_2(z) - (1 + k_c) f''(z) \right] r^4 + (1 + k_c)^2 \frac{\partial^4 V(0, z)}{\partial z^4} r^4 + \dots \quad (23)$$

The procedure for meeting the boundary conditions goes through as before.

This technique may also be used for some nonlinear equations, although the series expansion corresponding to (23) may be much more difficult to carry out and the convergence may be poor. Further study in this area is presently being carried out.

An example solution of Laplace's equation

As an example of the kind of boundary value problem which can be solved by this method, we have selected a problem which has also been solved with a digital computer by our colleague, Dr. A. D. Sutherland.⁹ His work on the computer analysis of electron guns requires the solution of Laplace's equation in the structure being analyzed. Dr. Sutherland has developed a program which uses finite differences in axially symmetric coordinates and accepts Dirichlet, Neumann, and mixed boundary conditions within a "reasonable" geometry that can be specified in terms of a chain of simple geometric curves.

A typical electron gun structure is illustrated by Figure 2. The potential is known at each of the mesh points shown from Sutherland's finite difference solution. This solution was obtained with the convergence criterion that the potential at each mesh point cannot change

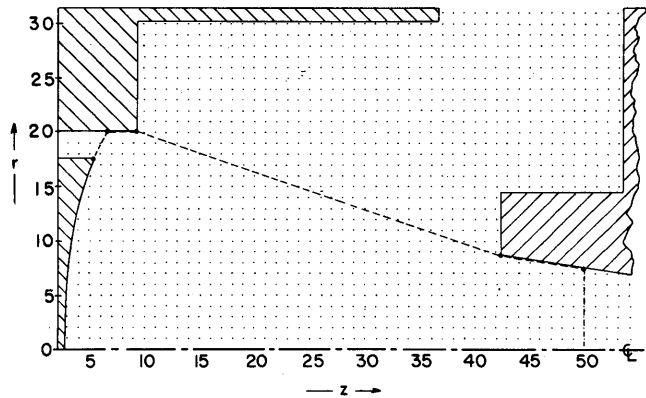


FIGURE 2—Sectional view of an electron gun. The dots are the mesh points of a finite difference solution. The dashed line is the boundary curve to be used in the hybrid computer solution.

by more than 10^{-4} of its previous value. A portion of this structure and solution was abstracted to form a problem with non-coordinate plane boundary geometry. The boundary chosen is shown by the dashed line of Figure 2. This choice of the boundary so that $V_b(z)$ and $b(z)$ are both single valued functions was necessary in order to match the form given in Eqn. (3.) This $b(z)$ curve was set in a 10-segment diode function generator of an AD-80 iterative analog computer.

The potential along the chosen boundary was determined from the finite difference solution and set in a second diode function generator. The straight line segment curves actually used in the AD-80 computer are shown in Figure 3. The circled points are the actual points read from the finite difference solution. Notice the high gradient and the discontinuity in the gradient at the anode tip.

The dimensions on the curves are given in terms of the finite difference solution normalization of one unit

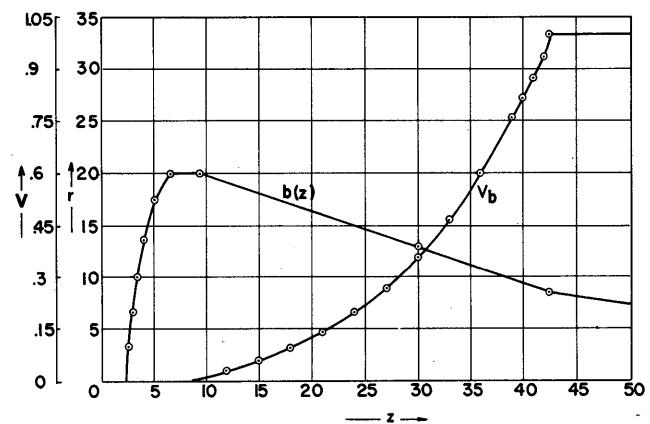


FIGURE 3—Function generator approximation of boundary conditions. The circled points are values from the finite difference solution.

spacing between mesh points. All of the analog scaling is based on a factor of 5 volts per unit. Also, the z variable has an offset that makes the $z = 0$ mesh point fall at -90 volts in the analog computer. Both computations use a normalized anode potential of unity.

The first step of the hybrid computer solution is to solve Eqn. 11 truncated to two terms in the analog section. The patching for an AD-80 iterative analog computer is given by Figure 4. Rather than use a track-hold pair to iterate for the starting value of V_z , its value was set manually by adjusting potentiometer 2A6. The solution which matches end point potential is shown in Figure 5. The axial potential agrees with the digital finite difference solution within 1% and most of the error can be attributed to the use of 10 segment approximations for $V_b(z)$ and $b(z)$. The sharp corner for V_z'' near $z = 43$ is caused by the discontinuity in the first derivative of $V_b(z)$ at this point. We will see that the solution away from the axis of symmetry will be inaccurate in this region.

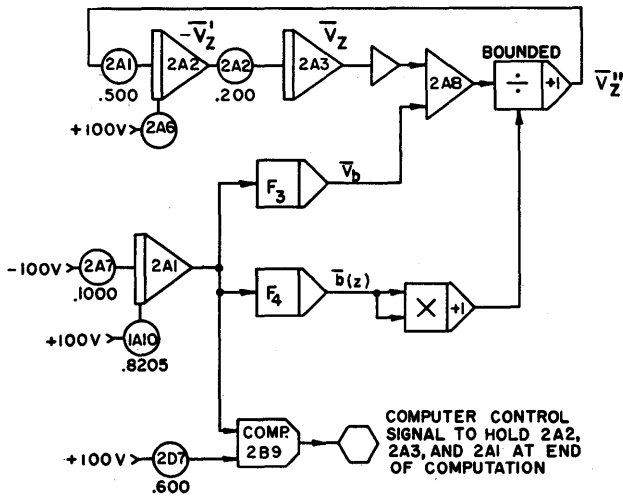


FIGURE 4—Analog solution of Eq. 11 for the electron gun axial potential

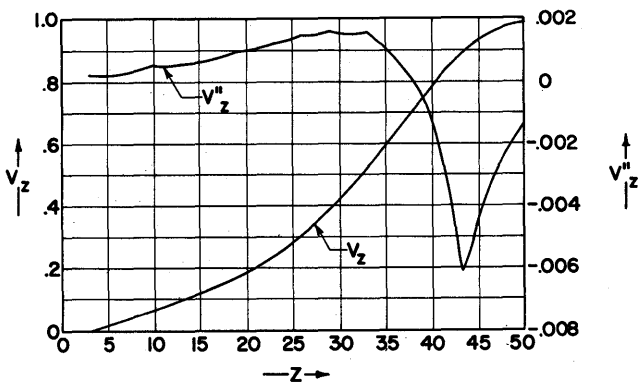


FIGURE 5—The axial potential and its derivatives for the boundary conditions of Fig. 3

The sensitivity of the computation to the gradient was studied by observing the final value for V_z as a function of the starting gradient. (A five-place digital voltmeter is needed to resolve these small increments in starting gradient.) The result is shown in Figure 6. This high sensitivity must be considered if a track-hold pair is used to automatically determine the starting gradient. The problem of high sensitivities to errors is well-known in numerical P.D.E. solutions¹⁰ and leads to ill-conditioned matrices to be inverted in finite difference algorithms. Physically ill-posed problems are generated when the region is so large that the far boundary conditions should be ignored. A smaller range for z would have simplified the numerical difficulties with this problem, however, it would not correspond to the actual physical problem given.

The basis for extending this solution to the region between the axis and the boundary is Eqn. 10. The solution of Eqn. 11 makes both V_z and V_z'' available; therefore, these two variables are supplied to A/D converters and sent to the digital computer for storage in core memory. (If the gradient of the solution is required for electron trajectory computation, V_z' is also stored.) Equation 10 can be implemented in either the digital or analog portion of a hybrid computer. We will

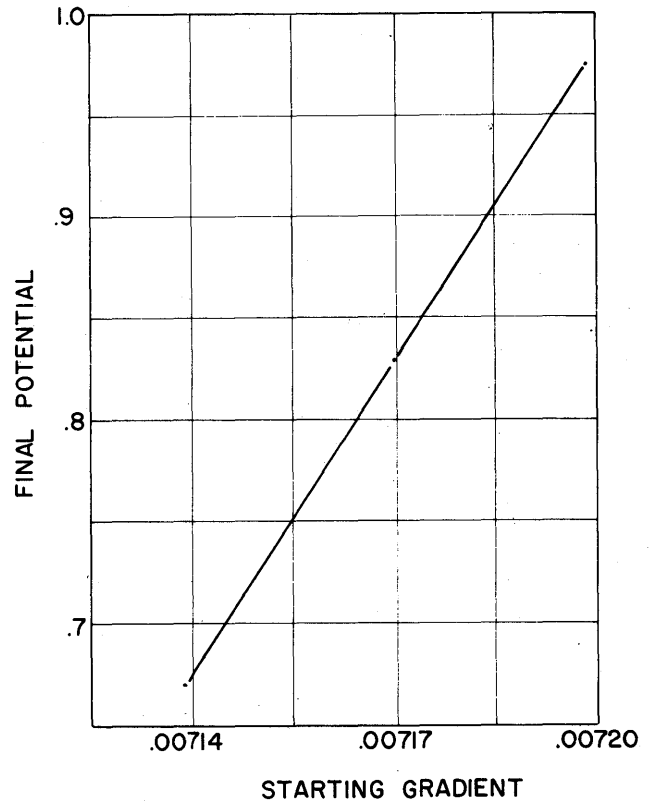


FIGURE 6—Sensitivity of the electron gun problem to starting gradient

illustrate the use of the analog portion because the analog output is continuous in r and easy to display on either a storage oscilloscope or an X-Y plotter. Since $V_z(z)$ and $V_z''(z)$ were stored in the digital computer memory, the digital computer could have been used to complete the solution by evaluating the truncated version of (10) for selected values of r and z .

The hybrid generated generated plots of Eqn. (10) were generated as solutions to the equation

$$\frac{\partial^2}{\partial r^2} V(r, z) = -\frac{1}{2} V_z''(z) \quad (24)$$

with the digital computer supplying (for sampled values of z) $z, V_z''(z)$, and $V(z)$. The solution to (24) with the initial values $V(0, z) = V_z(z)$ and $\frac{\partial}{\partial r} V(0, z) = 0$ (axial symmetry) can be obtained on the analog computer by treating (24) as an ordinary differential equation in $V(r, z)$ with z fixed and r the independent variable. The solution to (24) as r varies from 0 to $b(z)$ sweeps out the solution to the original P.D.E. along a line of $z = \text{constant}$. The digital computer is ideally suited to the task of reproducing the stored values of $V_z(z)$, and $V_z''(z)$ through digital to analog converters and controlling the stopping and starting of the analog integration.

Figure 7 is the result of this process. The entire region was carefully examined and no errors greater than those seen in Figure 9 were found. Notice that the finite difference solution shows a change in curvature at $z = 40$, $r = 8$ which the computing approximation used here cannot match. This is in the region of the sharp change in slope of the $V_b(z)$ curve used here.

We have also considered an all analog method which consists of storing V_z and V_z'' in diode function generators rather than the core memory of a digital computer. An inspection of Figure 5 will show the problem with this idea. The diode function generator which would be used to store V_z'' would need at least 20 segments and probably more. However, V_z is always a relatively smooth function that can be fitted accurately with a 10-segment function generator. A method which uses only V_z would then make possible the use of the iterative analog without the digital computer.

A method which avoids the problem of the inaccurate approximation of V_z'' with a diode function generator and does not require a digital computer can be done by an iteration process in the radial direction by finding V_z'' to meet the boundary conditions. To do this, the solution of Eqn. 11 for V_z is stored in a diode function generator. The functions $V_b(z)$ and $b(z)$ are also required. At a given value of z , we can HOLD the z integrator and iterate the solution of Eqn. 24.

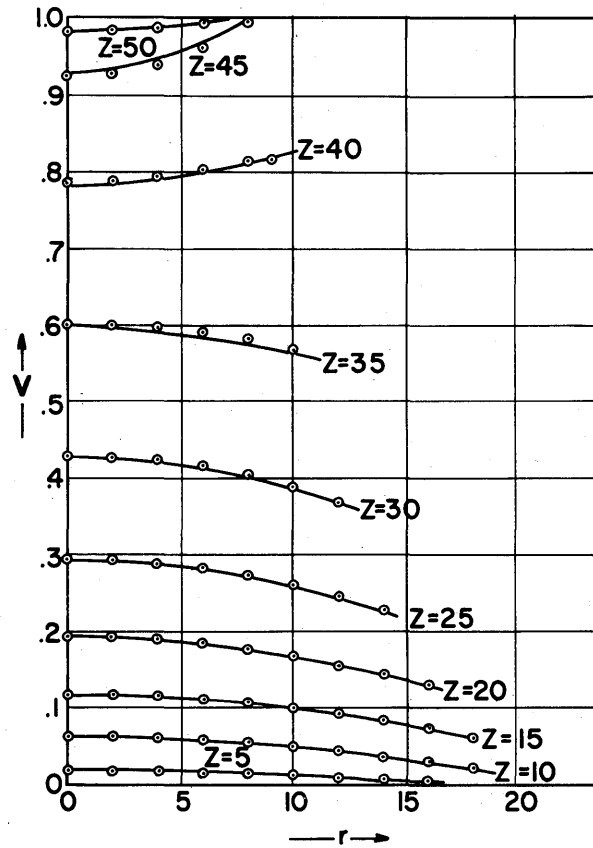


FIGURE 7—Potential in the electron gun. Circled points are from the finite difference digital solution

The starting values for this solution are $\frac{dV}{dr} = 0$ (axial symmetry) and $V = V_z$ at $r = 0$. A track and hold pair of integrators are used to iterate the value of V_z'' in Eqn. 24, to make the potential agree at the boundary. We have found that four iterations are usually sufficient. The analog patching to accomplish this is shown in Figure 8 and the positive logic (as used in an AD-80 computer) is shown in Figure 9. Except that the four iterations in the r direction for each increment in the z direction increase the computing time by a factor of 4, this method will plot the same answer shown in Figure 7.

CONCLUSIONS

The methods given here for solving boundary value problems based on Laplace's equation are comparable in accuracy to finite difference digital solutions if the region studied is several times longer than its maximum radius. The insertion of boundary values with diode function generators is very convenient and enables one to study the effect of boundary condition changes. The solution obtained is continuous in the radial direction

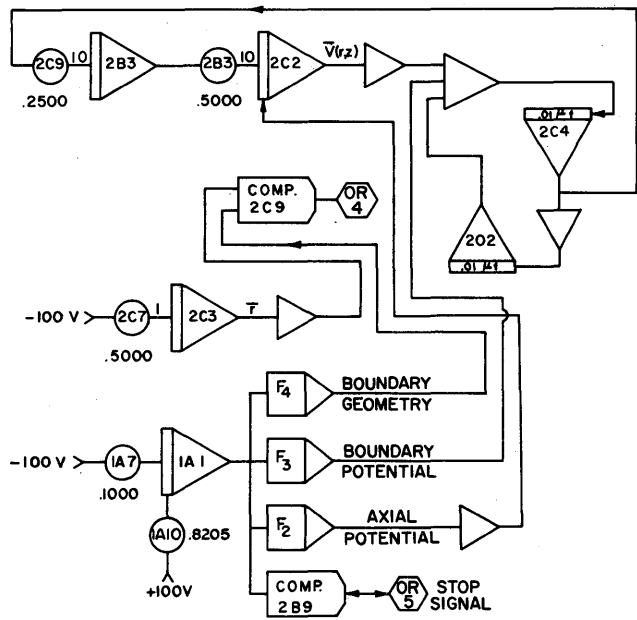


FIGURE 8—Analog patching for extension of the axial potential solution to the boundary of the region

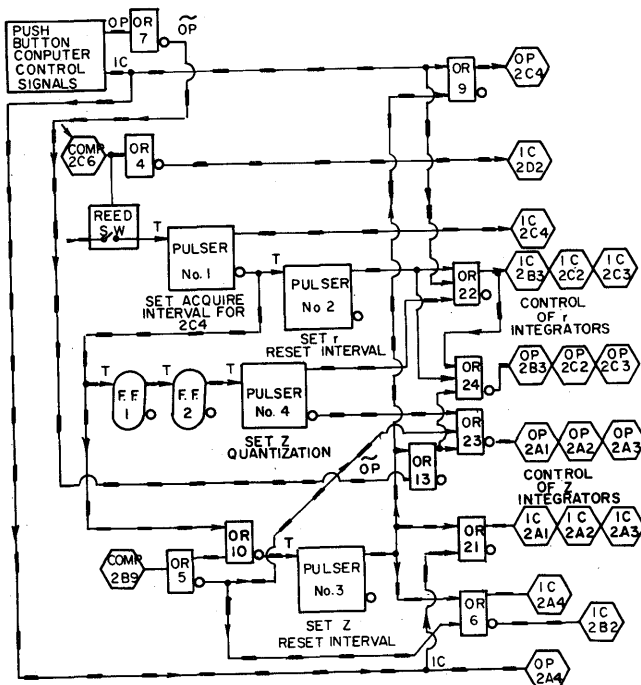


FIGURE 9—Positive logic patching for control of the analog computer circuit given by Fig. 8

and can be as finely incremented as desired in the axial direction with only an increase in computing time. The gradient of potential is also readily available. The method shows sufficient promise for other types of equations to encourage further work in this area.

ACKNOWLEDGMENTS

The solutions given in this paper were obtained with the Applied Dynamics AD-80, IBM 1800 hybrid computing facility of the Department of Nuclear Engineering, University of Florida, Gainesville, Florida. Dr. Alan D. Sutherland, Department of Electrical Engineering, University of Florida, contributed the finite difference digital solution used in the example along with much encouragement and sound advice. The work was supported by an ASSEE-NASA summer faculty fellowship from the Electronics Research Center, Cambridge, Massachusetts.

REFERENCES

- 1 G F FORSYTHE W R WASOW
Finite-difference methods for partial differential equations
Wiley New York 1960
- 2 R M HOWE V S HANEMAN
The solution of partial differential equations by difference methods using the electronic differential analyzer
Proceedings of IRE Vol 41 October 1953 pp 1497-1508
- 3 K N BRAUN A B CLYMER
Assumed mode methods for structural analysis and simulation
Proceedings Conference on Simulation for Aerospace Flight AIAA 1963 pp 244-260
- 4 G A KORN
Random process simulation and measurements
McGraw-Hill New York 1966 pp 204-207
- 5 P M MORSE H FESHBACK
Methods of theoretical physics
McGraw-Hill New York 1953
- 6 K R SPANGENBERG
Vacuum tubes
McGraw-Hill New York 1948
- 7 R RUDENBERG
Electron lenses of hyperbolic field structure
Jour Franklin Inst October 1948
- 8 J R ASHLEY
Iterative integration of Laplace's equation within symmetric boundaries
SIMULATION August 1967
- 9 E E ERICKSON A D SUTHERLAND
Analysis of non-laminar space-charge flow
MICROWAVES Proc of 4th International Congress Centrex Eindhoven Netherlands December 1962
- 10 R VICHNEVETSKY
A new stable computing method for the serial hybrid computer integration of partial differential equations
Proceedings of the 1968 SJCC-AFIPS Vol 32 Thompson Book Co pp 143-150

A hybrid computational method for partial differential equations

by G. A. COULMAN, J. F. SVETLIK and W. H. CLIFFORD

Michigan State University
East Lansing, Michigan

INTRODUCTION

A hybrid computational method for partial differential equations which significantly reduces the time of solutions and error propagation is presented. It is a truly hybrid method, relying on the accurate algebraic capability of the digital machine and the integration capability and speed of the analog.

The method reduces the partial differential equation to a semi-discrete form by means of a method which rigorously incorporates the boundary conditions into the set of coupled ordinary differential equations. These equations are then transformed, yielding a set of independent ordinary differential equations.

The independent set of equations may be solved individually or in arbitrary numbers on the analog computer. The independence eliminates error generated in data transfer associated with coupled solution methods. Also, the independence allows data to be read on any time interval, greatly reducing storage requirements and digital functions. Thus, a faster solution is possible.

Problem statement

A general second order linear parabolic partial differential equation of significant applied interest is as follows:

$$\frac{\partial u(\bar{x}, t)}{\partial t} = \left\{ \sum_i \frac{\partial}{\partial x_i} \left[P(x) \frac{\partial u}{\partial x_i} \right] + Q(\bar{x}) \frac{\partial u}{\partial x_i} \right\} + R(\bar{x})u + S(\bar{x}, t) \quad \bar{x} \in F, t > 0. \quad (1)$$

The associated boundary condition is:

$$C(\bar{x}) u(\bar{x}, t) + B(\bar{x}) \frac{\partial u}{\partial N} = G(\bar{x}) \quad \bar{x} \in H, t > 0, \quad (2)$$

where H is the external boundary of F and $\frac{\partial u}{\partial N}$ is the outward pointing normal on H . An initial condition is usually of the form:

$$u(\bar{x}, 0) = g(\bar{x}) \quad \bar{x} \in F \quad (3)$$

We choose those problems where $P, Q, R, S,$ and g are piece-wise continuous in F . This allows the judicious segmenting of the region later. We also require:

$$\begin{aligned} P(x) &> 0 && \text{in } F, \\ R(x) &> 0 && \\ C(x), \quad B(x) &> 0 && \text{on } H, \\ C(x) + B(x) &> 0 && \end{aligned}$$

This general formulation encompasses a great number of problems of significant physical interest.

Semi-discrete formulation

The objective of the semi-discrete formulation is the reduction of the partial differential equation to a set of coupled ordinary differential equations. The method used is presented by Varga¹ for two-dimensional space. The value of this method is the rigorous incorporation of the boundary conditions into the resultant system of equations. Further extensions of this formulation are presented by Quon.^{2,3} Many alternative methods exist, but this method has many advantages. The presentation of Varga can be extended to three dimensional space if Green's theorem is replaced by Gauss' theorem. The parameterization of the space variables requires a

considerably more complex form. An illustration of the details of the procedure is given for a two dimensional case in Appendix A.

The important result is that the mesh points which are located on H allow use of the boundary condition in the surface integral associated with $P(x)$ at that point. The resultant set of ordinary differential equations can be stated in matrix form as follows:

$$\frac{dU}{dt} = AU + \bar{S}(t). \quad (4)$$

The vector U can be ordered in any desired manner to produce the most useful structure of A . The spatial grid has not been restricted to uniform increments.

Computational method

In general, the problem reduces to one of determining the eigenvalues of the A matrix and the right modal matrix. For the single space dimensional problem, A is tridiagonal and is easily transformed to a symmetric tridiagonal matrix. More details of this will be illustrated in the example. It is shown in Appendix A that for a two dimensional case, A can be made real and symmetric. In a number of cases A can be transformed to this form for three dimensional cases. Frame⁴ presents a method for tridiagonalization of any symmetric matrix. Any tridagonal real symmetric matrix can be diagonalized by a recursive method which computes the eigenvalues to any specified accuracy within the accuracy of the computer. The method presented by Frame⁴ is excellent. The eigenvectors are easily computed by any of a variety of methods. The right modal matrix can be directly computed from the eigenvalues by a simple algorithm for the eigenvectors.

$$\begin{aligned} (\lambda_i I - A)X_i &= 0 \\ M &= [X_1 X_2 \cdots X_n] \end{aligned} \quad (5)$$

The initial step is to transform equation (4).

$$\frac{dY}{dt} = \Lambda Y + M^{-1}\bar{S}(t), \quad (6)$$

where

$$\begin{aligned} U &= MY \\ \Lambda &= M^{-1}AM(\text{diagonal}) \\ M &\text{ is a right modal matrix.} \end{aligned}$$

The result is n independent first-order ordinary differential equations. It is now possible to solve as many

or as few as desired (or possible based on available equipment) simultaneously on the analog computer. Any arbitrary sampling rate may be used for storage of the results without affecting the accuracy. The output sample rate for the forcing function depends on its frequency spectrum, the digital computer capability and the analog time scale.

The physical variables are computed in the digital computer using the right modal matrix.

$$U(t_i) = MY(t_i) \quad (7)$$

The individual problem will dictate the requirements for magnitude scaling. Time scaling can be performed by digital computer by examination of Λ and multiplication of equation (6) by an appropriate scalar. The schematic implementation of this method is illustrated in Figure 1.

Illustrative example

The example has been chosen to illustrate the procedure rather than the details. The equation is a simplification of (1) and is common in chemical engineering work.

$$\begin{aligned} \frac{\partial u(x, t)}{\partial t} &= \frac{\partial}{\partial x} \eta(x) \frac{\partial u}{\partial x} - \frac{\partial}{\partial x} [v(x)u] \\ &+ \xi(x)u + S(x, t) \end{aligned} \quad (1a)$$

F is the closed interval $x_1 < x_2$. The boundary condition from equation (2) becomes two point equations.

$$\begin{aligned} u(x_1, t) - B_1 \left. \frac{\partial u}{\partial x} \right|_{x=x_1} &= G_1 \\ u(x_2, t) + B_2 \left. \frac{\partial u}{\partial x} \right|_{x=x_2} &= G_2 \end{aligned} \quad (2a)$$

H is the points x_1 and x_2 .

$$0 \leq x \leq 1.0 \quad x_1 = 0 \quad x_2 = 1.0$$

The system may be subdivided into arbitrary non-intersecting intervals spanning the set. The partial differential equation is integrated over the interval $i - \frac{1}{2}$ to $i + \frac{1}{2}$ except the two terminal points. The

two terminal points are integrated from 1 to $1 + 1/2$ and $n - 1/2$ to n .

$$\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial u_i}{\partial t} dx = \int_{x_{i-1/2}}^{x_{i+1/2}} \left[\frac{\partial}{\partial x} \left[\eta(x) \frac{\partial u}{\partial x} \right] - \frac{\partial}{\partial x} [v(x)u] + \xi(x)u + S(x, t) \right] dx \quad (8)$$

When (2a) is applied to the terminal elements, the system of equations is the following.

$$\frac{d}{dt} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_i \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \begin{bmatrix} b_1 & c_1 \\ a_2 & b_2 & c_2 \\ & & & a_3 & b_3 & c_3 \\ & & & & & & \vdots \\ & & & & & a_i & b_i & c_i \\ & & & & & & & \vdots \\ & & & & & & & & \vdots \\ & & & & & & & & & a_n & b_n \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_i \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} + \begin{bmatrix} S_1 + \frac{\eta_1 G_1}{B_1 h_1} \\ S_2 \\ \vdots \\ S_i \\ \vdots \\ S_{n-1} \\ S_n + \frac{\eta_n G_2}{B_2 h_n} \end{bmatrix} \quad (9)$$

$$a_i = \frac{\eta_i + \eta_{i-1}}{2h_i^2} + \frac{\nu_i}{2h_i} \quad i = 2, n$$

$$-b_1 = \frac{\eta_1 + \eta_2}{2h_1^2} + \frac{\eta_1}{B_1 h_1} - \frac{\nu_1}{2h_1} + \xi_1$$

$$-b_i = \frac{\eta_{i+1} + 2\eta_i + \eta_{i-1}}{2h_i^2} - \xi_i \quad i = 2, n - 1$$

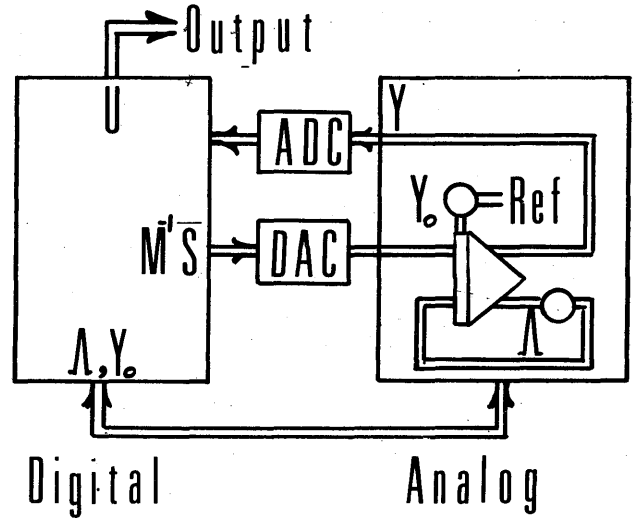


FIGURE 1—Hybrid processor schematic

$$-b_n = \frac{\eta_n + \eta_{n-1}}{2h_n^2} + \frac{\eta_n}{B_2 h_n} + \frac{\nu_n}{2h_n} - \xi_n$$

$$c_i = \frac{\eta_i + \eta_{i+1}}{2h_i^2} - \frac{\nu_i}{2h_i} \quad i = 1, n - 1$$

Each problem, although fitting within a general problem, has characteristics which make it easier to solve if they are recognized. This example is no exception. The matrix A can be converted to a symmetric tri-diagonal form which allows ease of computation of the eigenvalues

where

$$U = DTY$$

$$D = \begin{bmatrix} d_1 & & & & \\ & d_2 & & & \\ & & d_3 & & \\ & & & \ddots & \\ & & & & d_n \end{bmatrix} \quad d_i = \left[\prod_{j=1}^{i-1} \frac{a_{j+1}}{c_j} \right]^{1/2}$$

and T is the right modal matrix of A_1 , where $A^1 = D^{-1}AD$.

As a result of this transformation, the following is necessary to retain a real matrix.

$$h_i \ll \frac{\eta_i + \eta_{i+1}}{\nu_i}$$

Physically it is useful as it requires small intervals when

the dispersion is low which is well known.

The eigenvalues of A_1 are easily determined by a recursion method due to Frame.⁴ A matrix ($n \times n$) of this form has real distinct eigenvalues. A method is provided for estimating the bounds on the λ_i 's. A Sturm sequence is then computed where $P_n(\lambda)$ is the characteristic polynomial, which provides a convergence criterion. A variation count on the sequence $P_k(\lambda_i)$ is equal to the number of roots of $P_n(\lambda) = 0$ which exceeds λ_i in value. This is then a convergent self-checking method to determine the eigenvalues of any symmetric tridiagonal matrix. The right modal matrix (T) of A_1 is easily computed from the eigenvalues.

The example can then be stated in the form of equation (7).

$$\frac{dY}{dt} = \Lambda Y + T^{-1}D^{-1}S(t) \quad (10)$$

where $\Lambda = T^{-1}D^{-1}ADT$, a diagonal matrix.

For computational illustration, the following numerical data were used.

$$\begin{aligned} \eta(x) &= 0.05 + .5x^2 \\ \nu(x) &= 1.0, \quad \xi(x) = -.2 \\ S(x, t) &= e^{-2x}(1 - e^{-t}) \\ B_1 &= \eta_1 \\ B_2 &= \eta_n \\ G_1 &= G_2 = 1.0 \end{aligned}$$

A grid was arbitrarily established with the spacing relationship to η firmly in mind. This is illustrated in Figure 2.

In order to follow the detailed steps in the computer operation as discussed from here on, refer often to the

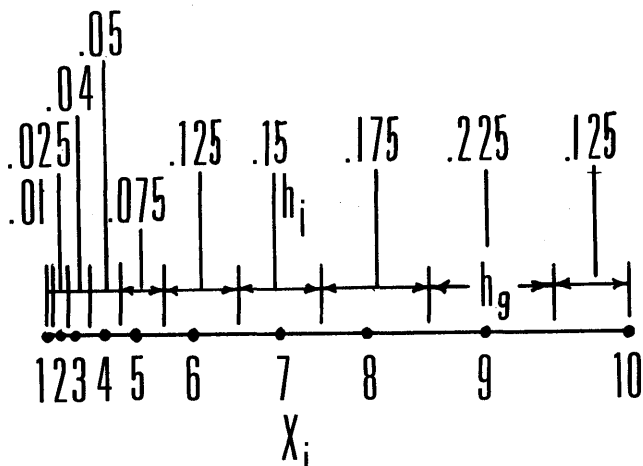


FIGURE 2—Discretized interval

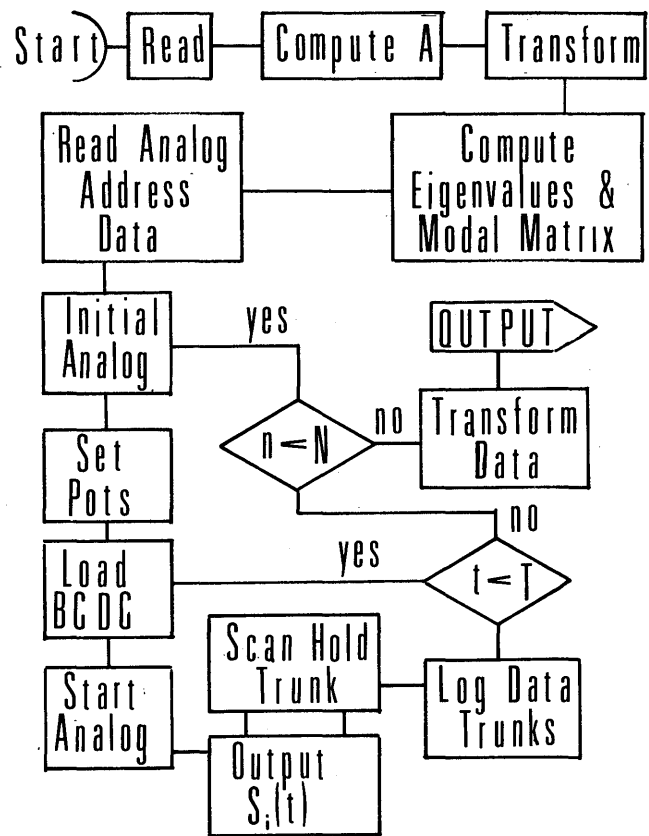


FIGURE 3—Flow chart

flow chart of Figure 3. The program details are omitted but the functional operations are presented. It is assumed that the analog computer is properly wired and prepared for hybrid operation. The digital computer reads the discretized data for the partial differential equation coefficients and coordinates. The A matrix is then computed and transformed to symmetric form. The eigenvalues and right modal matrix are then computed. When very large problems are being done, this portion is performed on the CDC 3600. The card output is then read into the IBM 1800 in the hybrid system. Hopefully a data link will eliminate this problem.

The necessary analog equipment and trunk addresses are entered into the digital computer. The digital machine initializes the analog computer and sets the potentiometers for the first group of elements. The time of computation is controlled from the analog, therefore the binary coded decimal counter is preset by logic signals. This is the scaled time period to the first set of data points. The analog and parallel logic are started simultaneously. The digital computer may be used as a function generator for $S_i(t)$ although in the example this is done on the analog. Synchronization with the analog clock and time scale is an important considera-

tion in the digital function generation. As this was not true in this example, a logic signal from the digital time-scaled the analog by a factor of 1000. It is important to note that conventional data transfer for coupled elements and short time interval computation is eliminated in this method.

When the analog switches itself to the hold mode, a logic signal advises the digital either on an interrupt or a monitored logic trunk. The digital computer reads the appropriate data trunks. If more time points are desired, the BCD counter is set again and the computation continues; otherwise, the analog is initialized, the potentiometers set and a new set of elements are solved at the same time points.

When all the elements have been solved, the vector of data at each time point is multiplied by the appropriate transform matrix. The data are then read out on the selected device in physical state variable form.

In the example chosen, the problem was solved twice with different time increment resolution to demonstrate the independence of accuracy on this choice. The graph in Figure 4 shows that the large time increment data fall on the curve for the high resolution data.

The problem of convergence to the exact solution has not been examined here. It is important to note that the numerical stability problem does not exist in this method. All solutions are stable as can be demonstrated by the definite form of the A matrix. The important problem relates to the magnitude of the coefficients and the selected size of the discrete intervals. It is generally known that as η_i becomes smaller, the problem form approaches the hyperbolic and extremely small intervals are required. In this illustration, this convention

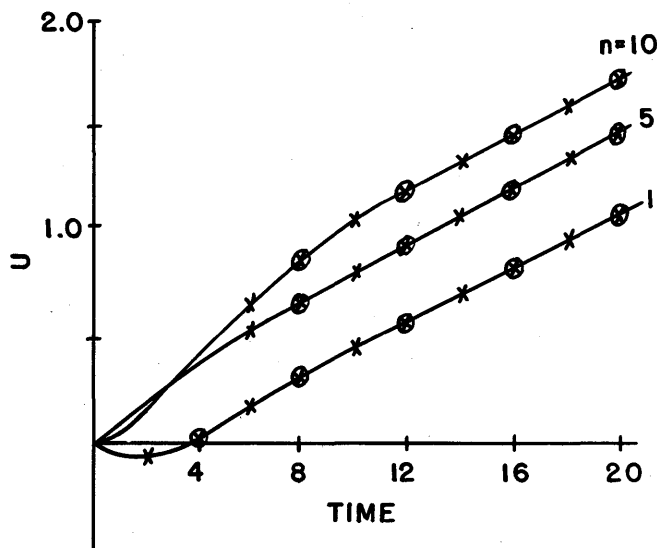


FIGURE 4—Solution response

was observed. Previous experience was the guide although something more rigorous is under study.

CONCLUSIONS

A hybrid method of solution of the parabolic partial differential equation has been developed and demonstrated. This approach makes time sharing of analog equipment a more precise operation. The result is the ability to handle very large meshes without error accumulation caused by the data transfer in time shared coupled computations by conventional methods. This is accomplished by a decoupling transformation. The decoupling significantly reduces the data storage required. The speed is much greater than numerical methods on a purely digital machine as a result of three things: (1) parallel integration of large numbers of elements, (2) speed of analog integration, and (3) the ability to use arbitrary intervals on the continuous independent variable.

FUTURE WORK

Several variations on this basic method are being examined. Preliminary work has been completed on the discrete time problem. It is feasible but not quite as straightforward. Examination of admissible nonlinearities is under way, as well as methods for handling simultaneous partial differential equations.

ACKNOWLEDGMENT

The support of the National Science Foundation under Grant GK-1366 is gratefully acknowledged.

REFERENCES

- 1 R S VARGA
Matrix iterative analysis
Prentice Hall Inc Englewood New Jersey 1962 pp 250-255
- 2 D QUON C R DARSİ
Analytical solutions to the semi-discrete form of the conduction equation for non-homogenous media
Canadian Journal of Chemical Engineering 44 No 5 263-269 1966
- 3 D QUON
Numerical solutions to second order partial differential equations
Preliminary draft of paper
- 4 J S FRAME
Matrix functions and applications
IEEE Spectrum 1 No 3-7 1964
- 5 R M HOWE S K HSU
Preliminary investigation of a hybrid method for solving partial differential equations
Applied Dynamics Report October 1967
- 6 W KAPLAN
Advanced calculus
Addison-Wesley Publishing Company Reading Massachusetts 1965

APPENDIX A

A reduced form of equation (1) and its associated boundary values can be shown to form a system with a symmetric matrix.

$$\frac{\partial u}{\partial t} = \sum_{i=1}^2 \frac{\partial}{\partial x_i} \left[P(x) \frac{\partial u}{\partial x_i} \right] + R(x)u + S(x, t) \quad (1A)$$

with consistent sets and boundary conditions. The summation is the most significant illustration of the finite difference algorithm. The equation is multiplied by dx_1 and dx_2 and integrated over the region about the internal discrete point ij .

$$\iint_{F_{ij}} \frac{\partial}{\partial x_1} \left[P \frac{\partial u}{\partial x_1} \right] + \frac{\partial}{\partial x_2} \left[P \frac{\partial u}{\partial x_2} \right] dx_1 dx_2 \quad (2A)$$

The application of Green's theorem reduces this to a contour integral. (In the case of three dimensions, Gauss' theorem is applied.)

$$\int_{H_{ij}} P \frac{\partial u}{\partial x_1} dx_2 - P \frac{\partial u}{\partial x_2} dx_1 \quad (3A)$$

The integration is performed along the contour illustrated in Figure 1A starting at point a . The resultant finite difference form is given in equation(4A).

$$- P_{i-\frac{1}{2}j} \frac{1}{\delta_{i-1}} \frac{u_{ij} - u_{i-1j} \delta_j + \delta_{j-1}}{2}$$

$$- P_{ij-\frac{1}{2}} \frac{1}{\delta_{j-1}} \frac{u_{ij} - u_{ij-1} \delta_i + \delta_{i-1}}{2}$$

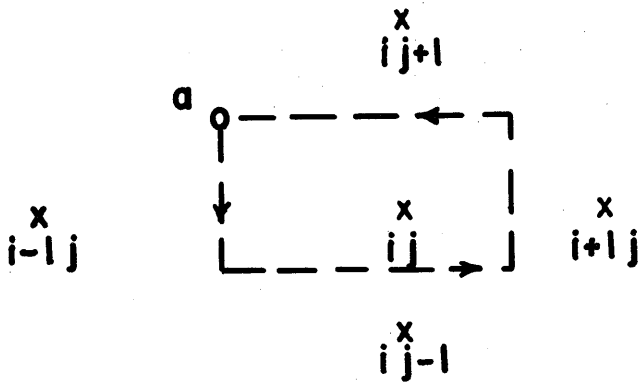


FIGURE 1A—Internal element

$$+ P_{i+\frac{1}{2}j} \frac{1}{\delta_i} \frac{u_{i+1j} - u_{ij} \delta_j + \delta_{j-1}}{2}$$

$$+ P_{ij+\frac{1}{2}} \frac{1}{\delta_j} \frac{u_{ij+1} - u_{ij} \delta_i + \delta_{i-1}}{2} \quad (4A)$$

$$\delta_i = (x_1)_{i+1j} - (x_1)_{ij}$$

$$\delta_j = (x_2)_{ij+1} - (x_2)_{ij}$$

From this it can be seen that the coefficient matrix is always symmetric when the discrete state variables are ordered sequentially. For the more general case of equation (1), a transformation must be found.

For the elements on the boundary a similar procedure is followed with one significant variation. The general element is depicted in Figure 2A. The equation form (3A) is used but parameterization is used along the exterior boundary. A mesh giving essentially linear boundary sections is used.

Define $x_1 = (x_1)_{lm} - y \sin \theta$

$x_2 = (x_2)_{ln} + y \cos \theta ;$

therefore, $\frac{\partial u}{\partial N} = \frac{\partial u}{\partial x_1} \cos \theta + \frac{\partial u}{\partial x_2} \sin \theta .$

Using $dx_1 = - \sin \theta dy$

$dx_2 = \cos \theta dy$

it is possible to write (3A) in the following form for the exterior line segment of the boundary element.

$$\int P \frac{\partial u}{\partial x_1} \cos \theta dy + P \frac{\partial u}{\partial x_2} \sin \theta dy$$

$$= \int P \frac{\partial u}{\partial N} dy = \int P[B^{-1}(G - Cu)] dy \quad (5A)$$

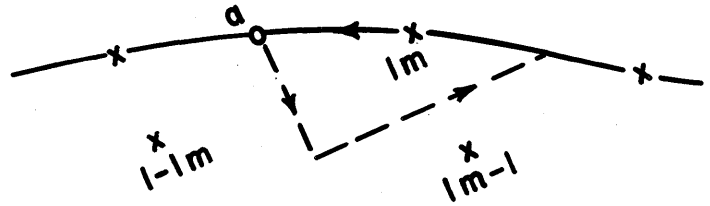


FIGURE 2A—Boundary element

The final form is the result of substitution of the boundary conditions. The contour integration is performed as before but using this final form for the appropriate segment. A complete set of equations results

incorporating all the boundary conditions for an arbitrary region. The result, also, has a real symmetric coefficient matrix.

Preliminary investigation of a hybrid method for solving partial differential equations

by STEPHEN K. T. HSU and ROBERT M. HOWE

*The University of Michigan,
Ann Arbor, Michigan*

INTRODUCTION

In the digital computer solution of a partial differential equation involving both time and spatial variables the partial differential equation is approximated by difference equations arising from the discretization of both the time and spatial variables. The number of difference equations equals the number of spatial mesh points or stations times the number of time increments. In the analog computer solution of a partial differential equation the equation may be approximated by a set of coupled ordinary differential equations, one equation for each station. The set of differential equations is then solved simultaneously. Unless the problem is simple and the number of stations is small, digital computation may take a long time and analog computation may involve much equipment. The present investigation is an attempt to indicate a profitable way to solve partial differential equations with hybrid computation, using only a limited number of high speed analog components for integration and employing the digital computer for function storage and playback.

We are primarily interested in initial value problems where a transient solution is involved. This leads to the consideration of parabolic and hyperbolic equations. Elliptic equations usually are boundary value problems.

First let us review some of the more common general approaches to computer solution of partial differential equations. The discrete-time-discrete-space (DTDS) methods where both time and space variable are discretized are clearly not suited to hybrid computation where one intends to take advantage of the analog computer's high-speed integration capability. The discrete-time-continuous-space (DTCS) methods, where the time variable is discretized and one of the spatial variables is treated continuously have been studied by a number of authors.¹⁻⁸ At each time increment these methods lead to a boundary value problem in the continuous spatial variable. Solution of this boundary value

problem is usually made impractical by stability difficulties. Lacking a good way to meet the end conditions, some investigators have turned to the use of integral equations instead of differential equations.^{1,7} Another approach to the solution of partial differential equations is the Monte Carlo method.¹³⁻¹⁷ This process is quite different and is based on the relation between the partial differential equation defining a probability distribution function and the related random process. The standard approach in analog computation is to use continuous-time-discrete-space (CTDS).⁹ However, instead of solving the set of coupled ordinary differential equations in parallel using many analog components (one set for each station), one can use a hybrid system and employ a relaxation technique. This is the method described in the present paper. Solutions are iterated in a serial fashion using a relatively small number of analog components which are time-shared among sets of mesh points. This CTDS approach appears to be a logical choice for hybrid computation. It has been proposed and tried before,¹⁰⁻¹² but no results beyond the preliminary stage have been found.

The CTDS formulation and hybrid computation

The CTDS formulation is easily adapted to hybrid computation to allow combined parallel-serial operation. The set of ordinary differential equations is integrated one equation at a time (or one group of equations at a time) in a serial fashion. Since the equations are coupled, the solutions at all stations not being solved by direct integration must be guessed in the beginning and stored digitally. When the equation of a particular station (or group of stations) is being solved by the analog, the guessed or previously computed solutions of neighboring stations are fed in as part of the forcing inputs through D/A converters while the computed analog solution(s) is sampled and stored in the digital storage, replacing the corresponding previous values.

The relaxation procedure then moves on to the next station (or group of stations). The process goes on to cover all the stations and to form a complete iteration. Iterations are repeated until the solutions converge to a satisfactory degree of accuracy.

This technique is quite similar to that used in digital computer solution of elliptic boundary value problems. In the latter case the solution at a station is approached in value, while in the CTDS relaxation method the solution function at a station is approached.

The iterative CTDS procedure has been used to solve one-dimensional and two-dimensional linear diffusion equations, a nonlinear one-dimensional diffusion equation and a one-dimensional wave equation. The following sections explore at some length these computer experiments. The feasibility of the approach is confirmed and the general characteristics of the iterative process are observed. Moreover, several methods for reducing the computational time are proposed and verified.

It should be pointed out that it is possible to prove convergence of the iterative procedure under quite general circumstances although the convergence proof is not included in this paper.

Simulation of the hybrid system

Because of the lack of availability of a suitable hybrid computer system, and also to insure experimental results of high precision, the hybrid process was simulated digitally using an IBM 7090. A fourth-order Runge Kutta integration formula was used to simulate the continuous analog integration. Simulation of the interface system (A/D sampling and D/A interpolation) was included. The effects of quantizing errors in the conversion processes were not included.

One-dimensional diffusion equation

Consider a simple one-dimensional heat flow problem (diffusion equation) described by the equation:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \quad (1)$$

with the boundary conditions

$$u(0, t) = 0, \quad \frac{\partial u}{\partial x} \Big|_{x=1} = 0, \quad (2)$$

and the initial condition

$$u(x, 0) = 1. \quad (3)$$

If we divide the spatial coordinate x into $(n + \frac{1}{2})$ equal

increments and let $u_i(t)$ be the temperature at $x = i\Delta x$, then the ordinary differential equations obtained are:

$$\frac{du_i}{dt} = \frac{1}{(\Delta x)^2} [u_{i-1} - 2u_i + u_{i+1}], \quad i = 1, 2, \dots, n-1 \quad (4)$$

$$\frac{du_n}{dt} = \frac{1}{(\Delta x)^2} [u_{n-1} - u_n],$$

where $u_0(t) = u(0, t) = 0$. We will call the solution of these n coupled equations the CTDS solution. In the standard analog method of computation, these equations are solved simultaneously. But in the iterative CTDS process the equations are solved iteratively, one at a time. First we take an arbitrary guess for $u_i(t)$ and call it $u_i^0(t)$ for all i from 1 to n . For instance, we can let $u_i^0(t) = 1$ for all i and for all t . Now with the analog computer we solve for the temperature $u_1(t)$ at the first station using the digitally-stored u_0 and $u_2^0(t)$ through D/A converters as forcing functions. The analog solution for $u_1(t)$ is sampled and stored during the solution using the A/D converter system, and becomes $u_1^1(t)$, replacing $u_1^0(t)$. Next $u_2^1(t)$, the temperature at the second station, is computed with the analog system, using $u_1^1(t)$ and $u_3^0(t)$ as forcing functions. Continuing in this way we obtain $u_1^1(t), u_2^1(t), \dots, u_n^1(t)$ with the analog system, in each case storing the solution in the digital computer memory. The procedure is repeated to obtain $u_i^2(t), u_i^3(t), \dots, u_i^k(t)$. The process continues until $u_i^k(t)$ and $u_i^{k+1}(t)$ agree within a certain prescribed small number, .0005 in our example, over the whole computed time interval for each i .

The problem was simulated for $n = 6$ with a zero-order extrapolation for data playback. The converter sample period used was .01 units of t . Figure 1 shows $u_2^k(t)$ for various values of k . The functions are seen to converge very nicely, but not to the correct CTDS solution. This is because of the inaccuracy caused by the zero-order extrapolation used in function playback for each successive iteration.

Since implementation of higher order interpolation can use considerable analog equipment, a zero-order extrapolation with the playback function advanced in time by one-half the sample period was used to compensate for the average time delay. The result obtained (Figure 2) stays within .01 of the true CTDS solution. Even better agreement would have resulted if the playback sample period had been smaller.

The first guess $u_i^0(t)$ has a definite effect on the rate of convergence, at least initially, as is demonstrated by Figures 3 and 4. Note that $u_i(t) = 0, i = 1, 2, \dots, 6$,

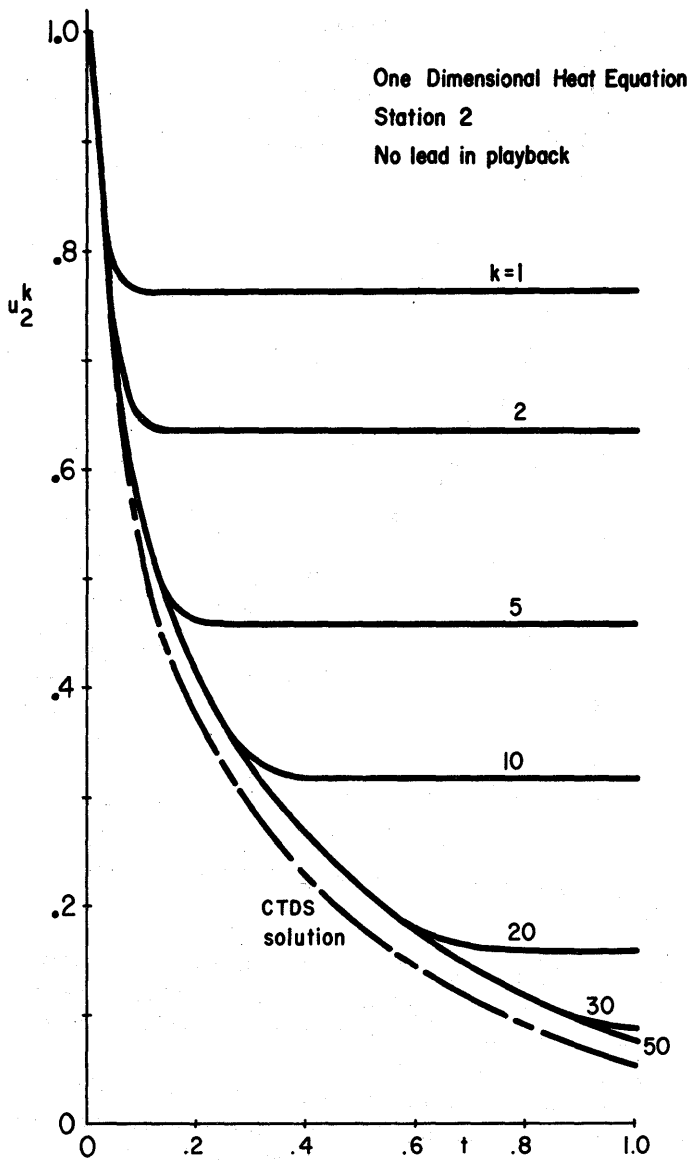


FIGURE 1—Iterative solution

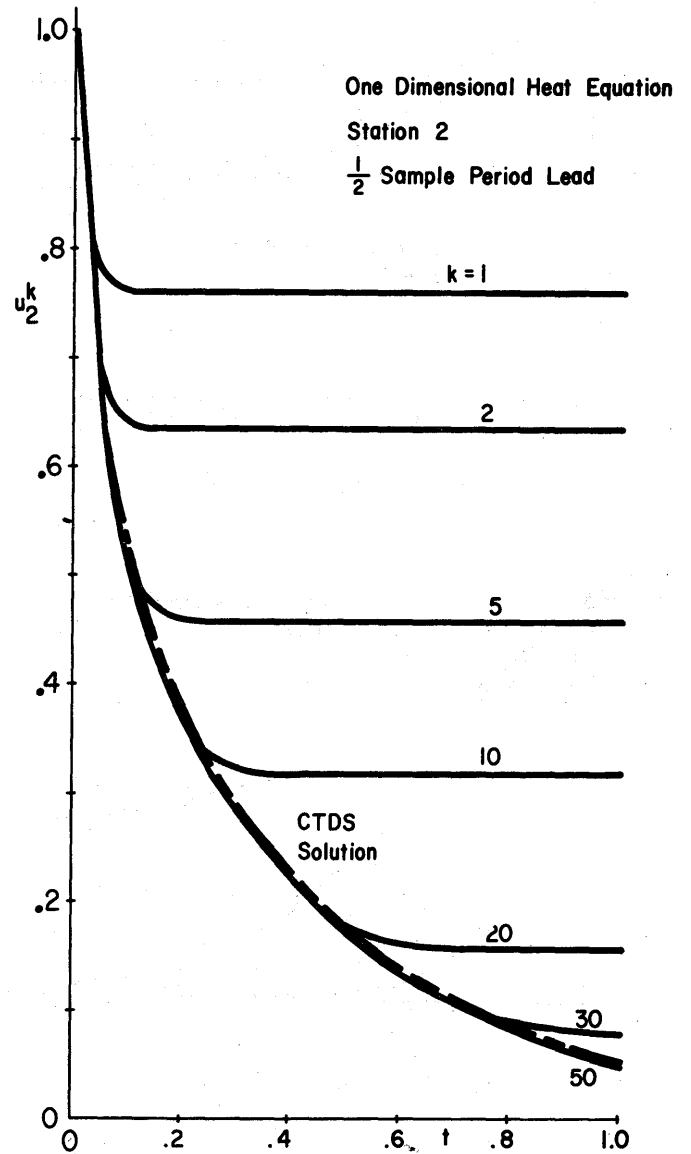


FIGURE 2—Iterative solution with lead in playback

is a steady state solution of the problem and is used as the initial guess, $u_i^0(t)$, in Figure 4. It is clear that choosing the steady state solution as a first trial guess may not be as good as choosing the initial condition for the first guess, as in Figure 3.

If a problem is symmetric about its midpoint in x , only one half of the stations need be computed. The $6\frac{1}{2}$ station problem described above can be interpreted as one half of a 13 station problem with zero boundary conditions at both ends. The number of iterations required for convergence is the same, although in the 13 station problem twice as many runs per iteration are needed.

Observation of the results in Figures 1 to 4 suggests

that it is almost useless to compute $u_i^k(t)$ beyond the t where it departs considerably from the CTDS solution. It is possible to take advantage of this by dividing the desired solution time interval into small sub-intervals. In this way total computing time can be reduced considerably. For example, the time interval $[0,1]$ might be divided into ten equal sub-intervals. During the first sub-interval the iterative computing procedure is applied until the specified convergence criterion is satisfied. The resulting end points of the first sub-interval are used as the starting points for the second interval. This procedure is then repeated for each subsequent interval. Furthermore, the use of a linear extrapolation rather than $u_i^0(t) = \text{constant}$ as the first guess at each

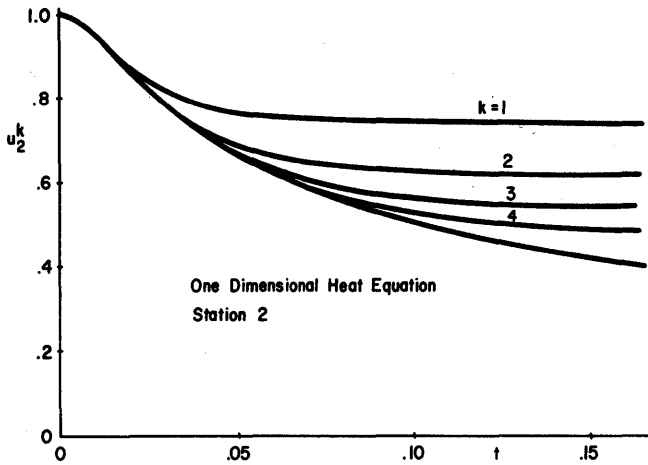


FIGURE 3—Solution with $u_i^0 = 1$

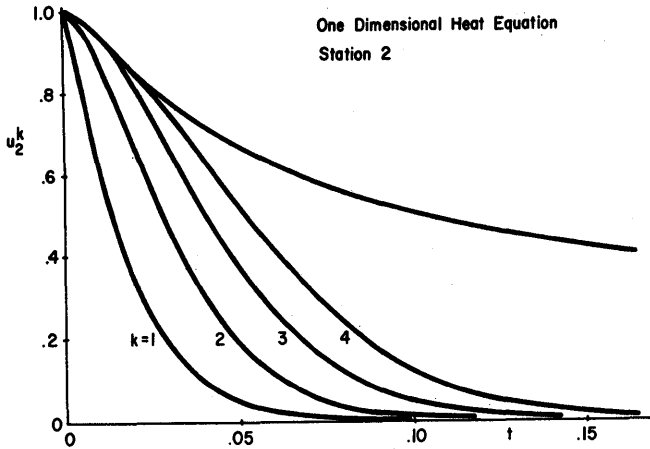


FIGURE 4—Solution with $u_i^0 = 0$

interval helps to reduce the number of iterations required for convergence. More specifically, let

$$u_i^0(t) = \bar{u}_i(T) + \frac{du_i}{dt} \Big|_{t=T} (t - T) \quad (5)$$

$T \leq t \leq T + \Delta T$

where $\bar{u}_i(t)$ is the approximation to the CTDS solution obtained on the interval $T - \Delta T \leq t \leq T$ when the convergence criterion on that interval has been met. If the time interval ΔT is small, this linear approximation will not be far from the true CTDS solution.

The same heat flow problem described above was run using twenty sub-intervals to cover the interval $0 \leq t \leq 1$. The total number of iterations (72) was somewhat larger than that required earlier in Figure 2, but since each solution is only one-twentieth as long, the

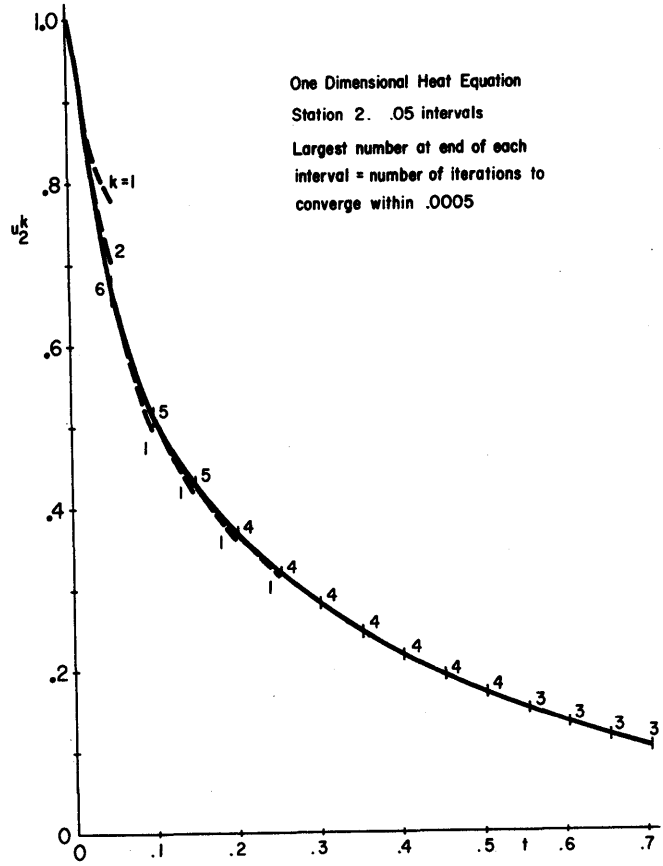


FIGURE 5—Solution using sub-interval iterations

overall saving in computational time is very substantial. Figure 5 shows the approximate solution, which is very close to the true CTDS solution. The latter is not plotted for the sake of clarity.

Since the .0005 value of the convergence criterion is not zero, the end point error may accumulate and appears to have an upper bound of $.0005 \times 20 = .01$ at the 20th interval. Actually, for linear stable systems, the end point errors tend to attenuate with time. In the above example, the error accumulation is less than .001 at the end of the 20th interval. If a value smaller than .0005 is used, the accumulated error will be correspondingly smaller. Ultimately the error obtained depends on the accuracy of the analog and interface components.

These results not only confirm the early investigation of Max¹⁰ showing convergence, but also show the practicality of zero-order extrapolation and the advisability of computing successively through small intervals.

A nonlinear one-dimensional heat equation has also been solved by the method. The problem considered is

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x} u \frac{\partial u}{\partial x} \quad (6)$$

with $u(0, x) = 1$, $u(t, 0) = 0$ and $[\partial u / \partial t]_{x=1} = 0$. As in the linear case $6\frac{1}{2}$ stations are taken along x , yielding the following set of ordinary differential equations:

$$\frac{du_i^k}{dt} = 21.125[(u_i^{k-1})^2 - 2(u_i^k)^2 + (u_{i+1}^k)^2]$$

$$i = 1, 2, \dots, 5$$

(7)

$$\frac{du_6^k}{dt} = 21.125[(u_5^k)^2 - (u_6^k)^2]$$

with $u_0 = 0$, $u_i^k(0) = 1$ and $u_i^0(t) = 1$ for $i = 1, 2, \dots, 6$ on interval $0 \leq t \leq .05$.

The problem is solved successively through small time intervals of .05 units in t with linear approximation (eq. 5) for $u_i^0(t)$ and half-sample period lead in playback. The approximations converge nicely to within .005 of the CTDS solution; and $u_i^k(t)$ is shown in Figure 6.

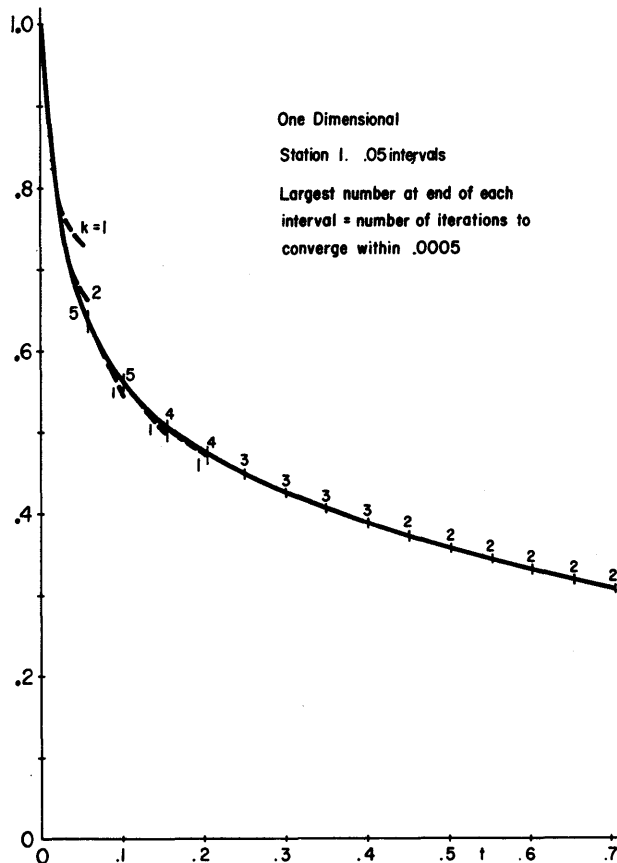


FIGURE 6—Solution of the nonlinear diffusion equation

One dimensional wave equation

As another example, consider the wave equation

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$$

(8)

with $u(0, t) = .4$, $u(x, 0) = 0$, $[\partial u / \partial x]_{x=1} = 0$ and $[\partial u / \partial x]_{x=0} = 0$. The space coordinate is divided into $3\frac{1}{2}$ stations and we have the following set of equations:

$$\frac{d^2 u_i^k}{dt^2} = 12.25[u_i^{k-1} - 2u_i^k + u_{i+1}^{k-1}] \quad i = 1, 2$$

(9)

$$\frac{d^2 u_3^k}{dt^2} = 12.25[u_2^k - u_3^k]$$

with $u_0 = .4$ and $u_i(0) = [du_i/dt]_{t=0} = 0$. The computational procedure is applied over successive small time intervals of width .5. In the first interval a constant $u_i^0(t) = 0$ is used as the first guess, while in subsequent intervals the linear approximation (eq. 5) extrapolating from the starting points is used to begin the computation. As before, a one-half sample period lead is used in playback. The behavior at station 1 is shown in Figure 7.

The CTDS solution of the set of coupled equations was also computed to check the accuracy. The iterative solution is very close to it and cannot be distinguished from it in Figure 7. Figure 8 shows the difference when plotted on a magnified scale.

The iterations converge very nicely as can be seen. The .0005 convergence criterion applies to $u_i^k(t)$ as well as their derivatives. In other words, iteration does not stop until both $u_i^k(t)$ and their derivatives con-

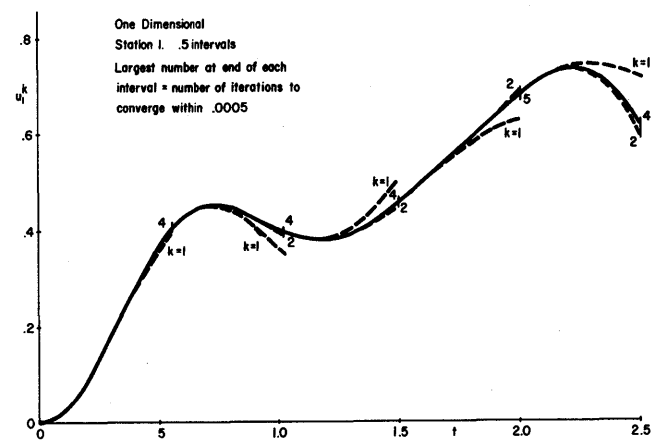


FIGURE 7—Solution of the wave equation

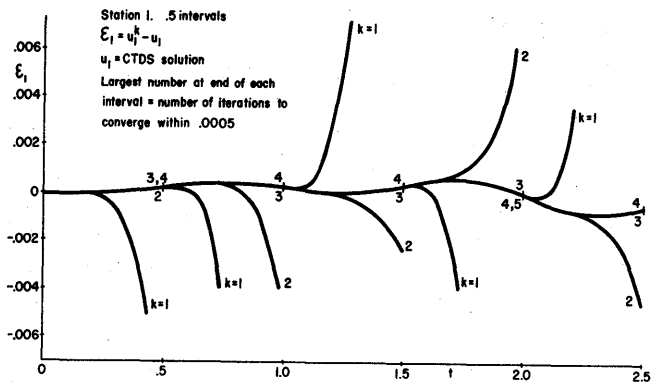


FIGURE 8—Error of the iterative solution

verge within .0005 in successive iterations. With this criterion, the maximum error at the end of the fifth interval is only about .001. Due to the nature of the equation the error oscillates with increasing amplitude, as shown in Figure 8.

Two dimensional diffusion equation

Consider next the diffusion equation in two spatial dimensions

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \quad 0 \leq x \leq 1, \quad 0 \leq y \leq 1 \quad (10)$$

over a unit square with the temperature $u = 0$ on all four boundaries and a unit initial temperature $u(x, y, 0) = 1$.

Dividing the square into $(n + 1) \times (n + 1)$ equal increments leads to $n \times n$ internal points. The CTDS equation is

$$\frac{du_{i,j}}{dt} = \frac{1}{(\Delta x)^2} [u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1} - 4u_{i,j}] \quad (11)$$

where i corresponds to x and j corresponds to y . Iteration can be done across the square point by point. This, however, will be a long, tedious process. To save time, a group of points can be connected together and solved in parallel at each run. Assume there is enough analog equipment to cover all n points along x at a certain j so that $u_i^k, j(t)$ for $i = 1, \dots, n$ are computed simultaneously. Starting from $j = 1$, the computation proceeds from line to line in the y direction and iterates until all the $u_i^k, j(t)$ converge to within a small prescribed value between successive iterations. In this arrangement, the iterative equation becomes

$$\frac{du_{i,j}^k}{dt} = \frac{1}{(\Delta x)^2} [u_{i-1,j}^k + u_{i+1,j-1}^k + u_{i+1,j}^k + u_{i,j-1}^{k-1} - 4u_{i,j}^k] \quad (12)$$

$i = 1, \dots, n$

Again a fourth-order Runge-Kutta integration is used in the digital simulation. The sample period is $1/400$ units in t . In playback, a half sample period lead s used as discussed before.

For the problem stated the unit square is divided into 7×7 divisions. Due to symmetry only $1/4$ of it, a $3\frac{1}{2} \times 3\frac{1}{2}$ square with proper boundary conditions, need be computed. The behavior of $u_{2,2}^k(t)$ is shown in Figure 9. It converges so closely to the CTDS solution

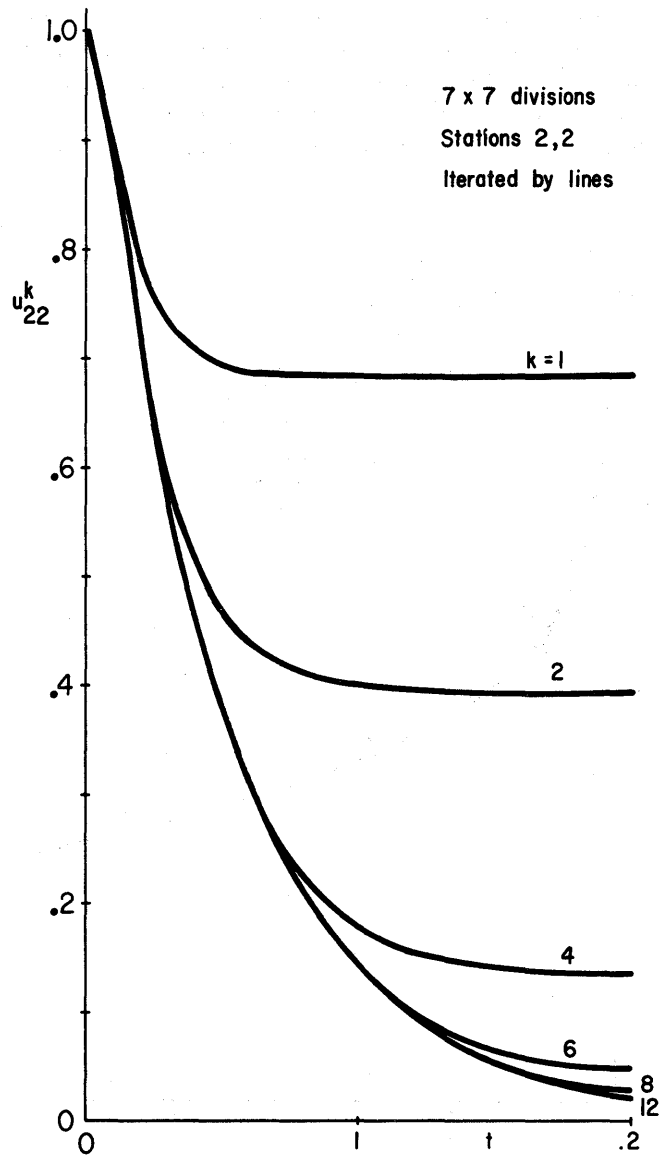


FIGURE 9—Two dimensional diffusion equation, 7×7

that the latter cannot be distinguished in the graph.

It is also possible to implement the parallel solution in square blocks instead of lines. To study this a 10×10 station problem was solved, first iterating line by line, then block by block. In the latter case the 9×9 internal points are divided into nine 3×3 blocks. Either way one iteration takes 9 runs to cover all the points. Iterating by blocks has a faster rate of convergence as can be seen in comparing Figure 10 with 11.

To save time, the computation is done in intervals of width .05. The procedure starts with a constant $u_{i,j}^0(t) = u_{i,j}(0) = 1$ in the first interval and uses a linear extrapolation for the subsequent intervals. Convergence is clearly evident.

Iteration by blocks has another advantage in the

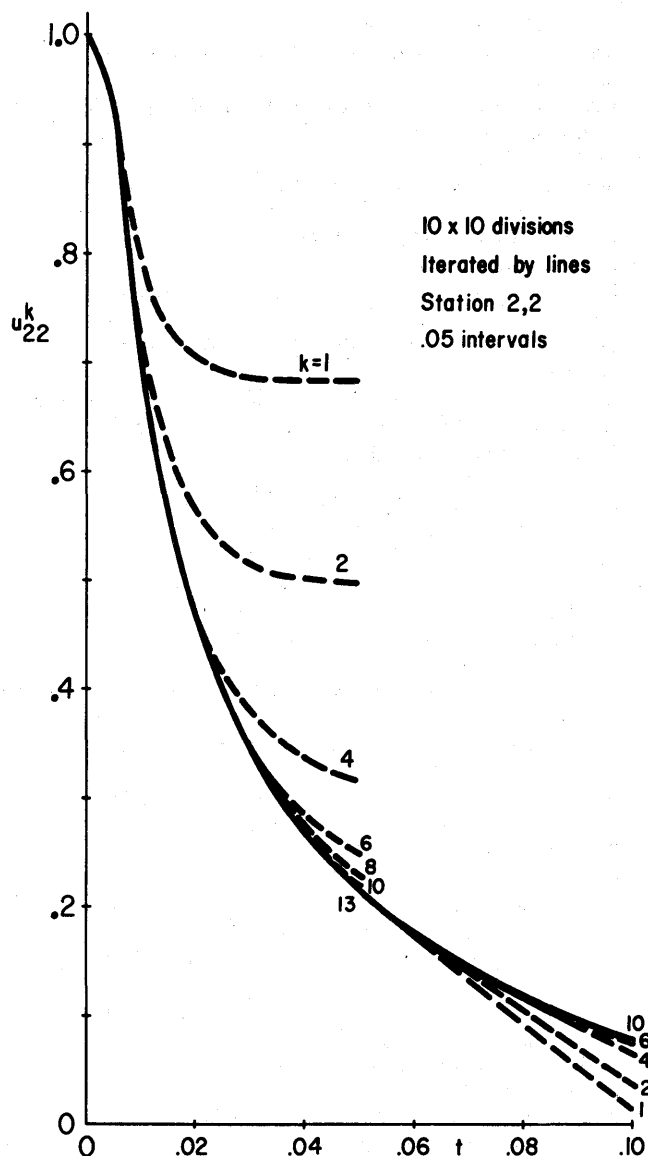


FIGURE 10—Two dimensional diffusion equation, 10×10 , iterated by lines

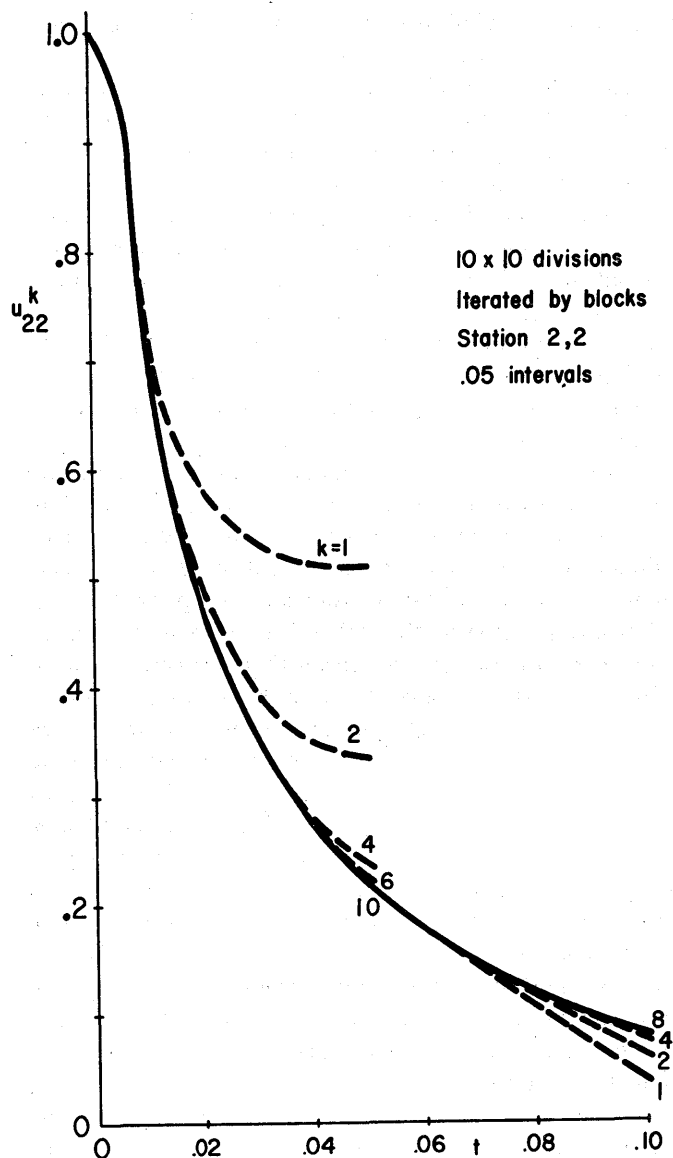


FIGURE 11—Two dimensional diffusion equation, 10×10 , iterated by blocks

actual hybrid system. Consider the 10×10 case. Iteration using lines requires 9 A/D converters for sampling and 18 D/A converters in playback; iteration using 3×3 blocks requires 9 A/D converters, but only 12 D/A converters. If digital print-out is not required for the internal point(s) in the blocks, the number of A/D converters is reduced to 8 for each interior block. Indexing for the block sequence, however, will be more complicated.

CONCLUSION AND DISCUSSION

It has been demonstrated by means of several examples that iterative solution of CTDS equations is a promising approach to the hybrid computer solution of certain

classes of partial differential equations. Many questions need to be investigated further.

The solution of CTDS equations having many stations involves many A/D and D/A transfers and large storage capacity. However, arithmetic operations on the digital data are generally not required. This suggests that the standard hybrid computer configuration which is based on a general purpose digital computer may not be optimum. Instead, it would be better to use a small digital computer for control and provide a special parallel in-parallel out memory coupled to a large A/D and D/A conversion system. It should be noted that the technique of iterating on sub-intervals should reduce appreciably the memory requirements.

With respect to the details of implementing the iterative procedure many variations are available. The possibility of iterating by blocks or by lines in the two dimensional problem has been indicated above. But many other patterns of iteration are possible, e.g., alternating between the "black" and "white" squares of a checker board or the interleaving of successive lines. These additional schemes may have advantages in speed of convergence, reduction in the number of data transfers, effective utilization of the analog elements, etc. As has been indicated in an earlier section the iterative method has a strong similarity to methods used in the solution of elliptic equations. This suggests the possibility of using over-relaxation techniques. This has been tried on the $6\frac{1}{2}$ station diffusion problem. The best over-relaxation factor obtained was 1.1 and the improvement on the number of iterations was not very impressive. For larger problems (more stations and longer period of time) over-relaxation may have a more beneficial effect.

It is possible to prove convergence of certain of the iterative methods under reasonably mathematical conditions. Also there is reason to believe that analysis can yield effective comparisons of convergence rates for different iterative methods. These results will be reported in a subsequent paper.

In the case of both the diffusion and wave equation it can be seen from the solutions shown in the figures that each additional iteration adds approximately a constant time increment over the previous solution before the solution diverges from the true CTDS solution. Thus in Figure 2 each 10 iterations extend the time for accurate solution by about 0.23 units in t , whereas in Figure 8 each 4 iterations extend the time for accurate solution by about 0.5 units in t . In fact it has been found in the cases studied thus far that, after a few beginning iterations, the added time increment for accurate solution following each iteration is roughly equal to $(\Delta x)^2$, assuming dimensionless distance and time variables. This allows at least a crude estimate of total hybrid

computational time for problems falling in the category of those studied here.

The potential payoff of the methods described appears to be greatest in complex nonlinear problems where the analog system performs the nonlinear computations as well as the integrations. Additional computer experiments need to be performed in order to verify this potential.

ACKNOWLEDGMENT

The authors wish to thank Professor Elmer G. Gilbert and Dr. Edward J. Fadden for their suggestions and help in the preparation of this paper.

BIBLIOGRAPHY

- 1 R VICHNEVETSKY et al
Analog and hybrid computers in the solution of partial differential equations
Electronic Associates Inc Research and Computation Div
Princeton New Jersey
- 2 W BRUNNER
Serial solution of partial differential equations on an electronic analog computer
EAI PCC Report 180 June 21 1961
- 3 R MORONEY
Heat flow in a rod solved with continuous space and discrete time variables
MIT Electronic Systems Lab DSR 9128-WP-1 Oct 20 1962
- 4 D R ELLIS
Analog solution of the diffusion equation with continuous-space and discrete-time variables
MIT MS Thesis Dept of EE Aug 1963
- 5 H S WITSENHAUSEN
Hybrid solution of initial value problems for partial differential equations
MIT MS Thesis Dept of EE Sept 1964
- 6 R BENZ
A hybrid computer solution of certain field problems
Boeing Company Internal memo Jan 1965
- 7 S K CHAN
A study on certain hybrid techniques for the solution of the heat equations
MIT Electronics Systems Lab DSR 9128 Memo 11 Dec 1965
- 8 R VICHNEVETSKY
A new stable computing method for the serial hybrid computer integration of partial differential equations
Proc SJCC 1968 pp 143-150
- 9 R M HOWE V S HANEMAN
Solution of partial differential equations by the difference technique using electronic differential analyzer
Proc IRE v 41 Oct 1952
- 10 S MAX
A preliminary investigation of the analog-digital solution of partial differential equations
MIT Electronic Systems Lab R-108 March 1961
- 11 G S STUBBS
Application of time sharing techniques to simulate dynamic processes associated with fluid stream
EAI Hybrid Computer Course 1962 Lecture 13
- 12 T MIURA J IWATA

- Time-sharing computation utilizing analog memory*
 Ann AICA v 5 n 3 July 1963 pp 141-149
- 13 K CHUANG L F KAZDA B WINDEKNICHT
A stochastic method of solving partial differential equations using an electronic analog computer
 Project Michigan Report 2900-91-T Willow Run Lab U of Michigan June 1960
- 14 W D LITTLE
Hybrid computer solutions of partial differential equations by Monte Carlo methods
 PhD Thesis Univ of British Columbia Oct 1965
- 15 A C SOUDACK W D LITTLE
An economical hybridizing scheme for applying Monte Carlo methods to the solution of partial differential equations
 Simulation v 5 n 1 July 1965 pp 9-11
- 16 G A KORN
Hybrid computer Monte Carlo techniques
 Simulation v 5 n 4 Oct 1965 pp 234-247
- 17 W D LITTLE
Hybrid computer solutions of partial differential equations by Monte Carlo methods
 Proc FJCC 1966 pp 181-190

QUIP—A system for automatic program generation

by F. C. BEQUAERT

*International Business Machines Corporation
Cambridge, Massachusetts*

INTRODUCTION

Extensive programming effort is expended in the rewriting of minor variations of already existing programs. One method of reducing this redundant effort is to devise a method for automatically generating programs for the solution of any of a class of problems. One such technique is illustrated in Figure 1. A generator program produces, in response to user inputs, an output program for the solution of a particular problem. User inputs to the generator can be either in the form of commands in a special purpose language (e.g., macro statement) or responses to computer-generated queries. The output code produced by such a system is free of coding errors usual in handwritten programs. The output program is a free-standing entity which can be used for production data processing without user intervention. The literature contains a number of references^{1,2} to this type of approach.

There are two basic design goals that will make this approach economically attractive. First, techniques should be developed for minimizing the

effort required to produce a working generator program. Second, the generator program produced should be as simple to use as possible. The user should also be able to use the generator program in an interactive mode from a remote terminal.

This paper describes QUIP (for Query Interactive Processor), a system proposed for achieving these goals.

The query interactive processor

Figure 2 is a diagram of the QUIP system. The program operates in conjunction with a number of files and a terminal user. The program file contains a number of generator programs written in the special purpose QUIP language.* Any

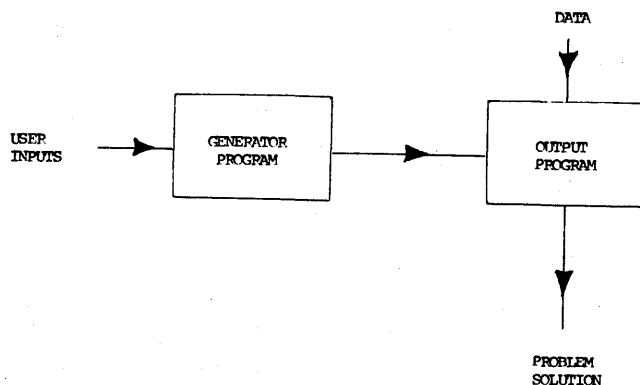


FIGURE 1—Program generator approach

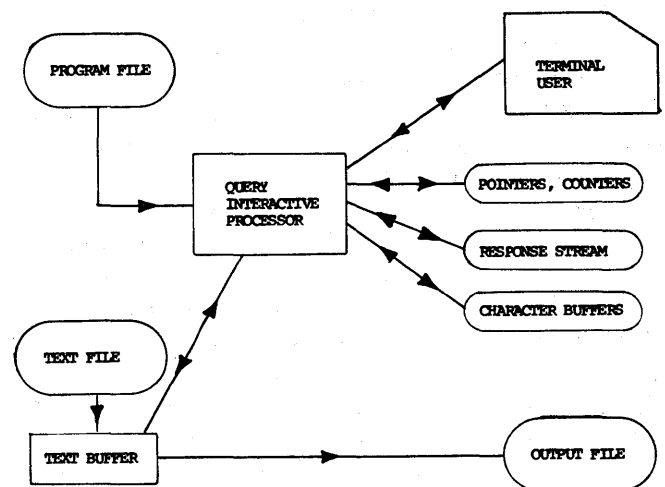


FIGURE 2—General diagram of QUIP

*In this document "QUIP language" always refers to the special purpose language for writing generator programs.

one of these programs may be called for operation by a terminal user request. Associated with each generator program is a text file which is loaded into the text buffer. This file represents the basic text of the program segment that is to be generated by the generator program. The generator program asks the terminal user a series of questions. Based on the responses to these questions, the program modifies the text in the text buffer and transfers portions of this text to the output file. As various generator programs are called up and executed, a program is built up in the output file. When completed, this program may be compiled, loaded and executed. The Response Stream, a sequential record of all user responses to QUIP generated queries, is of particular interest. It represents a complete record of operations and may be used with the generator programs and associated text files to regenerate the output file.

In addition to the query mode described above, the QUIP system may operate in a macro-mode in which the user specifies by macro statements the actions to be taken within the text buffer. The complexity of the language used for these statements is limited only by the complexity of the generator program which the programmer is willing to write. Special commands to assist the programmer in decomposing macro statements are provided in the QUIP language.

A regeneration mode is also possible in QUIP, in which the user's responses to QUIP queries are used to generate an equivalent macro statement for tutorial purposes.

The QUIP system is not limited to the generation of programs. The system has a general text manipulation capability with many potential uses. One interesting use of the system would be as an "inserted processor" to enhance the capability of a conversational system. User statements are read by QUIP one line at a time for such an operation. If a line is a valid statement in the conversational system in use, it is fed directly to the conversational processor. If the line represents a statement in the extended language, valid statements are generated and fed to the conversational system. The terminal user has a conversational system with an extended capability beyond that of the base system, while the base system deals with its input from the QUIP program exactly as it would deal with direct terminal inputs.

The QUIP language

The QUIP language is a PL/I-based language designed to facilitate the writing of QUIP generator programs. A generator program written in the QUIP language might be handled in one of two ways. (1) It could be fed to a PL/I pre-processor which would generate valid PL/I code as output. This output could then be compiled and used as a generator program. (2) It could be interpreted at execution time by a resident QUIP interpreter. For the purposes of this paper, either of these approaches could be selected.

Basic program structure

A generator program written in the QUIP language is built around a flow diagram that represents the actions to be taken in the interchange between the user and the generator program. The conditional execution statements described below provide a means by which a program function may be placed in a program segment. The function will be executed only if the specified condition state is set.

The basic program functions of a generator program are:

- a. Basic decision logic based on the contents of the response stream
- b. Generation of queries to a user and handling of his replies
- c. Text file manipulation.
- d. Transfer of data to the output file
- e. Macro statement decomposition
- f. Macro statement regeneration
- g. Handling of user requests for special information

Language commands

The commands available in the QUIP language can be summarized as follows:

1. Standard PL/I commands or, for an interpretive system, some subset of PL/I.
2. The conditional execution command. Prefixing any command with the symbol \$n (where n is an integer) specifies that command as conditional. It will be executed only when the state numbered n has been previously set for execution by a SETEXECUTION n command. Conditional execution commands represent a powerful programming tool for situations where a num-

ber of functions embedded within the body of a program must be selectively executed.

3. The ASK command which provides a simple method for generating user queries and automatically handling user responses. A user's response to an ASK is, for example, automatically embedded in the correct location within the response stream.
4. Character search commands that simplify the search for specified character strings within buffers. Pointers are provided to permit easy identification of the position of characters within a buffer.
5. String move commands to simplify the movement of character strings between buffers.

Use of QUIP language

The following example of a portion of a generator program written in the QUIP language illustrates the capabilities of the language. In this example, the generator program determines, from user responses, an expression to be embedded in a line of output text.

The program segment is as follows:

```

$2 A3: 'WHAT IS THE EXPRESSION?', C;
/* REQUEST USER RESPONSE */
$3 LP1 = LP1 + 1; /* ADVANCE A LINE
  POINTER */
$5 IS(IP) = CB4(CP2 + 1 : CP1 - 1);
/* EXTRACT AN EXPRESSION FROM
  MACRO */
$7 SETEXECUTION 3,5; /* SETEXECUTION
  STATES */
$5 IF IS(IP) = '?' THEN SETEXECUTION
  2,7; /* CHECK FOR ? */
$7 GO TO A3;
$3 TB(LP1) = CB2; /* LOAD NEXT LINE
  OF TEXT BUFFER */
$3 REPLACE 'DUM2' WITH IS(IP), LP1; /*
  MODIFY TEXT */

```

Conditional execution prefixes are used on all statements. The interpretation of these symbols is:

- \$2 Specifies statements which make user queries
- \$3 Specifies statements which modify the text buffer
- \$5 Specifies statements which extract information from a macro statement
- \$7 Specifies statements which handle the case

where a "?" is encountered in a macro statement

In order to query a user directly, this program segment would be executed with condition states 2 and 3 set. (These states would be set with a previous QUIP language statement SET-EXECUTION 2,3.) Only the statements with either a \$2 or \$3 prefix would be executed.

Program operation would then be as follows. The first statement presents the user with the query WHAT IS THE EXPRESSION? The user's response is checked for validity and then placed in the current position in the response stream.

The second statement advances a line pointer (LP1) to the next line in the text buffer. The text buffer is being used for building the output text.

As only condition codes 2 and 3 are set, all statements down to the next to the last are skipped. The next to the last statement transfers a character string from a temporary buffer (CB2) into text buffer line number LP1 (TB(LP1)). The last statement replaced the dummy variable DUM2 in line LP1 of the text buffer with the current item in the response stream (IS(IP)).

Thus this program segment asks a user a question and produces in the text buffer a character string in which a dummy variable is replaced by the user's response.

Now consider the case where it is desired to extract the expression from a user-generated macro statement. Assume that the macro statement is stored in temporary buffer CB4. In this case conditional states 3 and 5 are set. The first statement with a code of \$2 is now skipped. The second statement is executed as before. The third statement extracts the appropriate expression from the macro text (in buffer CB4) and places this expression in the input stream (IS(IP)). The variables CP2 and CP1 in this expression are buffer pointers that have been previously set to point to the characters immediately in front of and following the expression to be extracted.

The next three statements handle the case where a "?" is encountered in the macro being decomposed. In this case the 3 and 5 execution states are reset and the 2 and 7 states are set. On encountering the \$7 statement, a branch to A3 occurs and the user is asked for the expression in question. His response replaces the "?" in the input stream. The \$5 statement is then ignored, and the \$7 statement following resets

the execution states for continuation of macro decomposition.

The final two statements operate as previously described, replacing a dummy variable (DUM2) with the expression in the response stream.

Typical program operation

The following describes a hypothetical interactive session using a QUIP generator program designed to produce a program for radar data reduction.

The system user is assumed to be an engineer who understands how to block diagram a simple program, how to write arithmetic statements in standard PL/I (or FORTRAN) notation, and how to describe arrays (tables) and elements of arrays of data in PL/I (or FORTRAN) notation. He is seated at a typewriter terminal which operates in an interactive mode with a QUIP generator program.

Operation with generator program

The user at the terminal logs in to the QUIP system. The computer types: (In all subsequent discussion computer outputs are in italics.)

What generator do you want?

The user types REDUCE to call up the data reduction generator program. The computer types:

1. Do you have a file dictionary?

In this system, a data file (e.g., a data tape) is described by a dictionary that relates data names to bit positions in the file. Once generated for a tape of a particular format, the dictionary may be reused to assist in generating any data reduction programs that use this tape without requiring the user to have any detailed knowledge of tape format. In this case the user types:

YES

The computer responds:

2. Name?

The user responds with the name of the correct dictionary file. The computer then asks:

3. Do you want a dictionary listing?

The user responds:

YES

The computer types variable names for referenc-

ing data in the file and their definitions as follows:

TIME Time of day in seconds
TELEV True elevation angle in degrees
CELEV Commanded elevation angle in degrees
VELEV True elevation angular velocity in degrees/second

The user now has the required information, so he hits the ATTENTION key on the keyboard to stop this output.

The computer responds:

4. What is the next operation?

Each "operation" will generate a segment of output code that performs one function. The user types:

READ

The following conversation then takes place:

Entry name = READ1

5. What is the maximum number of items in a table and how many tables are there?

200, 2

7. What are table names?

VELO, DANG

The conversation continues:

9. Enter expression for VELO as a function of NROW

VELEV (NROW)

10. Enter expression for DANG as a function of NROW

TELEV (NROW)-CELEV (NROW)

11. What is next operation?

A section of output code has now been generated that will read from the data file into two tables 200 consecutive values of angular velocity in elevation and the corresponding differences between true and command elevation angles.

The user now wishes to specify the next operation, namely the plotting of the data. The conversation is as follows:

PLOT

12. Do you want a single page plot?

YES

13. *What variables do you want plotted on the x and y axes?*

VELO, DANG

15. *Do you want to set plot size?*

YES

16. *Minimum and maximum values of x = ?*

-2, 2

18. *Minimum and maximum values of y = ?*

-.5, .5

20. *How many points do you wish plotted?*

200

End of PLOT generation

21 *What operation do you want?*

The necessary inputs for the plot program segment have now been specified. The user thinks that a shorter macro form may be available for answering the questions concerning the plot routine. For future reference he wants an example of the use of this macro. He now types:

RECONSTRUCT

The computer responds:

```
MACRO FOR PLOT OPERATION
PLOT (ONEPAGE=YES, VELO,
DANG, SETEDGE=YES, -2,2
-.5,.5,200)
```

The above statement represents a shorthand (or macro) method of encoding the same information that a user would normally provide in response to queries from the computer. As a tutorial aid, the computer gives the user a model macro statement with his previous responses encoded in the correct format within the macro.

In using the macro form of input, it is possible to provide only partial information and the system will automatically request the missing information. Thus if in response to question 11 above, the user typed

```
PLOT(ONEPAGE=YES,?,?,SETEGE
=YES,-2,2,-.5,.5,200)
```

the computer would respond:

What variable do you want plotted on the x and y axes?

and the user could specify this information.

Continuing with the previous example, the user has now completed the necessary program except for termination. He now types:

GOTO, READ1

This inserts the correct branch to loop the program back to begin file reading again.

Typing END terminates the program generation.

Up to this point it has been unnecessary to consider the language in which the output program was written. From the point of view of the user, this is irrelevant. The QUIP system is designed to handle any text. In the example above, it might output an assembly language program or PL/I source statements. Although the generator program would be different in these two cases, the user queries and his required responses would be essentially identical.

Assume that the output program is PL/I source statements. The generated program could immediately be compiled, loaded, and executed in a batch mode and sample output made available to the user.

Suppose that the user now wishes to make a program change. He re-enters the system, and specifies the name of his program. He is now able to specify changes by requesting previous queries by number. Specifying, for example:

CHANGE 18

would cause the system to ask that number question again and permit modification of the minimum and maximum values of the y axis on the output plot. The program would then automatically regenerate the output program with the necessary modifications.

How effective is this type of programming? With a system⁸ considerably cruder than the one described here, the author was able to generate and debug, in a single three-hour session at a console, a 300 statement FORTRAN data reduction program.

Pilot model of QUIP

A rudimentary version of a QUIP system has been written in PL/I to demonstrate the capabilities of the QUIP approach. The program was initially written for batch operation, but then converted to operate in an interactive mode under the Conversational Programming System (CPS) on the IBM System 360 Model 50.

A fixed format query language is used in this

system for writing generator programs. The pilot program interprets query language statements at execution time and, under direction of these commands, makes queries of a user and performs the requisite text manipulation to produce an output text file.

The capability of this system to generate error-free PL/I source programs as a result of a terminal user's responses to computer generated queries has been demonstrated.

CONCLUSIONS

In the introduction, two goals for the design of a generator program were given. How well does the QUIP system achieve these goals?

- (1) The QUIP language offers a number of tools to assist in the writing of generator programs. Of particular power is the conditional execution statement.
- (2) QUIP attacks the problem of a user's learning to use a generator program by building a number of tutorial aids into

the mainstream of program operation. Ideally, such a system would provide the user with any required information concerning program operation. Such an ideal is limited by the effort that the writer of the generator program is able to expend in providing information files. Hopefully, with a properly designed system there will be no limitations due to a lack of interconnections into the system which permit the user simple access to information.

REFERENCES

- 1 A S GINSBERG et al
Programming by questionnaire
The Rand Corporation RM-4460-PR AD 613-976 April 1965
- 2 I C PYLE
Data input by question and answer
Communications of the ACM Vol 8 Number 4 April 1965 p 223
- 3 F C BEQUAERT
RPL A data reduction language
Proceedings SJCC 1967 p 571

The XPL compiler generator system*

by W. M. McKEEMAN

University of California
Santa Cruz, California

and

J. J. HORNING, E. C. NELSON
and D. B. WORTMAN

Stanford University
Stanford, California.

Objectives and results

The development of the system described here was originally motivated by the need to develop a good student language compiler for a large IBM System/360. An examination of the tools and methods available caused us to establish the subgoal of developing a translator writing system in which we could prepare the student compiler. In our opinion, then and now, the total effort was smaller, and the end product better, for the combined project than for the original project using previously available tools.

In the belief that this system will be useful to others who are interested in language development, we have extended it beyond our immediate application and are making it generally available. We hope that this will help dispel the myth that the generation of new compilers is necessarily a long and arduous task, consuming many man-years of effort. Students in the systems programming course at Stanford have been using this method to write operating translators in three to six weeks.

We made two major decisions in designing our TWS: the choice of a compilation method and the choice of a language in which to express the translators. We selected a bottom-up parsing algorithm rather than recursive descent since the IBM System/360 is ill-suited to recursion. In any case, code generation is easier with bottom-up methods. The particular method used is an extension of several which have been previously reported [Floyd 63] [Wirth and Weber 66] [McKeeman 66].

Compilers have been written in assembly languages, "general purpose" high-level languages, and special pur-

pose "compiler compiler" languages. We rejected the first as being completely beyond our available manpower to program, debug, or maintain. Our choice was a compromise between the second and third options. The language XPL contains many features which are sufficiently "close to the machine" for efficient translators, and omits features not required for translation, yet retains much of the flexibility and style of PL/I.

Our TWS consists of four major programs: XCOM, XPLSM, ANALYZER, and SKELETON. XCOM is a one-pass compiler from XPL into System/360 machine language. Since it is written in XPL (it consists of about 3500 source cards) it is self-compiling on a System/360.

XPLSM, the interface between XPL programs and OS/360, is a small assembly language program which handles program loading and input/output using OS/360 data management methods.

ANALYZER accepts a BNF* grammar, checks it for compatibility with the parsing algorithm, and prepares parsing decision tables in the form of XPL declaration statements with the initial attribute. The program is written in XPL (1400 cards).

decision tables in the form of XPL declaration statements with the initial attribute. The program is written in XPL (1400 cards).

SKELETON is a proto-compiler, which when supplied with tables from ANALYZER becomes a table-driven syntax checker, to which code emitters may be added to create a compiler.

*This work was supported by the Stanford University Computation Center, Campus Facility.

*Backus-Naur Form or Backus Normal Form

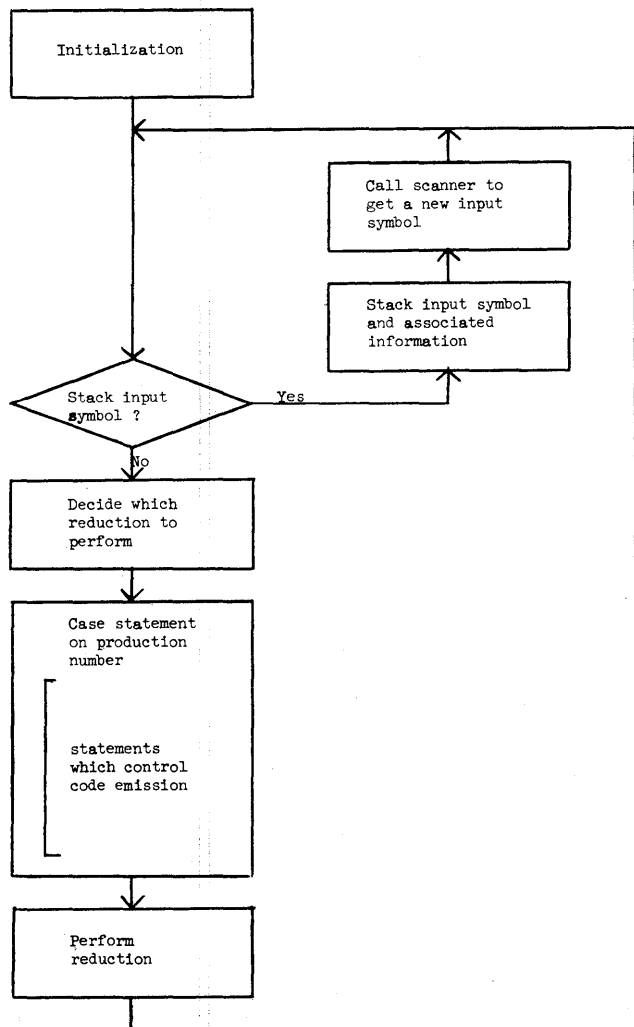


FIGURE 1

The method.

The translation method depends upon the well-known fact that the phrase-structure of a BNF-described source language can be used to direct the production of object code [Wirth and Weber 66]. All of the following phrases apply to this method:

table-driven,
syntax-controlled,
bottom-up,
one-pass,
left-to-right,
no-backup, and
canonical parse.

The general parsing algorithm, depicted in Figure 1, consists of two cycles corresponding roughly to the actions of:

- (1) Recognizing and stacking the basic symbols (identifiers, numbers, operators, etc.) of the source language.
- (2) Substituting a phrase class name for a phrase and emitting the associated object code.

Within the inner loop there is a call on the text scanning procedure. The responsibility of the scanner is to recognize the symbols on the lowest level of detail in the language (identifiers, reserved words, etc). The scanner is a source language dependent routine that is programmed with the text manipulation features of XPL. As a result the most frequent decisions (those on individual characters) are not made by the general parsing algorithm for reasons of efficiency.

The success of the general parsing algorithm depends upon how generally, and how efficiently the decisions indicated in Figure 1 can be made. The stacking decision is made on the symbol pair consisting of the top of the stack (where any symbol is possible) and the input symbol (where only terminal symbols can occur). There are four possibilities:

- (1) The input symbol is stacked and the question is asked again for the next symbol.
- (2) The input symbol is not stacked, instead a reduction is made and (perhaps) some code emitted.
- (3) The symbol pair does not contain enough information upon which to make the decision.
- (4) A syntactic error in the source text is discovered and error recovery initiated.

When the pair fails to contain enough information (3), an auxiliary table of triples, which correspond to the two top elements of the stack and the input symbol, is consulted. Since the number of decisions which must be made using triples is small for many useful grammars, the necessity for searching a table need not slow down the compiler appreciably. The current XPL grammar has 89 symbols, 42 of which are terminal so that the symbol pair table has 37382-bit entries. Of these 3175 represent illegal combinations (syntax error), 239 indicate that the input symbol should be stacked, 306 indicate that it should not be stacked, and 18 represent conflicts for which the decision must be made using triples, of which there are 165.

Another series of tables, equivalent to (1,1) context [Floyd 64], is used to decide which reduction to make once the reducible string is located on the top of the stack. These tables are automatically produced from the BNF description of the source language, as described in section 4.

It can be shown, based upon these tables, that if the general parsing algorithm proceeds to normal completion (i.e., to the point where all the source text has been

processed and only the goal symbol of the language remains on the parse stack), then the parse obtained is correct. Conversely, any correct program will parse to the goal symbol. [McKeeman, *et al.* (to be published)]

The production of code is done by a series of statements written in XPL and invoked in the case statement (Figure 1) as each reduction is made. Information required, such as addresses, operand types, etc., is kept in stacks parallel to the parse stack or in a symbol table. Except as XCOM serves as a model, the system has no special aids for the production of object code.

The language XPL

Translation methods are not "all alike," and we do not suggest that all translators should be written in the same language. XPL was designed for a particular class of translators that require very elementary kinds of data structure and control. Aside from the problem of text manipulation almost any existing procedural language is sufficient and in fact, considering our bias towards simplicity, overly elaborate.

There are no startling innovations in the following description. One reason is that the efficiency of translators is an important concern, so constructs reflect the structure of the IBM System/360. A second reason is that the technique used in the recognition phase of the translator was first presented [Floyd 63] with the explicit purpose of being efficiently implementable on existing conventional machines.

Before getting into the details, we can describe the basic concepts which must underlie any computer language design. We must have:

- (1) selection by name (the idea of a set);
- (2) selection by position (the idea of an ordered set);
- (3) sub-definition (definition of something in terms of its components);
- (4) iteration;
- (5) meaningful primitive operations; and
- (6) control over the allocation of computer resources.

We expect translators particularly to be maintained, debugged, modified, studied and copied by many people besides the original authors. So we insist that the language be conducive to *readable* statements of translators.

We want the user to be able to easily follow the logic of a program at two levels. First, he should be able to quickly determine the major structural units, their intent, and the flow of control between them. Second, it should be convenient for him to determine precisely what operations are being performed on the data at any given point. It should be possible to break the program into units both logically (procedures) and visually (paragraphing).

The programmer should be able to *access* the memory efficiently by operating on the directly addressable quanta of memory (characters, words, etc.) We also require the ability to describe and operate on the basic units of information (bits) both for the generation of arbitrary object code and for efficient packing and unpacking of various tables.

Four kinds of data are particularly useful for translators: bits for recording yes-no decisions; strings of bits for packing information; integers for arithmetic; and strings of characters to represent text, such as the source input, the output program listing, diagnostic messages, and identifiers.

Many of the operations within a translator require the evaluation of integer arithmetic expressions.* Our language must include the arithmetic operations of addition, subtraction, multiplication, division and remainder. We also need the logical operations (&, |, ¬, and shifting) on either single bits (logical expressions) or on groups of bits (masking operations). Within each data type we must have comparison operations (<, ≤, ±, etc.) and we need to convert data from one type to another (for instance, from integers to character strings for output).

Text (strings) presents additional problems. Input text must be tested and separated into constituent strings of varying length (e.g., identifiers). The output listing must generally be built up by joining shorter strings. The basic string operations are those of substring selection and concatenation, as well as the extraction of internal codes for the characters.

We would, of course, expect the writer of a translator to be able to provide for emitting the code for any machine instruction. What is frequently true, is that he may want to cause the execution of any instruction during compilation. The constraints placed upon the translator by the machine itself are already severe; it is unwise to compound them by further restricting the translator to some subset of the machine's capabilities. No single translator is likely to use the full instruction repertoire of a large scale computer but there is no instruction that we can classify *a priori* as unusable. The simplest example of this kind of requirement is the floating point operations. The compilation mechanism itself has no need for them, but if the language being compiled has floating point values, it is convenient to use the floating point hardware to convert input numbers. Rather than add a type to the compiler language, we make a general provision for instruction-by-instruction, in-line execution of arbitrary machine code.

A basic concept is *selection from an ordered set*. For

*Even in translation of languages involving floating point numbers, floating point expressions are not frequently used.

ease of expression and efficiency of operation both data and instructions are usually arranged so that the appropriate element can conveniently be selected from a set of alternatives. Arrays with one subscript position are the simplest form representing this construct. Out of arrays we can build stacks (for the syntactic recognition algorithm) and tables (for symbols, types, forward references and the like) and pointer spaces (for lists, sorting, etc). We do not find sufficient use for more complex constructs (multiple subscripts, queues, explicit lists, PL/I structures) in our translators to justify their implementation. Similarly, it is possible to select among alternative instruction sequences dependent on the value of an expression. In the most common case the expression will be logical (two-valued) in nature and the set of instruction sequences will contain two elements. The usual representation for this action is the IF-THEN-ELSE conditional statement and the CASE statement. [Wirth and Hoare 66]

Repetition of selected sequences of operations is another basic requirement. Two forms of repetition seem particularly important. We may wish to repeat the operations a specific number of times (often with specific values for a controlled variable) or we may wish to repeat them as long as a particular condition is satisfied.

Transfer of control also takes two forms. We may temporarily transfer to a subprogram (possibly supplying it with some parameters) which will return to the original instruction sequence (procedures and functions). Occasionally, however, it is desirable simply to enter the instruction sequence at some other point, with no provision for return (GO TO together with labels).

We chose PL/I as a base language for several reasons. It contains most of the features we require. It is widely known and will probably be the next dominant programming language. The growth of the PL family of languages will require the development and refinement of PL.

There are several distinct reasons for not choosing PL/I itself for the description of translators. First, it does not contain quite all the features we require. More important, however, is that it is a very large language and the machine we are using (IBM System/360) is ill-suited to its full implementation. Consequently all existing compilers for the full PL/I are very large complex programs which produce code unacceptably inefficient for this application. For the same reasons, we do not recommend attempting a compiler for full PL/I written in XPL.

We present here several program fragments written in XPL to give the reader a better feel for the language.

1) A simple indirect sort:

```
/* SORT IN ASCENDING ORDER */
K, L = ND;
DO WHILE K <= L;
    L = -1;
    DO I = 1 TO K;
        L = I - 1;
        IF (DESCRIPTOR(DX(L)) & MASK) > (DESCRIPTOR(DX(I)) & MASK) THEN
            DO;
                J = DX(L); DX(L) = DX(I); DX(I) = J;
                K = L;
            END;
        END;
    END;
```

2) Code emission procedure from a translator:

```
EMIT-NAME:
PROCEDURE (L,N);
    /* L IS THE LEXIC LEVEL, N IS THE ORDER NUMBER */
    DECLARE L FIXED, N FIXED;
    IF (L < 3 | L = LL) & N < 16 THEN
        DO; /* A SHORT NAME */
            IF L = LL THEN L = 3;
            CALL EMIT(SHL(L,4) | N);
        END;
    ELSE
        CALL EMIT_3(NAME_OP,L,N); /* A LONG NAME */
    END EMIT-NAME ;
```

3) Instruction interpretation routine from an interpreter:

```
/* CASE 1 SHORT NAME OPERATOR */
DO;
    SP = SP+1; /* PUSH THE STACK */
    /* DECODE LL AND ON VALUES */
    /* ENCODING PICKED FOR CODE DENSITY */
    DO CASE SHR(OP,4);
        RUN_VALUE(SP) = DISPLAY0 + (OP & "OF");
        RUN_VALUE(SP) = DISPLAY1 + (OP & "OF");
        RUN_VALUE(SP) = DISPLAY2 + (OP & "OF");
        RUN_VALUE(SP) = DISPLAYLL + (OP & "OF");
    END;
    RUN_TYPE(SP) = INDIRECT_TYPE;
END;
```

4) Printing the date:

```
DECLARE EJECT_PAGE LITERALLY 'OUTPUT(1) = 1',
        DOUBLE_SPACE LITERALLY 'OUTPUT(1) = 0';
PRINT-DATE:
PROCEDURE (MESSAGE, D);
    DECLARE MESSAGE CHARACTER, D FIXED;
    DECLARE MONTH(11) CHARACTER INITIAL ('JANUARY', 'FEBRUARY',
        'MARCH', 'APRIL', 'MAY', 'JUNE', 'JULY', 'AUGUST',
        'SEPTEMBER', 'OCTOBER', 'NOVEMBER', 'DECEMBER'),
        DAYS(11) FIXED INITIAL (0, 31, 60, 91, 121, 152, 182,
        213, 244, 274, 305, 335);
    DECLARE YEAR FIXED, DAY FIXED, M FIXED;
    YEAR = D/1000 + 1900;
    DAY = D MOD 1000;
    IF (YEAR & 3) = 0 THEN IF DAY > 59 THEN DAY = DAY + 1;
    M = 11;
    DO WHILE DAY <= DAYS(M); M = M - 1; END;
    OUTPUT = MESSAGE || MONTH(M) || ' ' || DAY-DAYS(M) || ', ' ||
        YEAR || '.';
```


access scratch storage on direct access devices. FILE (I,J) specifies the Jth record on the Ith file. FILE may appear on the left or right side of an assignment statement causing a device-dependent number of bytes to be written or read from the selected record and file.

For example:

```
DECLARE X (900) FIXED;
/* ASSUME 2311 DISKS*/
X = FILE (1,K);
FILE (3,K + 1) = X;
```

transfers a record from one file to another.

Generalizing the selective capability of the IF statement, the CASE statement allows the selection of any one as sequence of statements. The expression following CASE is evaluated and used as an index to select one of the statements in the group body counting from zero, starting from the top. For example:

```
I = 2;
DO CASE I;
  OUTPUT = 0;
  OUTPUT = "1";
  OUTPUT = 2;
  OUTPUT = '3';
END;
OUTPUT = 4;
```

will cause 2 and 4 to be printed.

The ability to execute arbitrary machine instructions is provided by the pseudo function INLINE. The arguments of this function are placed directly into the code stream at the point of the function call. An option is provided whereby XCOM will calculate the proper base and displacement fields from the names of the variables. An example of the use of INLINE to do floating point arithmetic in an interpreter written in XPL is given below.

```
DO; /* FLOAT ARITHMETIC */
  /* FIRST LOAD THE FLOATING REGISTER WITH BV */
  CALL INLINE ("78", 0, 0, BV); /* LE 0,BV */
  DO CASE OP - ADD_OP; /* NOW EXECUTE A FLOATING INSTRUCTION */
    CALL INLINE ("7A", 0, 0, AV); /* AE 0,AV */
    CALL INLINE ("7B", 0, 0, AV); /* SE 0,AV */
    CALL INLINE ("7C", 0, 0, AV); /* ME 0,AV */
    CALL INLINE ("7D", 0, 0, AV); /* DE 0,AV */
  END;
  /* NOW STORE THE RESULT IN BV */
  CALL INLINE ("70", 0, 0, BV); /* STE 0,BV */
END;
```

One should note which capabilities of the machines are missing from XPL. By their absence we are marking

them as irrelevant to our translation process and thereby simplifying both our language and its translator. It is worthwhile to note that the addition of features to a language is not a linear process as far as the translator is concerned. Constructs interact with each other, and the addition of a *single feature* may, in a bad case, *double* the size of the translator. The potentially exponential growth of translator size with increasing language complexity has two important implications. First, a translation method which works well for deliberately simple test languages or machines may fail for practical languages or existing machines. Second, an elaborate language (we have in mind the full PL/I) may need a much larger translator than needed by several smaller languages which have in aggregate the same features.

Programming in BNF.

Just as we need a language for the description of translators, we need a metalanguage for the description of languages to be translated. BNF has become the most widely used formal metalanguage for programming languages; it is a concise, readable, and unambiguous way to express any context-free grammar. We have adopted it for use in our system.

Each language which has a context-free grammar has arbitrarily many context-free grammars. The art of programming in BNF is to impose additional criteria to select an optimal grammar for use in a translator. Some general requirements are:

- (1) The grammar must be compatible with the parsing algorithm used by the translator, which implies (2).
- (2) The grammar must be unambiguous, so that each sentence will have a unique phrase structure.
- (3) The structure assigned to each sentence should correspond to the "intended" interpretation of the language.
- (4) The grammar should permit easy association of code generation with the canonical parse.

Requirements (1) and (2) concern properties which a given grammar either does or does not possess, while (3) and (4) are (at the present state of the art) still matters of judgment and degree.

One program of our system (ANALYZER) builds decision tables for the parsing algorithm from a BNF grammar. In the process, it determines whether the grammar meets condition (1), and if not, gives error diagnostics to pinpoint the problem. Basic to the ANALYZER algorithm is the concept of "running the parser backwards." Instead of successively reducing a string of text to a goal symbol, it successively produces text from the goal symbol, tabulating all the decisions which would be required to parse the produced texts.

Expansions continue until all possible decisions (for the chosen complexity of tables) have been recorded.

Ideally ANALYZER would prepare correct tables for every grammar which users fed it. However, people don't write good grammars. For one thing, they usually start out ambiguously (even the Algol 60 Committee created an ambiguous grammar). After the ambiguities

have been removed, other changes may be necessary to make the grammar compatible with the specific parsing algorithm.

One criterion in designing our parsing algorithm was that it be compatible with a wide class of grammars. However, speed and space considerations required that the context considered in decisions be limited. This, in

FIGURE 2

GRAMMAR ANALYSIS -- STANFORD UNIVERSITY -- ANALYZER VERSION OF JULY 30, 1968.

TODAY IS JULY 30, 1968.

PRODUCTIONS

```

1  <PROGRAM> ::= _|_ <BOOLEAN EXPRESSION> _|_
2  <BOOLEAN EXPRESSION> ::= BOOLEAN_IDENTIFIER := <BOOLEAN EXPRESSION>
3  | <ARITHMETIC EXPRESSION> = <ARITHMETIC EXPRESSION>
4  <ARITHMETIC EXPRESSION> ::= ARITHMETIC_IDENTIFIER := <ARITHMETIC EXPRESSION>
5  | <ARITHMETIC EXPRESSION> - <ARITHMETIC TERM>
6  | <ARITHMETIC TERM>
7  <ARITHMETIC TERM> ::= ARITHMETIC_IDENTIFIER
8  | ( <ARITHMETIC EXPRESSION> )
9  | # ( <BOOLEAN EXPRESSION> )

```

TERMINAL SYMBOLS NONTERMINALS

1	=	10	<PROGRAM>
2	-	11	<ARITHMETIC TERM>
3	(12	<BOOLEAN EXPRESSION>
4)	13	<ARITHMETIC EXPRESSION>
5	#		
6	::=		
7	_ _		
8	BOOLEAN_IDENTIFIER		
9	ARITHMETIC_IDENTIFIER		

<PROGRAM> IS THE GOAL SYMBOL.

*** ERROR, GRAMMAR IS AMBIGUOUS.

IT IS LEFT AND RIGHT RECURSIVE IN THE SYMBOL <ARITHMETIC EXPRESSION>

turn, limits the class of acceptable grammars, but the limitations cannot be expressed in terms of simple restrictions on the form of the grammar.

Since debugging a grammar can often be a chore, much of ANALYZER is devoted to the generation of rather complete diagnostic messages for the various error conditions. The computer output in Figure 2 was

produced by ANALYZER for a typical small grammar.

After printing the grammar, ANALYZER lists its terminal symbols, nonterminal symbols (phrase class names) and goal symbols. The grammar is checked for being left and right recursive in any symbol (one of the obvious, yet frequent causes of ambiguity) [this check was suggested by Don Knuth]. In this example there are

FIGURE 3

GRAMMAR ANALYSIS -- STANFORD UNIVERSITY -- ANALYZER VERSION OF JULY 30, 1968.

TODAY IS JULY 30, 1968.

PRODUCTIONS

```

1  <PROGRAM> ::= _|_ <BOOLEAN EXPRESSION> _|_
2  <BOOLEAN EXPRESSION> ::= BOOLEAN_IDENTIFIER := <BOOLEAN EXPRESSION>
3                          | <ARITHMETIC EXPRESSION> = <ARITHMETIC EXPRESSION>
4  <ARITHMETIC EXPRESSION> ::= ARITHMETIC_IDENTIFIER := <ARITHMETIC EXPRESSION>
5                          | <ARITHMETIC TERM>
6  <ARITHMETIC TERM> ::= <ARITHMETIC TERM> - <ARITHMETIC PRIMARY>
7                      | <ARITHMETIC PRIMARY>
8  <ARITHMETIC PRIMARY> ::= ARITHMETIC_IDENTIFIER
9                          | ( <ARITHMETIC EXPRESSION> )
10                         | # ( <BOOLEAN EXPRESSION> )

```

TERMINAL SYMBOLS

NONTERMINALS

1 =	10 <PROGRAM>
2 -	11 <ARITHMETIC TERM>
3 (12 <BOOLEAN EXPRESSION>
4)	13 <ARITHMETIC PRIMARY>
5 #	14 <ARITHMETIC EXPRESSION>
6 :=	
7 _ _	
8 BOOLEAN_IDENTIFIER	
9 ARITHMETIC_IDENTIFIER	

<PROGRAM> IS THE GOAL SYMBOL.

FIGURE 3

C1 MATRIX FOR STACKING DECISIONS:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1 =	Y	Y												
2 -	Y	Y												
3 (Y	Y	YY											
4)	NN	N	N											
5 #	Y													
6 :=	Y	Y	YY											
7	Y	Y	#YY											
8 <u>BOOLEAN IDENTIFIER</u>			Y											
9 <u>ARITHMETIC IDENTIFIER</u>	NN	N	YN											
10 <PROGRAM>														
11 <ARITHMETIC TERM>	NY	N	N											
12 <BOOLEAN EXPRESSION>		#	#											
13 <ARITHMETIC PRIMARY>	NN	N	N											
14 <ARITHMETIC EXPRESSION>	#	#	N											

TABLE ENTRIES SUMMARY:

83
 22 Y
 16 N
 5 #

C1 TRIPLES FOR STACKING DECISION:

```

1 N FOR = <ARITHMETIC EXPRESSION> )
2 Y FOR ( <BOOLEAN EXPRESSION> )
3 Y FOR ( <ARITHMETIC EXPRESSION> =
4 Y FOR ( <ARITHMETIC EXPRESSION> )
5 N FOR := <BOOLEAN EXPRESSION> )
6 N FOR := <BOOLEAN EXPRESSION> |
*** ERROR, STACKING DECISION CANNOT BE MADE WITH (2,1) CONTEXT.
7 # FOR := <ARITHMETIC EXPRESSION> =
8 N FOR := <ARITHMETIC EXPRESSION> )
9 Y FOR | <BOOLEAN EXPRESSION> |
10 Y FOR | <ARITHMETIC EXPRESSION> =
11 N FOR <BOOLEAN EXPRESSION> | |
    
```

50 ENTRIES FOR 11 TRIPLES.

TABLE ENTRIES SUMMARY:

5 Y
 5 N
 1 #

FIGURE 3

PRODUCED HEAD SYMBOLS:

		11111
		12345678901234
1 =	Y	
2 -	Y	
3 (Y	
4)	Y	
5 #	Y	
6 :=	Y	
7 <u> </u> <u> </u>	Y	
8 BOOLEAN_IDENTIFIER	Y	
9 ARITHMETIC_IDENTIFIER	Y	
10 <PROGRAM>	Y Y	
11 <ARITHMETIC TERM>	Y Y Y Y Y	
12 <BOOLEAN EXPRESSION>	Y Y Y Y Y Y Y Y	
13 <ARITHMETIC PRIMARY>	Y Y Y Y	
14 <ARITHMETIC EXPRESSION>	Y Y Y Y Y Y	

CONTEXT CHECK FOR EQUAL AND EMBEDDED RIGHT PARTS:

THERE ARE 12 AND 12 VALID CONTEXTS, RESPECTIVELY, FOR
 6 <ARITHMETIC TERM> ::= <ARITHMETIC TERM> - <ARITHMETIC PRIMARY>
 7 <ARITHMETIC TERM> ::= <ARITHMETIC PRIMARY>
 THEY CAN BE RESOLVED BY LENGTH.

ANALYSIS OF (2,1) CONFLICTS:

THE TRIPLE := <ARITHMETIC EXPRESSION> = MUST HAVE THE VALUE N FOR
 4 <ARITHMETIC EXPRESSION> ::= ARITHMETIC_IDENTIFIER := <ARITHMETIC EXPRESSION>
 IN THE CONTEXT (... =
 IN THE CONTEXT := ... =
 IN THE CONTEXT | ... =

THE TRIPLE := <ARITHMETIC EXPRESSION> = MUST HAVE THE VALUE Y FOR
 3 <BOOLEAN EXPRESSION> ::= <ARITHMETIC EXPRESSION> = <ARITHMETIC EXPRESSION>
 IN THE CONTEXT := ...)
 IN THE CONTEXT := ... |

ANALYSIS COMPLETE FOR ITERATION 1
 * ONE ERROR WAS DETECTED

FIGURE 4

GRAMMAR MODIFICATION TO ATTEMPT TO RESOLVE CONFLICTS:

```

11 <:=1> ::= :=
   2 <BOOLEAN EXPRESSION> ::= BOOLEAN_IDENTIFIER <:=1> <BOOLEAN EXPRESSION>
12 <:=2> ::= :=
   4 <ARITHMETIC EXPRESSION> ::= ARITHMETIC_IDENTIFIER <:=2> <ARITHMETIC EXPRESSION>

```

PRODUCED HEAD SYMBOLS: PAGE 1 OF 1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1 =																
2 -																
3 (
4)																
5 #																
6 :=																
7 <u> </u>																
8 <u> </u> BOOLEAN_IDENTIFIER																
9 <u> </u> ARITHMETIC_IDENTIFIER																
10 <PROGRAM>																
11 <ARITHMETIC TERM>																
12 <BOOLEAN EXPRESSION>																
13 <ARITHMETIC PRIMARY>																
14 <ARITHMETIC EXPRESSION>																
15 <:=1>																
16 <:=2>																

CONTEXT CHECK FOR EQUAL AND EMBEDDED RIGHT PARTS:

THERE ARE 3 AND 4 VALID CONTEXTS, RESPECTIVELY, FOR

```

12 <:=2> ::= :=
11 <:=1> ::= :=

```

THEY CAN BE RESOLVED BY (1,0) CONTEXT.

THERE ARE 14 AND 14 VALID CONTEXTS, RESPECTIVELY, FOR

```

6 <ARITHMETIC TERM> ::= <ARITHMETIC TERM> - <ARITHMETIC PRIMARY>
7 <ARITHMETIC TERM> ::= <ARITHMETIC PRIMARY>

```

THEY CAN BE RESOLVED BY LENGTH.

FIGURE 4

C1 MATRIX FOR STACKING DECISIONS:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1 =	Y	Y	Y													
2 -	Y	Y	Y													
3 (Y	Y	YY													
4)	NN	N	N													
5 #	Y															
6 :=	N	N	NN													
7 _ _	Y	Y	#YY													
8 BOOLEAN_IDENTIFIER			Y													
9 ARITHMETIC_IDENTIFIER	NN	N	YN													
10 <PROGRAM>																
11 <ARITHMETIC TERM>	NY	N	N													
12 <BOOLEAN EXPRESSION>		#	#													
13 <ARITHMETIC PRIMARY>	NN	N	N													
14 <ARITHMETIC EXPRESSION>	#	#	N													
15 <:=1>	Y	Y	YY													
16 <:=2>	Y	Y	Y													

TABLE ENTRIES SUMMARY:

94
 25 Y
 20 N
 5 #

C1 TRIPLES FOR STACKING DECISION:

1 N FOR = <ARITHMETIC EXPRESSION>)
 2 Y FOR (<BOOLEAN EXPRESSION>)
 3 Y FOR (<ARITHMETIC EXPRESSION> =
 4 Y FOR (<ARITHMETIC EXPRESSION>)
 5 Y FOR _|_ <BOOLEAN EXPRESSION> _|_
 6 Y FOR _|_ <ARITHMETIC EXPRESSION> =
 7 N FOR <BOOLEAN EXPRESSION> _|_ _|_
 8 N FOR <:=1> <BOOLEAN EXPRESSION>)
 9 N FOR <:=1> <BOOLEAN EXPRESSION> _|_
 10 Y FOR <:=1> <ARITHMETIC EXPRESSION> =
 11 N FOR <:=2> <ARITHMETIC EXPRESSION> =
 12 N FOR <:=2> <ARITHMETIC EXPRESSION>)

55 ENTRIES FOR 12 TRIPLES.

TABLE ENTRIES SUMMARY:

6 Y
 6 N
 0 #

two distinct phrase structures for the form:

```
ARITHMETIC IDENTIFIER :=
<ARITHMETIC EXPRESSION>
  - <ARITHMETIC TERM>
```

We can remove this ambiguity by inserting another phrase class, and separating the recursions involving := and -. The output of a successful run on the revised grammar is given in Figure 3.

After listing the grammar, the terminal, non-

FIGURE 5

```
DECLARE NSY LITERALLY '16', NT LITERALLY '9';
DECLARE V(NSY) CHARACTER INITIAL ( '<ERROR: TOKEN = 0>', '=',
  '-', '(', ')', '#', ':=', ' _', 'BOOLEAN IDENTIFIER',
  'ARITHMETIC IDENTIFIER', '<PROGRAM>', '<ARITHMETIC TERM>',
  '<BOOLEAN EXPRESSION>', '<ARITHMETIC PRIMARY>',
  '<ARITHMETIC EXPRESSION>', ':=', ':=' );
DECLARE V INDEX(21) FIXED INITIAL ( 1, 6, 7, 8, 8, 8, 8, 8, 8,
  8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 10);
DECLARE C1(NSY) BIT(34) INITIAL(
  "(2) 00000 00000 00000 00",
  "(2) 00010 10001 00000 00",
  "(2) 00010 10001 00000 00",
  "(2) 00010 10011 00000 00",
  "(2) 02202 00200 00000 00",
  "(2) 00010 00000 00000 00",
  "(2) 00020 20022 00000 00",
  "(2) 00010 10311 00000 00",
  "(2) 00000 01000 00000 00",
  "(2) 02202 01200 00000 00",
  "(2) 00000 00000 00000 00",
  "(2) 02102 00200 00000 00",
  "(2) 00003 00300 00000 00",
  "(2) 02202 00200 00000 00",
  "(2) 03003 00200 00000 00",
  "(2) 00010 10011 00000 00",
  "(2) 00010 10001 00000 00");
DECLARE NC1TRIPLES LITERALLY '5';
DECLARE C1TRIPLES(NC1TRIPLES) FIXED INITIAL ( 199684, 200193,
  200196, 461831, 462337, 986625);
DECLARE PRTB(12) FIXED INITIAL (0, 328460, 782, 0, 0, 1804,
  0, 0, 2063, 2818, 0, 3585, 2320);
DECLARE PRDTB(12) BIT(8) INITIAL (0, 10, 9, 0, 0, 1, 8, 5, 2,
  6, 7, 3, 4);
DECLARE HDTB(12) BIT(8) INITIAL (0, 13, 13, 16, 15, 10, 13,
  14, 12, 11, 11, 12, 14);
DECLARE PRLNGTH(12) BIT(8) INITIAL (0, 4, 3, 1, 1, 3, 1, 1,
  3, 3, 1, 3, 3);
DECLARE CONTEXT CASE(12) BIT(8) INITIAL (0, 0, 0, 2, 0, 0, 0,
  0, 0, 0, 0, 0, 0);
DECLARE LEFT CONTEXT(0) BIT(8) INITIAL ( 9);
DECLARE LEFT INDEX(7) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 1);
DECLARE CONTEXT TRIPLE(0) FIXED INITIAL ( 0);
DECLARE TRIPLE INDEX(7) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 1);
DECLARE PR INDEX(16) BIT(8) INITIAL ( 1, 1, 1, 1, 3, 3, 5, 6, 6,
  7, 7, 8, 9, 11, 13, 13, 13);
```

terminal, and goal symbols, ANALYZER has printed the matrix of produced 1—heads (X is a produced 1-head of A if a string starting with X can be produced from A). After computing all valid contexts of all productions, it has checked equal and embedded right parts to determine the context necessary to distinguish them. (If two productions could not be distinguished using one symbol of context to the left and the right, it would have listed all of the problem contexts.) It then computed and printed the stacking decision matrix and the triples for the resolution of pair conflicts. Where the stacking decision could not be made based on the top two symbols of the stack plus the incoming symbol, it flagged the triple and listed the problem production and contexts.

One of the principal problems for bounded context parsing algorithms is the “insulating comma”—that is, a symbol which occurs in many contexts as punctuation, and does not itself serve as adequate context for various decisions [Lynch 68]. In our example, as a result of the use of := in both arithmetic expressions and boolean expressions, one stacking decision cannot be made with the triples table. Frequently this problem is solved by requiring the scanner to make the decision and return <comma1>, <comma2> ... (or <:=1>, <:=2>) as determined by some more global and ad hoc context. An alternative solution for an algorithm which can handle equal right parts is to leave the comma as a terminal symbol and add to the grammar production of the form

```
<comma1> ::= ,
<comma2> ::= ,
```

...

as required. This places the burden of decision on the

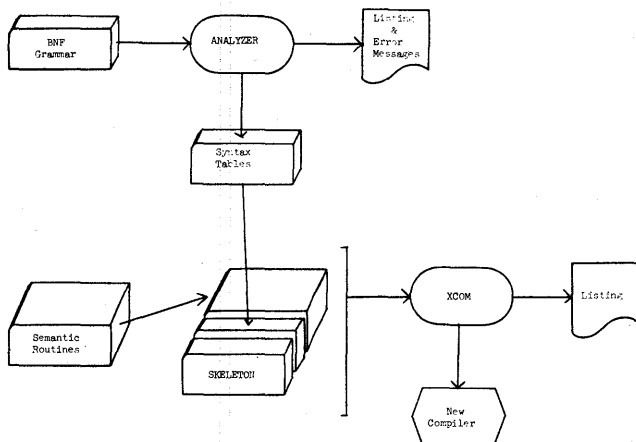


FIGURE 6

general parsing algorithm. The process can be partially automated and ANALYZER contains a procedure which will optionally modify the grammar in this fashion to remove conflicts. In the example above, two new nonterminal symbols, and two new productions have been added to the grammar. Subsequent output in Figure 4 shows that this process has removed the local ambiguity from the grammar.

Finally, if desired, the tables in the form required for the SKELETON program may be listed or punched. The tables for the example are shown in Figure 5.

Building a translator.

Once the BNF grammar for the source language has been debugged, the construction of the translator can begin. When the tables punched by ANALYZER are inserted into SKELETON, (and its scanner modified to reflect the identifier, comment, etc. conventions of the new language), SKELETON becomes the XPL description of a table-driven syntax checker for the source language. The new program can then be compiled by XCOM and run under XPLSM. To turn this proto-translator into a translator, emitters for the object language must be inserted into the case statement for the various productions (refer to Figures 1 and 6). An example of some simple semantic routines to compile the language of section 4 for a single-address, single accumulator machine is given in Figure 7. The compilation method does not restrict the form of the object code. XCOM produces absolute machine language as output; another compiler we have written (for the student language) produces an intermediate code similar to Polish postfix, which is then interpreted. Other choices are possible.

What we have described is a fully operational system which has been in use at Stanford since 1967. Although we are continuing to experiment and improve, we plan to release a stabilized version soon through the SHARE organization. Preliminary documentation, which will ultimately appear in the form of a book, is available from the authors.

REFERENCES

- 1 J. A. FELDMAN and D. GRIES
Translator writing systems
Comm ACM 11 Feb 1968
- 2 R. W. FLOYD
Operator precedence and syntactic analysis
J ACM 10 July 1963 pp 316-333
- 3 ———, *Bounded context syntactic analysis*
Comm ACM 7 Feb 1964 pp 62-65
- 4 W. C. LYNCH
A high-speed parsing algorithm for ICOR grammars
Jennings Computer Center Report 1097 Case Western Reserve

FIGURE 7

```

DO WHILE COMPILING;
  /* ONCE AROUND FOR EACH PRODUCTION (REDUCTION) */
  :
  /* LOCATE LEFTMOST REDUCIBLE SUBSTRING
  LEFT_END AND RIGHT_END DELIMIT IT */
  :
DO CASE PRDTB(PRD);
  /* ONE STATEMENT FOR EACH PRODUCTION OF THE GRAMMAR */
  :
  /* <ARITHMETIC TERM> ::= <ARITHMETIC TERM> - <ARITHMETIC PRIMARY> */
DO;
  IF TYPE(LEFT_END) = VARIABLE THEN
    CALL EMIT(LOAD, LOC(LEFT_END));
  CALL EMIT(SUB, LOC(RIGHT_END));
  TYPE(LEFT_END) = EXPRESSION;
END;

  /* <ARITHMETIC TERM> ::= <ARITHMETIC PRIMARY> */
  ;

  /* <ARITHMETIC PRIMARY> ::= ARITHMETIC_IDENTIFIER */
DO;
  LOC(LEFT_END) = ID_LOOKUP(NAME(LEFT_END));
  /* FIND THE LOCATION OF THE IDENTIFIER FROM THE SYMBOL TABLE */
  TYPE(LEFT_END) = VARIABLE;
END;

  /* <ARITHMETIC PRIMARY> ::= ( <ARITHMETIC EXPRESSION> ) */
DO;
  /* SAVE THE INFORMATION IN THE PARALLEL STACKS */
  TYPE(LEFT_END) = TYPE(LEFT_END + 1);
  LOC(LEFT_END) = LOC(LEFT_END + 1);
END;

  /* <ARITHMETIC PRIMARY> ::= # ( <BOOLEAN EXPRESSION> ) */
DO;
  /* THE <BOOLEAN EXPRESSION> MUST ALREADY BE IN THE ACCUMULATOR */
  CALL EMIT(Sharp_op, 0); /* UNARY OPERATOR */
  TYPE(LEFT_END) = EXPRESSION;
END;

  /* <:=1> ::= := */
  ;

  /* <:=2> ::= := */
  ;

END; /* OF CASE ON PRODUCTION NUMBER */

RIGHT_END = LEFT_END;
HEADS(RIGHT_END) = HDTB(PRD); /* THE REDUCTION */

END; /* OF DO WHILE COMPILING */

```

University 1968

5 W M McKEEMAN

An approach to computer language design

Technical Report CS 48 Computer Science Department

Stanford University August 1966

6 W M McKEEMAN J J HORNING D B WORTMAN

A compiler generator implemented on the IBM system/360

To be published

7 N WIRTH H WEBER

EULER a generalization of algol and its formal definition parts I and II

Comm ACM 9 Jan Feb 1966 pp 13-25

pp 89-90

8 N WIRTH C A R HOARE

A contribution to the development of algol

Comm ACM 9 June 1966 pp 413-431

APPENDIX FORMAL DEFINITION OF XPL

XPL is defined by the BNF grammar given below:

```

<program> ::= <statement list>

<statement list> ::= <statement>
                  | <statement list> <statement>

<statement> ::= <basic statement>
               | <if statement>

<basic statement> ::= <assignment> ;
                    | <group> ;
                    | <procedure definition> ;
                    | <return statement> ;
                    | <call statement> ;
                    | <go to statement> ;
                    | <declaration statement> ;
                    | ;
                    | <label definition> <basic statement>

<if statement> ::= <if clause> <statement>
                | <if clause> <>true part> <statement>
                | <label definition> <if statement>

<if clause> ::= IF <expression> THEN

<>true part> ::= <basic statement> ELSE

<group> ::= <group head> <ending>

<group head> ::= DO ;
              | DO <step definition> ;
              | DO <while clause> ;
              | DO <case selector> ;
              | <group head> <statement>

<step definition> ::= <variable> <replace> <expression> <iteration control>

<iteration control> ::= TO <expression>
                    | TO <expression> BY <expression>

<while clause> ::= WHILE <expression>

<case selector> ::= CASE <expression>

```

```

<procedure definition> ::= <procedure head> <statement list> <ending>

<procedure head> ::= <procedure name> ;
                  | <procedure name> <parameter list> ;

<procedure name> ::= <label definition> PROCEDURE

<parameter list> ::= <parameter head> <identifier> . )

<parameter head> ::= (
                  | <parameter head> <identifier> ,

<ending> ::= END
          | END <identifier>
          | <label definition> <ending>

<label definition> ::= <identifier> :

<return statement> ::= RETURN
                   | RETURN <expression>

<call statement> ::= CALL <variable>

<go to statement> ::= <go to> <variable>

<go to> ::= GO TO
          | GOTO

<declaration statement> ::= DECLARE <declaration element>
                        | <declaration statement> , <declaration element>

<declaration element> ::= <type declaration>
                        | <identifier> LITERALLY <string>

<type declaration> ::= <identifier type>
                    | <bound head> <number> ) <type>
                    | <type declaration> <initial list>

<type> ::= FIXED
        | CHARACTER
        | LABEL
        | <bit head> <number> )

<bit head> ::= BIT (

<bound head> ::= <identifier> (

<initial list> ::= <initial head> <constant> )

<initial head> ::= INITIAL (
                | <initial head> <constant> ,

```

```

<assignment>      ::= <variable> <replace> <expression>
                   | <left part> <assignment>

<replace>         ::= =

<left part>       ::= <variable> ,

<expression>     ::= <logical factor>
                   | <expression> | <logical factor>

<logical factor> ::= <logical secondary>
                   | <logical factor> & <logical secondary>

<logical secondary> ::= <logical primary>
                       | ! <logical primary>

<logical primary> ::= <string expression>
                       | <string expression> <relation> <string expression>

<relation>       ::= =
                   | <
                   | >
                   | <=
                   | <
                   | >
                   | < =
                   | > =

<string expression> ::= <arithmetic expression>
                       | <string expression> || <arithmetic expression>

<arithmetic expression> ::= <term>
                           | <arithmetic expression> + <term>
                           | <arithmetic expression> - <term>
                           | + <term>
                           | - <term>

<term>           ::= <primary>
                   | <term> * <primary>
                   | <term> / <primary>
                   | <term> MOD <primary>

<primary>        ::= <constant>
                   | <variable>
                   | ( <expression> )

<variable>       ::= <identifier>
                   | <subscript head> <expression> )

<subscript head> ::= <identifier> (
                   | <subscript head> <expression> ,

<constant>      ::= <string>
                   | <number>

```


A syntax directed processor writing system

by EORS N. FERENTZY and
JAMES R. GABURA

University of Toronto
Toronto, Ontario, Canada

INTRODUCTION

This paper introduces MPL/1, a processor writing system. It has two components: the MPL/1 language in which the processor specifications are written, and the translator which effects the translation of MPL/1 processor specifications into executable code. We emphasize processor writing system since the target programs produced by the system may do a variety of things, of which compilation is only a special case. There are numerous problems in linguistic analysis, artificial intelligence, data base conversion, and so on, where syntactic analysis and complex programs executed as semantic actions are deeply intertwined. The tasks to be performed usually require the availability of character and bit manipulation, file handling and list processing facilities. Two examples from the authors' field of interest follow.

In computer analysis of music, while analysing a composition, recognizing phrases and other syntactical structures, one would like to gather complex statistics on the occurrence of the syntactic formations observed. Since many musically significant constituents and delimiters exist which can only be discovered through the use of complex programs, such as chord root, key, modulation, and so on, a need arose for a flexible system which would allow free interplay between such programs and a controlling syntax specification.

In the analysis of postal addresses, first a facility allowing the use of variable syntax description is definitely needed, since it seems that a useful syntactic address description can evolve from experimentation only. On the other hand, as we proceed in the syntactic analysis of an actual address, we have to switch constantly to large file search routines to verify address components. These routines correspond functionally to the semantic action rules found in compilers, the difference being in the complexity of the action. In address analysis, for example, the semantic action "street name verification" may actually consist of a major

program for searching through a file hierarchy and using the result of previous analysis steps (the identified city name and province name) to cut down the search space.

Our choice for the semantic language of MPL/1 narrowed to PL/1 as this was an available ALGOL-like high-level language with many advanced facilities, including character and bit data types. Character and bit manipulation capabilities of PL/1 allowed us to use this language at different levels of the MPL/1 system. The translator for our meta-language is coded in PL/1. This translator is capable of accepting an MPL/1 specification of a processor containing PL/1 test in the semantic component and construct from it the corresponding processor referred to also as the Automation. The generated Automation itself consists of PL/1 text composed of a general driving mechanism, with the semantic and syntactic routines inserted, plus data for syntax and semantics linkage. Thus the generated Automation is executable on any computer equipped with a PL/1 compiler. Future enrichments of PL/1 become automatically available in MPL/1. The development of our system has been greatly influenced by Domolki's method of syntax analysis¹ the PL/1 Language Specifications², and the Revised ALGOL 60 Report³. Various extensions were made to the techniques and formalisms used by the languages in these sources in order to allow for the analysis of languages whose grammars are not context-free, and also to obtain more efficient syntax representations.

2. Example and notation

In this section we illustrate MPL/1 with an example, and develop a notation as we go along. The notation will be summarised in Section 2.2.

2.1 A sample MPL/1 program

Figure 1 shows the MPL/1 specification of a sim-

```

/* CONTROL */
DECLARE KWORD(4) CHARACTER(15), TYPE(4) FIXED BINARY;
      KWORD(1) = 'TOM';      TYPE(1) = 1;
      KWORD(2) = 'Y';       TYPE(2) = 1;
      KWORD(3) = 'SMITH';   TYPE(3) = 2;
      KWORD(4) = 'LI';     TYPE(4) = 2;

READ: GET LIST(CH);

/* INITIAL PRODUCTIONS */
<L> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z;
'' ::= '';

/* PRODUCTIONS & SEMANTIC ACTIONS */
1 <WD> ::= <L>'';          R(1) = V(1);
2 <PW> ::= <L><L>;         R(1) = V(1)||V(2);
3 <PW>> ::= <PW><L>;      R(1) = V(1)||V(2);
4 <WD> ::= <PW>'';       R(1) = V(1);
5 KW(<PN>|<LN>) ::= <WD>; R(1) = V(1);
6 <2> ::= ''<$>;         R(1) = V(2);
7 <S> ::= <PN><LN>;      PUT EDIT (V(1),V(2))(A(15),A(15));

/* PROCEDURES */
KW: PROCEDURE;
      DO I = 1 TO N; IF KWORD(I) = V(1) THEN RETURN(TYPE(I));
      END;
      GO TO READ;

END KW;

```

FIGURE 1—Personnel file processor

plified personnel file processor that reads from an external file a sequence of 'firstname + lastname' items in free format, analyses the items for syntactical correctness, checks the words of the items against standard personnel lists and prints a list of fixed format 'firstname + lastname' pairs. In Figure 1, abbreviations are used which have the following meanings:

L—letter
 “—blank character
 WD—word
 PW—partial word
 FN—first name
 LN—last name

The variables V and R are defined in the MPL/1 system. The small numbers in the left margin are used in the text which follows to identify the productions, and are not part of the example. The text following Figure 1 describes the function of each item used in the sample MPL/1 program.

Let us hypothesize an MPL/1 machine that directly interprets MPL/1, and let us execute the Personnel File Processor on this machine with actual data. Let this data consist of a list of 'firstname, lastname' pairs in the following form:

'Y""LI"', ""'TOM"SMITH""',...('' stands for blank).
 First the /* CONTROL */ section of the processor,

consisting of PL/1 code, is executed. This initializes the user-defined tables KWORD (a merged list of the first and last names of the personnel) and TYPE (in which 1 stands for firstname, 2 for lastname). In general, at this point the processor would initialize and read all the tables needed during actual processing. The 'GET LIST' statement reads in the first 'firstname, lastname' pair into the variable CH.

The /* INITIAL PRODUCTIONS */ are interpreted next. An Initial Production consists of the string ':: = ', to the right of which there can be a single character, or characters separated by the or-symbol represented by '|', or a sequence of two quote marks which is understood by the system to represent the blank character. We find on the left hand side of ':: = ' a letter sequence which will be referred to as 'terminal name' or 'terminal.' This letter sequence may be optionally surrounded by metalinguistic brackets (<>) to denote the fact that it is not actually a character string occurring in the primary input; however, it is still considered a terminal symbol with respect to the productions proper that follow in the /* PRODUCTIONS & SEMANTIC ROUTINES */ section of the processor: We will denote the set of all the left hand sides of the initial productions by V_t and call it the 'terminal vocabulary.'

The initial productions transform the primary input, CH = 'Y""LI"', into the following array:

This array, to which we will refer as the input stack (IS, $IS_i = (IN_i, IV_i)$ $i = 1, \dots, r$), is a pushdown stack. The IV-s contain the characters of the input 'Y""LI"', and the IN-s contain the corresponding terminal names, assigned to them by the Initial Productions.

Next the mechanism appearing on lines numbered 1-7 in Figure 1 is invoked. We will refer to this mechanism as the rule set R. A rule set consists of rules, $R = (R_1, \dots, R_r)$. In the present case Rule_j occupies line j. We refer to the text up to the 1st ';' on line j as the production component PR_j of R_j, and to the PL/1 statements up to PR_{j+1} as the semantic action component SEA_j of R_j. We refer to the text to the left and to the right of ':: = ' in PR_j as the left hand side (LHS_j), and the right hand side (RHS_j) of PR_j, respectively. We recognise in the PR_js occurrences of the terminals. We also find constructions of the form $A = \langle \text{letter sequence} \rangle$, $A \in V_t$. Let us call these strings non-terminals, and denote the set of all terminals and non-terminals by V. We define a right hand constituent RHC or left hand constituent LHC as either an element of V or a construction of the type <\$> or <integer>. Then it is true that

$$RHS_j = (RHC_j^1, \dots, RHC_j^{m_j})$$

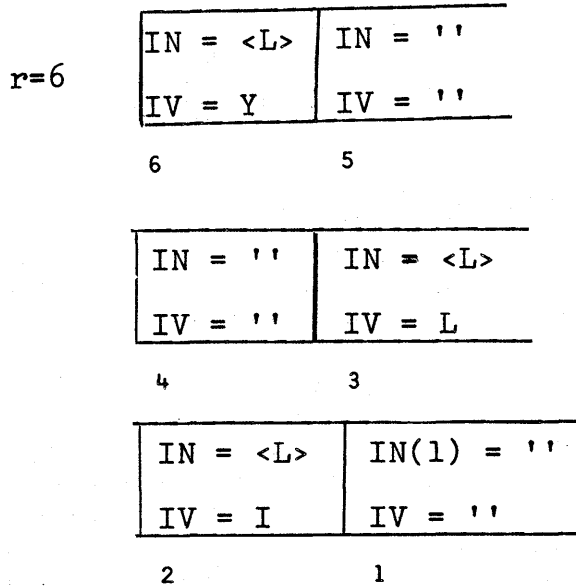


FIGURE 2—Initial input stack

$$LHS_j = (LHC_j, \dots, LHC_j^j)$$

or

$$LHS_j = (FC, (LHA_j^1, \dots, LHA_j^j))$$

where FC is a PL/1 identifier
and LHA_j^i is a list of constituents

The invoked rule set R operates on two pushdown stacks: on the input stack IS and on the analysis stack AS, defined now as $AS = (AS_j, \dots, AS_1)$
 $AS_k = (AN_k, AV_k)$.

We will refer to AN_k , which may store terminals or non-terminals, as the name component of AS_k . AV_k will be called the value component of AS_k : it stores character strings.

We show in Figure 3 the 14 states into which (IS,AS) is successively transformed as the result of invoking the rule set in Figure 1 and using the input stack of Figure 2. By the time the last state is reached, the original input "Y" "LI", verified and reformated, is printed out. Figure 3 shows for each state the state number, the index of the top element of IS (since we'll only take out elements from IS, the top index characterises the state of IS), the state of AS explicitly, and the Operation of Changing State_j into State_{j+1}. In the text below, 'we' refers to the MPL/1 machine.

State No.	State of IS (r).	State of AS			Operation
		AN ₃ AV ₃	AN ₂ AV ₂	AN ₁ AV ₁	
1	6				Move
2	5			<L> Y	Move
3	4		<L> Y	'' ''	R ₁
4	4			<WD> Y	R ₅
5	4			<FN> Y	Move
6	3		<FN> Y	'' ''	Move
7	2	<FN> Y	'' ''	<L> L	R ₆
8	2		<FN> Y	<L> L	Move
9	1	<FN> Y	<L> L	<L> L	R ₂
10	1		<FN> Y	<PW> LI	Move
11	0	<FN> Y	<PW> LI	'' ''	R ₄
12	0		<FN> Y	<WD> LI	R ₅
13	0		<FN> Y	<LN> LI	R ₇
14	0			<S>	READ

FIGURE 3—States of the input and analysis stacks

State 1 We start with AS empty, IS as produced by the Initial Productions. Execute 'Move', that is move the top element of IS onto the top of AS. The result is State 2.

State 2 The RHS_s of the productions are compared to $AN_1 = <L>$. Since no match is found, execute 'Move'.

State 3 The RHS_s of the productions are compared first to $AN_1 = ''$, then to $(AN_2, AN_1) = (<L>'')$. This time there is match with RHS₁, and the actions associated with R₁ are executed. (We will say that 'R₁ is applicable' or 'PR₁ is applicable'.) (AN_2, AN_1) is replaced by LHS₁. We call this operation a syntactic action, actually the syntactic action of rule 1, denoted by SYA₁. We also execute the semantic action component SEA₁ of R₁. This means that we execute all the PL/1 statements, whatever they should be, up to the statement that precedes PR₂. The values of AV₂, AV₁ are made available to these statements to operate on, by letting $V(1) = AV_2$, $V(2) = AV_1$ take place automatically. (AV_2, AV_1) is overwritten by whatever is left in R(1) after having executed the PL/1 statements. See the result in State 4.

State 4 We find that R_5 is applicable, therefore we execute SYA_5 and SEA_5 . Syntactic action 5 consists of the following steps: the PL/1 procedure, of which the name appears in LHS_5 , that is the procedure KW, is called. Since $V(1) = 'Y'$, KW returns the value 1. We therefore select the 1st element of the list that follows KW in PR_5 and replace AN_1 by that element. See the result in state 5. We refer to this mechanism as the 'computed rewrite feature' of MPL/1.

State 7 When comparing the RHS_6 , since $AN_2 = ''$, $AN_1 = <L>$, a match is found with RHS_6 , since $AN_2 = RHC_6^1$ and $AN_1 = RHC_6^2$. The latter equality holds, since $<\$>$ by definition equals anything which may pop up in an AN_u . Syntactic action 6 consists of the following steps: since LHS_6 is of the form $<integer>$, namely $<2>$, (AN_2, AN_1) will not be overwritten by $<2>$, but by the AN_u corresponding to RHC_6^2 , that is by $<L>$. See the result in State 8. This demonstrates the 'reference feature' of MPL/1.

State 9 Rule 2 is applicable. In SEA_2 , '||' is the concatenation operator of PL/1.

State 12 We find that, just as in State 4, Rule 5 becomes applicable. We see that this time $AN_1 = <WD>$ is rewritten into $AN_1 = <LN>$. Note that the choice was made depending on the value ('Y' or 'LI' resp.) associated with $<WD>$. In other words, through the 'computed rewrite feature' we allow the value components to monitor the rewriting process; the way in which this happens is specified by user defined procedures.

State 13 Rule 7 is applicable. In SEA_7 , "PUT EDIT (V(1), V(2)) (A(15), A(15));" is the PL/1 print statement that prints both V(1) and V(2) in fixed 15 character fields.

State 14 As $<S>$ appears in AN_1 , we transfer back to the READ label in the /* CONTROL */ section, and the processing of the next item on the input file begins. A 'transfer to READ' might also occur as an error exit of an SEA_j or an SYA_j .

Notes:

Although the illustration hopefully served its dual purpose of introducing the mechanisms of MPL/1 and of introducing the descriptive notation which is used later, it is misleading in many ways:

1. The reason for introducing an existing high-level language to specify semantic actions is to be able to easily write and execute complex programs as SEA_j s, rather than such trivial routines as are the SEA_j s of the illustration.

2. It would be rather inefficient to use the rule-application mechanism to recognise words, numbers, and eliminate blanks. It is possible in MPL/1 (as part

of the INITIAL PRODUCTIONS mechanism) to request direct word, number, and blanks recognition, so that R is applied to an IS, where $<WORD>$, $<NUMBER>$ are the terminals.

3. A LHS_i of a production may consist of a list of names, rather than just a single one.

4. If more than one R_j is applicable at a given time, only the first is applied.

2.2 Notation

2.2.1 Structure of the Automaton

The purpose of this section is to summarise the notation developed along with the analysis of an MPL/1 processor in the previous section. We do not claim to present a formal characterization of the broad class of Automata to which the processors specifiable in MPL/1 belong, and Automaton is used here as a synonym of 'MPL/1 program' or 'processor' only. However, it seemed worthwhile to present a very compact sketch of the static and dynamic structure of an MPL/1 program and comment on the variables of the notation. Better understanding of the variable's meanings can be obtained from the context in which they were introduced in Section 2.1 and from Sections 3 and 4.

Automaton = (V, V_i , S, R, IS, AS)

$R = (R_1, \dots, R_n)$

$R_i = (PR_i, SYA_i, SEA_i)$

$PR_i = (LHS_i, RHS_i)$

$LHS_i = (LHC_i^1, \dots, LHC_i^{a_i})$

or

$LHS_i = (FC, (LHA_i^1, \dots, LHA_i^{a_i}))$

$RHS_i = (RHC_i^1, \dots, RHC_i^{b_i})$

$IS = (IS_1, \dots, IS_q)$

$IS_q = (IN_q, IV_q)$

$AS = (AS_j, \dots, AS_1)$

$AS_q = (AN_q, AV_q)$

where

V = vocabulary of terminal and non-terminal names

V_i = vocabulary of terminal names

S = a distinguished non-terminal name

R = rule set

PR_i = production component of rule i

LHS_i = left hand side of production i

RHS_i = right hand side of production i

RHC_i^d = right hand constituent of the production i
 LHC_i^d = left hand constituent of the production i
 LHA_i^p = a list of constituents
 SYA_i = syntactic action component of rule i
 SEA_i = semantic action component of rule i
 IS, AS = input stack, analysis stack, respectively. Both are pushdown stacks.
 IS_q, AS_q = q^{th} element of IS, AS respectively
 IN_q, AN_q = name component of IS_q, AS_q respectively
 IV_q, AV_q = value component of IS_q, AS_q respectively
 FC = a PL/1 procedure name

Let us introduce the following auxiliary notation:

$$P = (PR_1, \dots, PR_n)$$

P stands for the production list, where PR_i is the production component of R_i .

$$AN = (AN_j, \dots, AN_1)$$

$$AV = (AV_j, \dots, AV_1)$$

AN, AV stand for the name, value component lists of the elements of AS .

$$AS_{1 \rightarrow k} = (AS_k, \dots, AS_1)$$

$AS_{1 \rightarrow k}$ denotes the list of the top k elements of the analysis stack. We will refer to this list informally also as 'top of AS .'

$AN_{1 \rightarrow k}, AV_{1 \rightarrow k}$ denote the list of the name components, value components of the elements of $AS_{1 \rightarrow k}$ respectively, and are referred to also as 'top of AN ,' 'top of AV .'

Note that the Automaton described differs essentially from a grammar $G = \{V, V_t, S, P\}$ by substituting R for P , that is by including the syntactic and semantic action rules in the Automaton. This corresponds to the fact that we allow an interplay between productions and action rules (see Sections 3,4) and thus the SYA_i and SEA_i 's form an integral part of the rewrite mechanism of the grammar. The input stack and analysis stack are included, since they also interact with the grammar specification.

The Automaton is viewed basically as a recognizer, although one could use it in a generative mode also, since we allow more than one constituent on the LHS.

2.2.2. Operation of the automaton

I Generate IS from the input AS is empty

II Loop1: Move top element of the pushdown

stack IS on the top of pushdown stack AS

Loop2: Search for k, i such that top of AN matches RHS_i .

Formally: $match(AN_{1 \rightarrow k}, RHS_i) = match(AN_k, RHC_i^r) \wedge \dots \wedge match(AN_1, RHC_i^r) = true$

a) if k, i found, execute SYA_i, SEA_i , go to Loop2

b) if k, i not found, go to Loop1

c) on *error*, go to error exit.

Notes

1. We will use the term 'PR_{*i*} is applicable' to state that we are in the state labelled a) above. An R_i is said to be applicable if its production is applicable. In order to be able to discuss certain features of the rule, we will normally assume that we are in this state.

2. The 'match' function is implicitly defined in Section 3.

The syntactic and semantic actions include those to stop the Automaton. Normally this will be the popping up of S in AN .

3. The *error* condition and backtracking are discussed in Section 4.

4. The Automaton is driven by a bottom up parsing algorithm, discussed in Section 4.

3. The MPL/1 language

Automatons introduced in Section 2 might be specified in MPL/1. Thus MPL/1 has facilities to specify the vocabulary of terminal names V_t , the vocabulary V and the final non-terminal S , to control the generation of the input stack IS , and to specify the rule set R . For complete example of an MPL/1 specification see the Appendix and Section 2.1.

3.1 Specification of V_t

Elements of V_t are introduced in a special set of BNF productions called Initial Productions. These productions link the characters of the primary input (which has to be a character string) to elements of V_t . The relationships defined through Initial Productions are used by the Automaton to generate IS from the primary input. The terminals introduced might be referred to in the production set P . Any character string (not containing '<' and '>') might be used to denote a terminal.

Example: $\langle DIGIT \rangle ::= 0|1|2|3|4|5|6|7|8|9;$

3.2 Specification of V and S

Non-terminals are implicitly defined by their first use in the productions. Denotation of non-terminals has to conform to '<character string>', where 'character string' may not contain '<' or '>'. S has to be used to represent S.

3.3 Specification of the rule set R

Since R consists of the rule list (R_1, R_2, \dots, R_v), we give first the form of $R_i = [PR_i, SYA_i, SEA_i]$. We assume that R_i is applicable, in order to be able to speak more easily about dynamic features of R_i . (See Section 2.2 for definition of 'applicable').

3.3.1 Form of the productions P

The form and action of the productions is an extension of the BNF mechanism.

Examples: $\langle APP \rangle ::= \langle APT \rangle \langle N \rangle$;

$\langle 2 \rangle ::= \langle A \rangle \langle \$ \rangle$;

Productions are 'free form' in the sense that card boundaries are ignored and arbitrary spacing between constituents of the production is allowed. A ';' terminates the production.

The following constructions are allowed as RHC:

- i. $RHC_q = \langle \$ \rangle$ will cause a match with AN_{mi-q+1} iff $AN_{mi-q+1} \in V$.
- ii. $RHC_q = \langle \$ - C_1 - \dots - C_u \rangle$ will cause a match with AN_{mi-q+1} iff $AN_{mi-q+1} \in \{V - C_1 \dots - C_u\}$.
- iii. $RHC_q = \langle C_1 + \dots + C_u \rangle$ will cause a match with AN_{mi-q+1} iff $AN_{mi-q+1} \in \{C_1, \dots, C_u\}$ and $C_i \in V$.
- iv. $RHC_q = \langle C \rangle, C \in V$ or $RHC_q = d, d \in V_i$.

If none of i, ii, iii holds, then RHC_q has to be an element of V and 'match' requires literal identity between RHC_q and AN_{mi-q+1} . i, ii, iii, and iv also define the function match used in the Appendix.

3.3.2 Syntactic actions SYA_i

Syntactic actions to be taken are deduced from the format of the LHS_i by the MPL/1 translator.

3.3.2.1 Standard syntactic action

If LHS_i is in proper BNF (a denotation of a list of elements of V) then the standard syntactic action is

taken: LHS_i updates AS, that is $AN_{1 \rightarrow k}$ is replaced by LHS_i.

3.3.2.2 Reference variable feature

If an $LHC_q = \langle s \rangle$, where s is an integer, then LHC_q is replaced by AN_{mi-s+1} . This action is repeated for all LHC_q of the form $LHC_q = integer$ and the resulting LHS_i updates AS. Note that the main role of the reference variables is to access names in AN matched by such constituents as $\langle \$ \rangle$ in the RHS.

Example: $\langle 2 \rangle ::= \langle A \rangle \langle \$ \rangle$;

3.3.2.3 Computed rewrite feature

If LHS_i is of the form: $FC(LHA_i^1 | LHA_i^2 | \dots | LHA_i^i)$ where FC is the name of a user defined PL/1 procedure, then $AN_{1 \rightarrow k}$ is replaced by LHS_i^r where r is an integer value returned by the procedure FC. Note that $AV_{1 \rightarrow k}$ is available at the time FC is invoked, so that FC may use these values as parameters.

An important special application of this facility occurs when RHS_i is a single constituent, and FC is a search procedure KEYWORD, which compares AV_1 with a value list VL, and returns r if a match is found with VL_r , which in turn causes AN_1 to be replaced by LHS_i^r. See an example of an application of this facility in the Appendix.

Example: $KW(\langle APT \rangle | \langle SD \rangle | \langle RR \rangle | \langle PN \rangle | \langle WG \rangle) ::= \langle W \rangle$;

3.3.2.4 Semantic actions SEA_i

Semantic actions are specified through any legal sequence of PL/1 statements. On applicability of R_i this sequence is executed. Some care must be taken to ensure compatibility among SEA_j 's belonging to different R_j 's, as it will become clear from Section 4. An SEA_i statement may call a procedure declared in another SEA_j . Input to the SEA_i comes from $AV_{1 \rightarrow k}$; its output updates $AV_{1 \rightarrow k}$. User defined input/output variables may also be used in the semantic routines.

3.3.3 Organization of PR_i, SEA_i's into a rule list R

As it is clear from the foregoing, we have explicit representations for the PR_j 's and the SEA_j 's (SYA_j 's are hidden in the format of the PR_j 's). The format of R has to conform to:

$\langle PR_1 \text{ denotation} \rangle \langle PL/1 \text{ text} \rangle \langle PR_2 \text{ denotation} \rangle \langle PL/1 \text{ text} \rangle \dots \langle PR_v \text{ denotation} \rangle \langle PL/1 \text{ text} \rangle$.
The PL/1 statements following the PR_j denotation are automatically interpreted by the MPL/1 translator as SEA_j .

Various control cards are needed to make the speci-

fication acceptable. A simplified example of a specification can be found in the Appendix and Section 2.1.

4. The automaton

An Automaton is the output of the MPL/1 translator, derived from the MPL/1 specifications of the Automaton. Section 5 gives some details concerning how the translation is actually performed. We are concerned here with the description of the Automaton only. The Automaton consists of a data part and a program part (predominantly PL/1 text). The data part might be further classified into data representing the syntax and data linking the syntax to the action rules. The program part is composed of the Automaton's core driving mechanism and of the syntactic and semantic action rules. As shown in Section 2, the operation of the Automaton consists of two major steps:

1. Find an applicable production PR_i .
2. Execute the syntactic and semantic actions associated with PR_i .

4.1 Retrieval of the first applicable production

Step 1 above is known as the parsing problem for recognition grammars. It consists of the identification of the index of the first applicable production in the production list. This step might be achieved in a number of ways. In our Automaton we have chosen to use a method developed by B. Domolki to achieve this goal.¹ The Domolki method is based on a compact representation of the RHS's of the productions, stored in a matrix with zero-one elements called the Syntax Matrix. We actually store a matrix element as a single bit in the computer (and refer to it as '1'BIT or '0'BIT) and operate on these entries as Boolean *true* or *false* values. Explicit search for a production with applicable RHS to the top of AS is replaced by bit string manipulation operating on a status bit vector Q , and row vectors of the matrix. As a result of the computation carried out on these bit strings, the index of the first applicable production is returned. Since PL/1 is a high-level language containing bit strings as one of the data types, we could program a direct representation of the bit manipulation required in the Domolki algorithm in PL/1. This proved very useful while experimenting with several forms of the algorithm. To obtain additional speed, we have replaced the bit manipulation program section with assembly language routines.

4.1.1 The Domolki parsing method

Let $s = \sum_{i=1}^v m_i =$ the sum of the lengths of all RHS's.

Let $M = \begin{bmatrix} M^1 \\ \vdots \\ M^t \end{bmatrix} =$ Syntax Matrix. M^j is a bit string of length s , and corresponds to the j^{th} syntactic name.

Let Q, U, V be bit strings of length s .

Let PT_1, \dots, PT_t be a vector of integers such that $PT_r = j$, where $r = \sum_{i=1}^j m_i$.

Let us define decompositions for M, Q, U , and V :

$$\begin{aligned} M &= M_1 M_2 \dots M_t \\ Q &= Q_1 Q_2 \dots Q_t \\ U &= U_1 U_2 \dots U_t \\ V &= V_1 V_2 \dots V_t \end{aligned}$$

such that the length of M_i^j, Q_i, U_i , and V_i , $j = 1, \dots, t$, is m_i .

We define (M, Q, U, V) implicitly, by giving the definitions for the components (M_i, Q_i, U_i, V_i) , $i = 1, \dots, v$.

Let us restrict temporarily the RHC's to elements of V . (The consequences of removing this restriction are discussed in 4.3.2). Then M_i is a representation of RHS_i , where $M_i^{j,k} =$ '1'BIT iff the j^{th} vocabulary element = the k^{th} constituent in the RHS_i . It follows that M_i has as many rows as there are syntactic names and as many columns as there are constituents on the RHS_i . Information concerning the match of RHS_i (partial or total) with the top of AN is stored in the bit string Q_i .

By definition, $Q_{i,k} = (RHC_k^1 = AN_k) \wedge \dots \wedge (RHC_k^k = AN_1)$, $k \leq m_i$, where $Q_{i,k}$ is the k^{th} component of Q_i .

It is clear from the definition that the last bit of Q_i is '1'BIT or '0'BIT depending on whether or not R is applicable. The power of the algorithm stems from its ability to efficiently update Q_i when executing the basic operation of the automaton, namely to put a new element on top of AS. Let AN' denote the stack derived from AN by a pushdown operation, putting IN_r on top, and let Q^1 denote the updated Q_i . Let IN_r be the name n . It follows that

$$AN_1' = n, AN_2' = AN_1, \dots, AN_{j+1}' = AN_j$$

Then

$$\begin{aligned} Q_{i',1} &= (RHC_i^1 = n) = M^{n,i_1} \\ Q_{i',k} &= [(RHC_i^1 = AN_k') \wedge \dots \wedge (RHC_i^{k-1} = AN_2')] \wedge (RHC_i^k = AN_1') \\ &= [(RHC_i^1 = AN_{k-1}) \wedge \dots \wedge (RHC_i^{k-1} = AN_1)] \wedge (RHC_i^k = n) \end{aligned}$$

$$= Q_{i,k-1} \Lambda M_i^n, \quad k = 2, \dots, m_i.$$

Note that we represent syntactic names in the algorithm with their respective row indices in M . Thus the M -row containing all the information on a syntactic name can be immediately retrieved.

Let U_i, V_i denote bit strings of length m_i .

$$U_i = '10 \dots 00'$$

$$V_i = '00 \dots 01'$$

Let M_i^n denote the n^{th} row of M_i

Let \vec{Q} perform a right shift on Q by one position

Let Λ -and, V -or operations be applied to the bits of their argument bit strings in parallel.

Then

$$Q_i' = (U_i \vec{V} Q_i) \Lambda M_i^n \quad (1)$$

$$Q_i' = '00 \dots 00' \text{ if } R_i \text{ can never be applicable to the top of the stack (from the definition of } Q_i) \quad (2)$$

and

$$Q_i' \Lambda V_i = '00 \dots 00' \text{ if } R_i \text{ is not applicable to AS} \quad (3a)$$

$$'00 \dots 01' \text{ if } R_i \text{ is applicable to AS} \quad (3b)$$

From previous definitions of (M, Q, U, V) it follows that (1) holds for \vec{Q} also:

$$Q' = (UV \vec{Q}) \Lambda M^n$$

$Q' = '00 \dots 00'$ implies that no production may become applicable to the top of AN . This should cause either an error exit or invoke a backtracking step. See Section 4.3.2 for a brief discussion of backtracking.

Now in $Q' \Lambda V$, there is a '1' BIT in each position j such that if R_j is an applicable rule in the rule set, $j = PT_j$. It follows that if there is at least one applicable production, then the position g of the first '1' BIT returns in $i = PT_g$, the index of the first applicable production. On the other hand if every bit in $Q' \Lambda V$ is a '0' BIT, no rules are applicable to AS'.

4.2 Execution of semantic and syntactic actions

Depending on the outcome of the search operation for an applicable production (the $Q' \Lambda V$ calculation), we either move the next element on top of the analysis stack, or we execute the syntactic and semantic actions corresponding to R_i , of which the index n is known at this point (it was returned in $i = PT_g$). We assume that the code corresponding to SYA_i and SEA_i is placed

in the same program block in which the parsing mechanism operates, and that they are preceded by a label with index i . In this manner a switch statement of the form *go to L* SYA_i , and *go to L* SEA_i is sufficient to activate SYA_i and SEA_i respectively.

4.2.1 Semantic actions

The code for semantic actions is whatever the user specified on the MPL/1 level. He has access to all the facilities of the PL/1 language.

4.2.2 Syntactic actions

As already stated in discussing the format of the LHS of the production, the standard syntactic action is the traditional rewrite rule of parsing programs. The execution of this rule is the fastest of all the syntactic actions which are user specified, such as the reference feature and the computed rewrite mechanism. It is important to note that if the user doesn't want to specify these non-standard features, they are not even generated into the Automaton by the translator. If the non-standard actions are present, the program switches to them only if the respective rules are applicable, otherwise it uses the fastest standard action.

Since the code representing syntactic and semantic action rules forms part of a single program, this fact implicitly imposes a set of restrictions on the use of variables and labels in the action rules. Since PL/1 has block structure, the individual semantic action rules may be made into blocks or procedures, and in this way clashes of identifiers can be reduced. However, this solution would not be advisable for an action that will be invoked frequently, due to the inefficiency involved in calling procedures and blocks.

All PL/1 identifiers except those specified by the user in his syntactic and semantic routines, namely those used in the PL/1 text standing for the core of the Domolki mechanism, are suffixed with the character string '@@' to identify them as system identifiers, thus virtually eliminating potential inadvertent clashes with user defined labels and variables. Thus the user need be only minimally concerned with the mechanics of the MPL/1 system.

4.3 Remarks on implementation

To enhance the clarity of exposition, we have described an overly simplified version of the currently implemented type of Automaton. The main differences are listed below

4.3.1 Productions of length 1

Productions with RHS _{i} of length 1 (except those

having a computed rewrite type of LHS_i) are not explicitly represented in the syntax matrix M, resulting in less required storage and faster execution. Accordingly, the true definition of M_{i,j,k} is as follows:

M_{i,j,k} = '1'BIT iff the jth syntactic name matches the kth constituent of the RHS_i or the jth syntactic name can be generated by a sequence of production of length 1 from the kth constituent of the RHS_i. We leave to the reader to verify that the relevant relationships, in particular the derivation of Q' holds under the new definition of M.

4.3.2 On the representation of extended BNF constructions allowed in the RHS in MPL/1

1. <\$> is represented by a column of '1'BIT in the syntax matrix. Since the column to represent that RHS constituent would have been reserved in any case, the <\$> representation does not require more storage than the representation of a specific RHS constituent. A glance at the parsing mechanism convinces us that we have not lost in processing time either, and that the formulae are still valid.
2. <\$-syntactic name list> is represented by a column having '0'BITS in the rows corresponding to elements of the list, and '1'BITS otherwise.
3. <syntactic name list> is represented by a column having '1'BITS in the rows of the list elements, and '0'BITS in the rest of the rows. The remark made at 1. holds for 2. and 3. also. Thus we have introduced new, general syntactic elements into the MPL/1 language at no extra cost to the user.

4.3.3 Backtracking

The driving mechanism of the Automaton is described as a non-backtracking algorithm. To make backtracking possible, it is sufficient to store j and r (pointing to the top of AS and IS respectively), Q, containing all information concerning the applicability of the productions, and the LHS_i of the applicable production. Using these values, the backtracking mechanism is similar to the ones used in other bottom up parsing algorithms and is not described here. Backtracking is invoked by the Q' = '00...00' condition. Both backtracking and non-backtracking versions of the algorithm were implemented.

We note that the MPL/1 language has facilities (as illustrated below) to eliminate certain false branches in the parsing tree, and thus enlarge the scope of applicability of the more efficient non-backtracking version of the algorithm. Thus if

<C> ::= <A>; <D> ::= <A> ;

occurs in the productions, we may rewrite it as

<C> <2> ::= <A> <\$- >;

This facility could be termed a user controlled look-ahead, and is more efficient than the general ones built into some parsing algorithms, since look ahead is performed for those productions only where the danger of a 'false rewrite' exists.

4.3.4 The input stack generator

A standard procedure of the automaton relieves the user of casting his input into the IS = (IN_r, IV_r, . . . , IN₁, IV₁) form. At present, the input denoted by CH can have the form of character string (e.g., a coded musical score or postal address), and the input stack is derived automatically by the Generator. Thus the functions of the Generator include splitting up CH into single characters and storing them into IVs, assigning syntactic names to the IVs and storing these in the INs. The Generator is monitored by a special group of BNF productions that is part of the MPL/1 specification of the Automaton. These Initial Productions link the character set of the input stream with the set of terminal names V_t. See Section 2.1 for examples.

The MPL/1 translator

The MPL/1 translator is a program of some 850 PL/1 statements, which accepts the MPL/1 language as input, and produces as output another PL/1 program and a set of syntax data associated with the program. If this new PL/1 program has been compiled, and when it has read in its syntax data, it becomes an Automaton, which is then ready to carry out its specified operations. The main functions of the MPL/1 translator are:

1. to extract from the MPL/1 input the productions defining the grammar
2. to generate from that information the Domolki syntax matrix of the form described previously
3. to merge PL/1 text standing for the core of the Domolki mechanism with the semantic action rules into a single PL/1 program
4. to supply information to the Input Stack Generator derived from the Initial Productions.

The semantic action rules are prefixed with labels and are given returns by the translator which allow them to be executed much like subroutines when the corresponding productions become applicable during the execution of the generated program. The translator must also generate tables containing the number of constituents

on the left and right hand sides of each of the productions in the grammar as part of the syntax linkage data. In addition, the translator must generate the vector PT, which defines the relationship between the columns of the syntax matrix and the productions in the grammar. Arrays are produced by the translator linking both the semantic and syntactic action rule associated with each production. An array is produced which specifies the row number in the Domolki matrix for each constituent of the LHS of each production. The U and V bit vectors are generated, needed for the updating of the status bit vector, and for the testing for applicable productions. For each of the conditional rewrite rules arrays are generated containing the Domolki row numbers for each of the constituents in each of the alternatives involved. Various other dimensioning variables are produced, needed for the dynamic allocation of the various arrays required at run-time. Finally, the MPL/1 translator has provision for inserting code into the various levels in the block structure of the final PL/1 program defining the Automaton, so that special dynamically allocated arrays may be used in the semantic routines if they happen to be required.

The MPL/1 System may be operated in several modes. Output from the translator may be directed through the punch unit, or optionally, control may be automatically passed over to the PL/1 compiler, and hence to the Automaton. Using the second option, there is uninterrupted machine action leading through the generation, compilation and execution of an Automaton. Therefore the user may test a particular processor specification (written in MPL/1) in a single pass.

An IBM System 360/50 was used for system implementation and testing. The system can be run on any computer equipped with a PL/1 compiler.

CONCLUSIONS

The type of system described could be easily duplicated in any high-level, ALGOL-like language, since we stayed in all phases of the system in the same high-level programming language. The distinctive features of the system are:

1. semantic rules are specified in a high-level language
2. as the bit manipulation steps form the core of the parsing operation, parsing is done efficiently.

We are aware of the theoretical problems concerned with the termination of parsing algorithms, interpretation of acceptance or non-acceptance of strings, and so on, raised by introducing in MPL/1 the generalised features in the syntax specification (context sensitivity, computed rewrite, and reference features). For the time

being, we merely point out that the new extensions to BNF were introduced in such a way that the user is not paying for this generality, assuming he does not want it. Thus the user may refrain from employing the added features, and may specify standard, context-free syntaxes only.

The major limitation of the current system lies in the fact that for extremely large languages, storage requirements for the syntax matrix may become excessive, and execution speeds may suffer correspondingly. We are therefore developing a hyper-language that allows several automata to be present and which has a variable type into which structured sets of automata can be stored. In this language, the names of automata as described in this paper will be the terminal symbols of the hyper-language. Parsing in this system will be done using a top down method; however, when a terminal name (automaton name) is encountered, the corresponding automaton is activated and performs the processing using the bottom up technique described in this paper. The idea in building such a system is that we expect to analyse structured information of which easily recognizable sub-structures can be characterized through independent grammars. For example, if

$$L = \{(a,b) | a \in L_1, b \in L_2\}$$

$$L_1 = L(G_1), L_2 = L(G_2),$$

then the method proposed would analyse 'a' using the productions of G_1 and analyse 'b' using the productions of G_2 , rather than analyse (a, b) using the union of the production sets of G_1 and G_2 . We hope in this fashion to break down large languages of interest in artificial intelligence into manageably sized sub-languages of which the grammars can be easily stored in the compact Domolki notation.

ACKNOWLEDGMENT

The authors would like to gratefully acknowledge the collaboration of Mr. John I. Weldon of the Dominion Bureau of Statistics concerning system applications, and of Mr. Edward Bierstone in system programming. This work was supported by the Department of Computer Science of the University of Toronto, the Canada Council, the National Research Council of Canada, and the Ontario Government under the Province of Ontario Fellowship Plan.

REFERENCES

- 1 *Domolki's algorithm*
In an unpublished set of notes on Syntax Analysis by P Wegner at Pennsylvania State University and brought to that author's

- attention by an unpublished paper by P H Frost
- 2 IBM system/360 operating system PL/1 language specifications
IBM Systems Reference Library Form C28-6571-4
 - 3 Revised report on the algorithmic language ALGOL 60
The computer Journal Jan 1963 p239
 - 4 R G TROUT
A Compiler-compiler system
Proc 22nd Nat l Conf ACM 1967 p 317
 - 5 S ROSEN
A compiler-building system developed by Brooker and Morris
Comm ACM 7 1964 p 403
 - 6 J C REYNOLDS
COGENT programming manual Report no 7022 Applied
Mathematics Division
Argonne National Laboratory March 1965
 - 7 J A FELDMAN
A formal semantics for computer languages and its application in
a compiler compiler
Comm ACM 9 1966 p 3

APPENDIX

Simplified postal address analyser

In the example above, the following abbreviations are used:

L—letter
D—digit
N—number
IN—intermediate number
W—word
IW—intermediate word
WG—general word
WL—word list
SD—street designator
APP—apartment part
APT—apartment
PN—province name
RR—rural route

```

/* INITIAL PRODUCTIONS */
<L> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z;
<D> ::= 0|1|2|3|4|5|6|7|8|9;
'' ::= '' ;
. ::= . ;
/ ::= / ;

/*PRODUCTION/SEMANTIC ROUTINE SECTION */
<S> ::= <N> <WL> <SD> <APP> <WL> <PN>;      PUT FILE (SYSPRINT) EDIT
                                              (V(6),V(5),V(3),V(2),V(1),V(4))
                                              (R(OUTF)); OUTF:FORMAT(A(15),
<S> ::= <N> <WL> <SD> <WL> <PN>;              A(60),A(10),A(45),A(7),A(12));
                                              PUT FILE (SYSPRINT) EDIT
                                              (V(5),V(4),V(3),V(2),V(1),'')
                                              (R(OUTF));
<S> ::= <N> <WL> <APP> <WL> <PN>;              PUT FILE (SYSPRINT) EDIT
                                              (V(5),V(4),'',V(2),V(1),V(3))
                                              (R(OUTF));
<S> ::= <RR> <N> <WL> <PN>;                    PUT FILE (SYSPRINT) EDIT
                                              (V(5),V(4),'',V(2),V(1),V(3))
                                              (R(OUTF));
                                              GO TO V1V2;
                                              GO TO V1;
<N> ::= <N> <N>;                               R(1) = V(1) || V(2) || V(3);
<N> ::= <IN> '' ;                               GO TO V1V2;
<N> ::= <IN> / <D> '' ;                          GO TO V1;
<IN> ::= <IN> <D>;                               GO TO V1V2;
<IN> ::= <D>;                                   GO TO V1;
<W> ::= <IW> . '' ;                             GO TO V1V2;
<W> ::= <IW> '' ;                               GO TO V1;
<IW> ::= <L>;                                   V1: R(1) = V(1);
<IW> ::= <IW> <L>;                               V1V2: R(1) = V(1)||V(2);
KW(<APT>|<SD>|<RR>|<PN>|<WG>) ::= <W>;         GO TO V1;
<WL> ::= <WG>;                                  BEGIN;
                                              DCL DW CHAR(15);
                                              DW = V(1);
                                              R(1) = DW;
<WL> ::= <WL> <WG>;                             END;
                                              BEGIN;
                                              DCL (DW1,DW2) CHAR(15);
                                              DW1 = V(1);
                                              DW2 = V(2);
                                              R(1) = DW1 || DW2;
<APP> ::= <APT> <N>;                             END;
                                              BEGIN;
                                              DCL DAPT CHAR(5), DNUM
                                              CHAR(7);
                                              DAPT = V(1);
                                              DNUM = V(2);
                                              R(1) = DAPT || DNUM;
<2> ::= '' <$>;                                 END;
                                              R(1) = V(2);

/* PROCEDURES */
In this section the user may define syntactic and semantic
procedures accessible from the /* PRODUCTION/SEMANTIC ROUTINE
SECTION */.

```


The minicomputer, a programming challenge

by ROBERT L. HOOPER

Compata, Incorporated
Tarzana, California

INTRODUCTION

Public attention, as well as the attention of groups such as this, has been focused on the design and application of large, expensive "super computers." Our national preoccupation with size and power makes this fact understandable. However, the minicomputer, which I define as a stored program computer selling for under twenty-five thousand dollars, is deserving of much more serious attention than it has heretofore been given.

While a 30% increase in units sold each year is possible over the next few years merely from increased use in the industrial areas, a much greater growth rate could come about if the minicomputer ever became an entry in the consumer goods market. At prices of two to four thousand dollars per main frame, a minicomputer with a dial-up service to specialized data bases could become attractive to the growing percentage of the population in the upper income bracket.

But such speculation is not the purpose of this paper. Rather, it is to examine the current status of programming support for the minicomputer and to suggest both software enhancements and hardware features to support them.

Ten years ago, an "inexpensive" computer typically sold for forty to fifty thousand dollars, was based on drum memory and discrete component technology, and could execute from a few hundred to a few thousand instructions per second. The numbers of all such computers delivered in any one year did not exceed a few hundred. Examples of such computers would include the Control Data LGP-30 (Librascope) and G-15 (Bendix).

Today, in 1968, the minicomputer is a proliferating breed, selling for anywhere from five to twenty-five thousand dollars, utilizing core mem-

ory and integrated circuits, and executing up to a million instructions per second. Over three thousand such systems will be delivered next year. It is my belief that during the next few years the percentage growth of this segment of the total computer market will increase three to four times faster than that of the overall computer market, which seems to be currently at 12% per year.

Speculations about the future in areas of technology usually suffer from extreme under- or overshoot. However, a continued sharp decrease in price (due to both manufacturing technology and to competition) and a modest increase in performance and reliability does not strain one's credulity. Changes in the software area are more uncertain. Price will no doubt continue to be the major determining factor in original equipment manufacturer orders, which are bread and butter in this price range.

Current minicomputer

Current minicomputers include the Digital Equipment Corp. PDP-8 series, the Hewlett-Packard HP2115A and 2114A, the Honeywell/Computer Control Division DDP-416, the Interdata models 2, 3 and 4, the Raytheon 703, the Scientific Control 650 and 655, the Systems Engineering Laboratories 810A and the Varian Data 620i and 520i. In addition, there are many other companies manufacturing or contemplating manufacture of minicomputers with a delivered price under \$15,000. There has been a recent "outburst" of offerings from lesser known companies offering computers in the \$5,000-\$10,000 range: SPC 12, DT 1600, Elbit 100 and PDC 808, to name only a few. Table I lists some current minicomputers available for under \$13,000. The more successful models have been delivered in quantities ranging

Manufacturer	Model	Price	Memory (bits)	Memory Cycle (usecs)
Digital Equipment Corp.	PDP-8/I	\$12,800*	4096×12	1.5
Digital Equipment Corp.	PDP-8/L	8,500*	4096×12	1.6
Elbit Computers	100	4,900	1024×12	2.0
General Automation	SPC-12	6,400	4096×8	2.2
Hewlett-Packard	2114A	9,950	4096×16	2.0
Hewlett-Packard	2115A	14,500	4096×16	2.0
Interdata	2	4,700	1024×8	2.0
Interdata	3	7,000	4096×8	2.0
Interdata	4	10,000	4096×8	2.0
Varian	DATA/520i	7,500	4096×8	1.5
Varian	DATA/620i	12,500	4096×16	1.8

*Price includes ASR-33

TABLE I—Current minicomputers (under \$13,000)

from hundreds to over a thousand. In general, these computers have word lengths of twelve or sixteen bits, and standard memories of 4,096 words, expandable to 32,768 words as an option. An ASR-33 teletype is the usual I/O device, but optional equipment includes high speed paper tape reader and punches, card readers, line printers, magnetic tapes and disks.

To obtain some historical perspective concerning performance and cost of small control computers over the past eight to ten years, let us remember that prices have decreased by a factor of four to eight, while memory speeds have decreased by about the same factor. A trend has also developed away from the extremes of twelve and twenty-four bits toward sixteen bits as a standard word length for this type of computer, although the PDP-8 series, at twelve bits, has been and still is, the most popular system.

It is interesting to note that in this same time period there has been little change in the concept of what manufacturer-supplied software should be like. But more of this later.

Applications characteristics

As the price per unit dropped and reliability and performance improved, a predictable result occurred. More and more applications became feasible for computer-based systems. The hard-wired "black box" was increasingly replaced by a stored program computer. Applications of minicomputers are frequently of a "real-time, on-line"

nature and include closed loop control of processes, data acquisition and recording, analytical instrumentation, automatic test, and communication systems. Users of the minicomputers are unlikely to be professional programmers but rather to be scientists, engineers and technicians without extensive programming experience. For this reason, the minicomputer should be supported by software suitable both to the application and to the user. Unfortunately, most present day minicomputer software does not appear to meet this requirement.

Standard software

The typical software configuration for a minicomputer can be described as follows: FORTRAN is available for systems with 8K words of memory. A basic one-for-one assembler is available for 4K systems and a macro assembler is available for 8K systems. Loaders which link, desectorize and relocate are usual. A tape editor is quite common and every computer seems to have math packages of single precision and possibly double precision fixed point routines. Hardware fault detection and on-line debug routines are also standard. Basic I/O drivers are normally included, as are mathematical routines for elementary functions.

Some of the software problems are caused by the minimum hardware configurations sold:

- small memory (4,096 words)
- slow I/O (ASR 33)

- lack of mass storage
- limited ability to address memory within an instruction.

Other problems, previously mentioned, result from an attempt to make available the standard tools of the professional programmer to the novice or occasional user. This seems inappropriate.

An economic point is worthy of some discussion. The desire to keep the unit price low is certainly an overriding requirement. If ever there were a case for separate pricing of software, this would seem to be it. This would be especially appreciated by the quantity buyer. My suggestion would be that a minimum set of software (to be defined) be included with each computer sold at a small additional cost and that one copy of associated documentation be furnished. Additional documentation sets and/or additional programs would be available for added payment. The customer must realize that printing costs and tape reproduction and verification costs are not negligible. They might run as high as half of the amortized non-recurring development costs. For one manufacturer, these costs ran from a few hundred to several hundred dollars for documentation production costs associated with the delivery of a single minicomputer.

A basic software set might be:

ASSEMBLER (BASIC)
 LOADER (RELOCATING AND DESECTORIZING)
 TAPE EDITOR
 DEBUG PACKAGE (BASIC)
 I/O CONTROL SYSTEM
 HARDWARE FAULT DETECTION ROUTINES

This basic software set might be developed for under \$50,000. Amortized over 250 computers, the non-recurring cost would add only \$200 per system, with perhaps another \$100 added for reproduction costs, resulting in a \$300 additional charge for the customer who needs only this minimum software package.

An augmented set available at additional cost might be:

ASSEMBLER (MACRO)
 COMPILER—SPECIALIZED FOR APPLICATION
 LOADER (LINKING, RELOCATING AND DESECTORIZING)

OPERATING SYSTEM
 MATH ROUTINES—SINGLE AND MULTIPLE PRECISION ARITHMETIC
 HARDWARE FAULT LOCATION
 SYMBOLIC DEBUG PACKAGE
 SUPER CALCULATOR PACKAGE
 ASSEMBLER
 COMPILER { PACKAGE TO RUN ON
 LARGE COMPUTER

This package might run over a quarter million dollars to develop. Amortized over only 50 computers, a price of under \$6,000 would cover both development and reproduction costs. Since the person who wants this package probably bought an expanded hardware system, this cost increment may be insignificant (approximately the cost of a 4K memory).

However, the point of the discussion at this point is not to insist on specific software items but rather to suggest that there are alternative methods of configuring software which may be more attractive both to manufacturers and to users.

Typical applications

It may be desirable at this point to step back and examine a few "typical" applications of minicomputers. Some of the characteristics of these applications may lead us to certain conclusions about desirable characteristics of the supporting software.

In the case of data acquisition and recording, the computer sits between a multiplexed A → D converter and a magnetic tape. The computer will typically send channel selection information to the multiplexer and supply a start convert signal. After the conversion complete signal goes on, the computer will read in the sampled data and store it in memory. At this point, the data may be limit checked, converted to engineering units, etc., but the processing per sample will likely be low. Fixed point, single precision operations are very adequate. After a memory buffer has been filled, this buffer will be written on magnetic tape while an alternative buffer is being filled with sampled data. The throughput rate will be from a few hundred to a few tens of thousands of samples per second. Channels must be sampled at a very precise rate, to avoid skewing of data. This requires either an interrupt timer or precisely written loops.

The amount of computation is relatively small

but the I/O rate is relatively high. (1,000–10,000 word/second composite rate). Standard I/O control routines probably are too “fat” and would not be used. The program itself would be written in assembly language. The user of such a system would need only an assembler, loader and debug package from the manufacturer.

Other applications involve communications, which is largely a case of character processing and control. Often the minicomputer controls peripheral devices on one side and looks across a phone line to a large computer on the other side. Some logical ability is required but almost no computational capability is needed. Again, an assembler, loader and debug package will suffice.

Instrumentation and process control applications will typically make use of larger memories and the computational capabilities furnished by a language like FORTRAN. Interrupt processing may become quite important. Some bit shuffling must be done. Monitor programs may be of considerable value. This type of user will generally wish much, much more than the preceding two users, although all may be using the same basic computer. Since the user purchasing minimum hardware is not forced to buy the additional memory required for the more complex application, why should he pay for software he doesn't need either?

Proposed standard software

A question which commonly arises is whether program development should be carried out in assembly language or compiler language. The compiler language commonly furnished for minicomputers is some version of FORTRAN, a computationally oriented language. Problems in implementation of these languages have led to complaints about object program consumption of memory space and of execution time. Furthermore, the nature of some applications, which deal with bits and bytes as operands, does not always make FORTRAN an attractive language, regardless of the quality of the implementation. Algol, Jovial and PL/I all have their supporters, and if these compilers were to run on a different, larger system, and produce code for the minicomputer, some of the implementation problems might be relaxed. As of now, only one minicomputer manufacturer is known to have provided any language other than a FORTRAN dialect for a minicomputer. Hewlett-Packard has made both ALGOL

and BASIC available. It is not the purpose of this paper to make language suitability comparisons but rather to suggest that a language more suitable than FORTRAN for typical “bit-shuffling” applications can be found, and that the associated language translator does not have to run on a minicomputer with 8K memory and no mass storage. No manufacturer has taken this approach, although several DEC users are reported to be generating object code on larger systems.

Nonprocedural approaches such as report generators and decision tables might be expected to stir considerable enthusiasm among inexperienced users. Again, implementation in a common language on a larger system seems an attractive approach.

Another frequently used, but esthetically less attractive solution, is to drop from the universal to the particular to provide a series of compilers each accepting an application dependent language which presumably would appeal to a specialized class of users. A common syntax but a varied semantics might make this approach feasible.

Nevertheless, assemblers for 4K memories and FORTRAN compilers for 8K memories are well nigh universal. A desirable approach, mentioned above, is a translator running on a large- or medium-sized computer system and producing code for a minicomputer. This translator program itself could be written in a machine independent language and thus would be capable (theoretically) of running on several different systems. The point here is not which language, but rather, the concept of operating this translator on a larger computer system with facilities more appropriate to the task of program translation. This method of program translation has been validated by the manufacturers of airborne computers, who commonly provide translators which do not operate on the target computer.

The second point to be made is that debugging should be conducted at the symbolic level: octal or hexadecimal dumps of memory are not an adequate tool for the typical user. Source language debugging seems to be the appropriate approach even though it is space consuming. Furthermore, the real-time/on-line nature of many applications needs to be recognized and appropriate tools developed, perhaps in connection with interrupt service routines.

Desirable debugging techniques can often be achieved more easily when program operation is

interpretive: that is, the computer hardwired order code is used to simulate instructions which are desirable from a programming point of view. At the present time, fixed point multiply and divide, double precision add, subtract, multiply and divide and floating point operations are commonly furnished as subroutines. The programmed operators of the SDS 900 series went a long way toward relieving the user from writing calling sequences to subroutines which provide functions not available in the hardware. This is a typical example of how software can be supported by appropriate hardware features. The high internal speeds of computers today relieve the problem of speed reduction (20:1 to 50:1) connected with the use of interpreters. Five to ten thousand instructions per second is an adequate rate in many cases. It is my prediction that over the next few years, a microprogrammed approach to this problem will remain too costly for minicomputers.

Inasmuch as programmer time spent in program debugging is generally accepted as equal to time spent in coding, the present obvious emphasis on translators at the expense of debugging packages seems somewhat out of keeping with things as they are. This will be more true with inexperienced users.

The mnemonic and symbolic debugging package furnished by DEC for the PDP-8 series seems a step in the proper direction. The fact that many of us have survived with octal dumps for the past ten years does *not* mean that this is the appropriate approach to take. The concept of the small computer as a black box in a system should include not only the words "inexpensive," "reliable," "fast," but also "easy to use." In this sense, "easy to use" should encompass the effort to check out the program which enables the minicomputer to be a part of the larger system.

Other areas can be mentioned but the most important needs of manufacturers seems to be a psychological one: the awareness that the minicomputer will, within a very few years, be contributing a much greater part of the total main frame market than it is today. And a major market deserves major support. The importance of adequately supporting this type of system can be assessed by thinking of the background of the individual attempting to use it. Scientifically trained, perhaps, but certainly not likely to be a programmer, especially in view of the present and projected shortage in this field. The outcries of

early System 360 users will no doubt seem infrequent compared to the outraged voices of those novices who don't realize that early vintage software isn't supposed to work!

Hardware enhancements

Some suggestions concerning hardware development should be advanced at this point, if only to satisfy the hardware-oriented person who has stayed with it this far.

I would like first to advance the heretical idea that 18 bits are far more desirable than 16 bits of a minicomputer word size. Those two additional bits are exceedingly valuable in the instruction word, as anyone would know who has participated in minicomputer design. Since instruction words are generally more frequent than data words it would seem reasonable to produce a computer with an appropriate word size for instructions, rather than USASCII data. Perhaps the first step toward an 18 bit instruction word would be to discard the generally superfluous parity bit. In any event, I would divide the 18 bits in the instruction word somewhat as shown below:

- 5 bits for op code
- 1 bit for indirect addressing (paging is still necessary)
- 1 bit for "interpretive mode" (op code used as an address in a transfer table)
- 1 bit for use of current page/base page
- 10 bits for address, yielding an adequate 1024 word page size

My concept of indirect addressing would permit the direct addressing of 65K words via the 16 low order bits of the indirect word referenced. However, the leading 2 bits of the indirect word also would play a part in generating the true address. These two bits would be used as follows:

- 00 this word contains the effective address
- 01 use contents of hardware index register 1 added to the low order 16 bits of this word
- 10 use contents of hardware index register 2 added to the low order 16 bits of this word
- 11 applying one more level of indirect addressing

This somewhat restricted use of hardware index registers is compatible with current concepts of paging in minicomputers, where the program is stored in one group of pages and the tabular data which is to be indexed is in a second group of

pages, which must be indirectly addressed. Non-memory instructions could be used for index stepping and testing.

The base page, of course, should be controlled by a programmable base page register. This is in keeping with a multilevel program structure, with each level of priority associated with a different base page.

Another point to be made is that most applications would be better off with 8K words of memory with a 4-8 μ sec cycle rate than 4K words with 1 μ sec cycle rate. Only rare applications demand the sub microsecond memory cycle found with some of the newer minicomputers. However, many applications require more than the standard 4K memory. Of course, present cost factors do not seem to permit this kind of tradeoff to be made. In this connection, I would like to insert a plea for a bulk read/write memory of modest dimensions (50,000 to 500,000 words) with access times ranging from 10 to 100 milliseconds. This device should be sufficiently low in price (ten to fifteen thousand dollars) to encourage wide usage. NCR appears to have made a major step forward on software development for their Century series by stating that the twin disk drives are an inseparable part of even the least expensive system.

Many minicomputers operate on numeric data which requires two or more words to express. Double precision fixed or floating operations are fairly common in some applications (process control, instrumentation). These can more easily be programmed, if when the "interpretive mode" bit in the instruction word is set, the program counter and other status information is stored in location A and the next instruction is taken from A+1. A is determined by adding some constant to the 5 bit operation code. This approach also permits "upward compatibility," if the minicomputer is the smallest number of a computer family

and can provide instruction compatibility through interpretation.

A final word concerning hardware should not overlook what I consider the most important single area in minicomputer design: I/O. The minicomputer *must* be able to communicate with a great variety of I/O devices. A reasonable approach to I/O includes the availability of a priority interrupt structure with sufficient capability such that each device can, if necessary, cause an interrupt to a different starting location. This will reduce the response time of the computer to the interrupt by removing the need for programmed "sorting out" of the interrupt signal from array several alternatives.

Such hardware must be supported by a comprehensive I/O control routine which encourages the potential user by its simplicity. Hewlett-Packard appears to have more in the right direction with their BCS package.

I/O data itself should be capable of being transferred either to memory or to the arithmetic registers when operating under a programmed I/O scheme.

CONCLUSION

The advent of medium and large scale integration and of batch memory systems, as well as improved intra-computer communication facilities will all have great impact on the future size of the small computer market. However, the greatest determiner will be the ease of use, as determined by the interaction of hardware and software. Low price and reliability alone will not satisfy the user who expects a job to be done, a task to be accomplished. Frustration after delivery may determine whether there is a five thousand unit market or a twenty-fifty thousand unit market for minicomputers in the early seventies.

The mini-computer—A new approach to computer design

by D. C. HITT, G. H. OTTAWAY and R. W. SHIRK

International Business Machines Corporation
Research Triangle Park, North Carolina

INTRODUCTION

When memories become many times faster than they are today at less cost, how will computers be organized? For those installations which require many times their current performance, present methods may suffice. However, there will be installations which don't need or can't use this much additional performance. How can they benefit from such improved memory design?

In an attempt to answer these questions, IBM built an experimental "Mini-computer" which incorporates new techniques and a unique organization. These methods could be used to take advantage of increased memory speed to reduce data flow and control hardware, lowering system cost. Although the memory actually used was neither unusually fast (2 μ sec) nor inexpensive, it was adequate to demonstrate principles.

The experimental computer (dubbed Mini) is a general-purpose, stored-program digital computer (Figure 1). However, it differs from such computers in two major respects: it has no arithmetic or logic unit and only a very elementary instruction set; and, it has only 512 bytes of storage for data and control. Mini's other specifications also reflect its name:

- Size—18 X 15 X 3 inches;
- Weight—24 pounds, excluding power supply and I/O;
- Cards—one SLT board (79 cards). If communication line control is desired, three additional cards on an external board are needed.

Although the Mini-computer is contained in one small tabletop "box," its capabilities are surpris-

ing to many people. Using a bare minimum of hardware and instructions, Mini can perform the arithmetic and logic of a central processor, plus those functions normally executed by I/O control units. Its attachments include bulk storage, a graphic display (including animation!), a keyboard/printer, and communication lines. Little additional circuitry was needed to attach these I/O units. In addition to the computer, the only hardware needed to control these devices is the analog circuitry required to electrically and mechanically interface with them (magnet drivers, sense amplifiers, voltage converters, etc.).

Finally, the Mini-computer uses only two types of instructions. Functions such as add, subtract, shift, index, and translate are all programmed with these basic instructions.

Organization of the Mini-Computer

The Mini concept starts with a small core storage device (512 8-bit bytes) around which the rest of the computer is built. (See Figure 2.) To this have been added a Storage Address Register and a Storage Data Register. The 8-bit Storage Data Register is funneled down to a 4-bit path going into the Data Register. This was done because of the extensive use of tables and the need to keep addresses compact to conserve storage.

The Data Register is the single "work" register, all data being shuttled back and forth between it and memory. It is used as an input/output register for I/O operations, as a test register for branch instructions, and as a temporary holding place for storage-to-storage transfers.

The Instruction Counter holds the address of the next instruction to be executed. The I/O Reg-

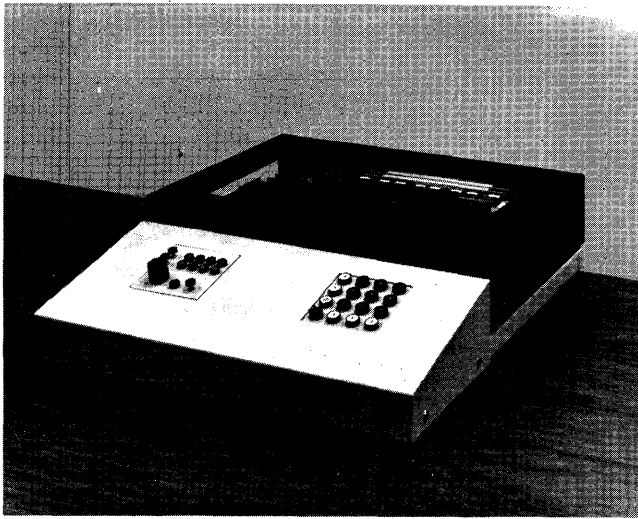


FIGURE 1a—The Mini-Computer—Without I/O

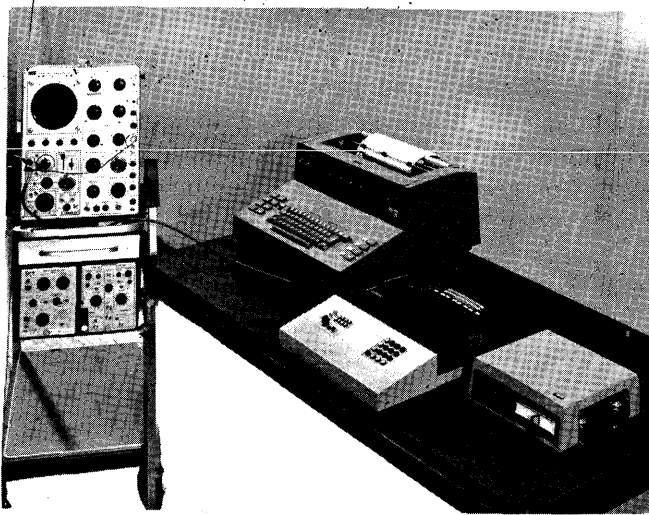


FIGURE 1b—The Mini-Computer system

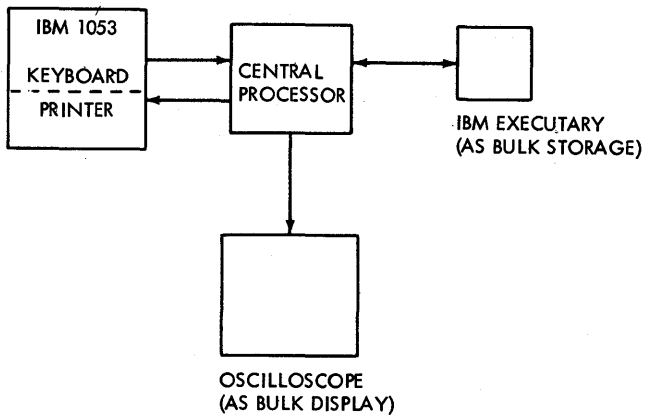


Figure 1c—Mini system block diagram

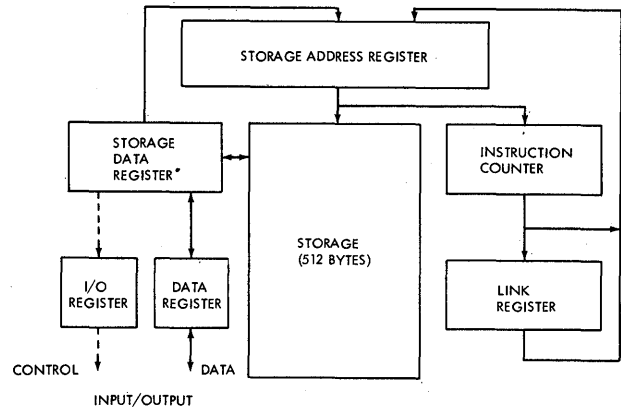


FIGURE 2—Mini data flow

ister holds the 5-bit, I/O instruction address and decodes it to select a particular device. I/O operations are thus controlled independently of the computer's other operations. Because of the limited instruction set, Mini relies heavily on subroutines to get work done. In order to get to and from subroutines easily, a way was needed to keep track of where the main program was left for re-entering. This is provided by the Link Register.

Mini has no processing unit, no index register, no register stack, etc. The hardware, in short, is accurately depicted by the name—Mini. There are, of course, the I/O units—which as I/O units generally—can be selected and tailored to fit the application.

Instruction set

The instructions, which have a single address, can be classified into two main categories: move and branch. The hardware-addressed or "implied" Data Register is associated with all instructions. Move operations transfer data into or out of the register, branch operations test data in it. The complete instruction set is shown below.

Move Instructions:

- **FETCH**—Data at the addressed storage location is moved to the Data Register;
- **STORE**—Same operation as above, except in the reverse direction;
- **INPUT**—Data from the addressed input device is moved to the Data Register;
- **OUTPUT**—Data is moved from the Data Register to the output device.

Branch Instructions:

- **BRANCH ON ZERO**

- **BRANCH ON NOT ZERO**
- **UNCONDITIONAL BRANCH**
- **BRANCH AND SAVE**—The address of the next sequential instruction is saved, and a branch occurs to the specified storage location;
- **RETURN**—A branch occurs to the storage location saved by the last **BRANCH AND SAVE** operation.

One of the two conditional branches is obviously redundant. However, having both is a convenience which aids programming and frequently conserves core space. The **BRANCH AND SAVE** instruction and associated **RETURN** permit subroutines to be performed with a minimum of programming overhead. As will be seen, subroutines play a very important role in the Mini-computer.

The basic unit of data operated on by the instruction set is the 4-bit half byte. Instructions themselves are either 1 or 2 bytes in length, with the 1-byte format accounting for roughly 70% of the instructions in a typical program.

Instruction formats

Two instruction formats are employed in an effort to reduce storage requirements. The long format uses a 6-bit operation code and a 10-bit address capable of addressing any of the 1,024 half bytes of core storage. Short-format instructions use a 3-bit op code and a 5-bit address, with the remaining five bits implicit in the operation code.

Thus, because a large percentage of **STORE** instructions is used to modify other instructions in the immediate vicinity of the store instruction, short-format store addressing can be relative to the Instruction Counter. That is, the remaining five bits of address are supplied from the high-order positions of the Instruction Counter. The same is true of short-format branches.

For short-format **FETCH** and **BRANCH AND SAVE** instructions, the required address bits are hard-wired into the computer, causing fixed storage areas to be accessed. These areas contain tables in the case of the **FETCH** instruction, or subroutine linkages or short, self-contained routines in the case of **BRANCH AND SAVE**. The short-format instructions make possible significant savings in core because, as stated earlier, they account for about 70% of all of the instructions used in a typical Mini program.

Circuits

The Mini model was constructed from readily available System/360 Model 50 circuit cards employing the Solid Logic Technology (SLT) 30-nanosecond circuit family. The memory unit chosen was a 2-microsecond, read-write core buffer used in the System/360 Model 20. No functional packaging or large-scale integration was attempted. The machine is packaged on a single SLT board and contains approximately the following 650 circuits:

Function	No. of Circuits	No. of Cards	Percentage of Total Cards
Data Flow	250	36	42
Storage	160	20	31
I/O	240	23	27

Calculation and data manipulation

Because Mini has no arithmetic or logical instructions, these functions are performed by subroutines. Core requirements for these are moderate, as shown in the following table of some of the subroutines which have been written. Of course, many more could be added.

Function	Bytes in Subroutines	Bytes in Tables
Control Counter (initialize, increment/decrement)	5	
Control binary trigger (set, reset, flip)	4	1
Control status switch (initialize, set, reset)	6	
Test Bit	4	8
Translate Code	4	8
Delay	11	
Increment Address	12	
Add/Subtract (decimal digit)	17	11
Compare (binary digit)	14	
Shift (binary digit)	9	16

Two programs have been written to show the arithmetic capability inherent in the Mini data flow and instruction set: the desk calculator and triangle calculator program. The desk calculator program accepts data from the 16-key keyboard and produces two lines of 14-digit output on the display (described below.) The first line displays

entered data, the second displays the results of arithmetic operations on the data. The functions provided are add, subtract, multiply, load, and no-op.

The triangle calculation program accepts data from the keyboard and produces up to six lines of 3-digit output on the display. The program starts by displaying a labeled triangle (Figure 3).

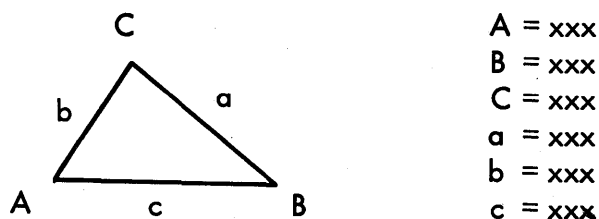


FIGURE 3—The triangle calculation program

The user keys in the specifications for the known angles and sides and these values are displayed on the appropriate lines. The program then computes the remaining values and displays them to complete the picture.

Since the triangle program is larger than the storage capacity of Mini, it requires successive loads from a bulk-storage device. Each load does a particular type of processing, such as calculating a sine, cosine, arcsine, or arc-cosine by a power series expansion or by a solution of the law of sines or cosines.

I/O unit control

Bulk storage

A standard IBM Executary dictating unit is controlled to perform functionally as a disk file. Labeled records are recorded on the belt in a fixed format, serial by bit. Synchronization, bit detection, deserialization, parity checking, label comparison, and data movement are all programmed, using 120 bytes of core. The special hardware required to attach the Executary was five cards.

When the Executary is used as a dictating machine, the user controls the machine through switches on the microphone. For use with Mini, the microphone was removed from its cable, and the wires that had gone to the microphone switches were connected to reed relays in the computer. This enables Mini to perform the same three functions by issuing I/O commands:

- Start moving the belt.

- Backspace one track.
- Turn on the write circuits.

Because the random-access, bulk storage used (Executary) is designed to record in the normal audio range, bits are recorded as audible beeps. Each "one" bit consists of three cycles of a 2K Hz tone; each "zero" bit three cycle times of silence.

Mini generates these audible tones with a program to eliminate the need for a separate 2K Hz oscillator. The oscillator program activates the output line and enters a subroutine which provides a delay of 250 usec, then it discontinues the output for 250 usec, activates it for another 250 usec, etc., for three oscillations. The recording format on tape is similar to standard start-stop communications format. Records have labels and markers to enable Mini to locate the beginning of the desired record, and a check character for validating data. The recording rate used with Mini is 40 bytes per second. Speed is limited by the Executary belt and amplifier characteristics, not by the computer. Data rates of 500 bytes per second have been achieved using a different tape transport and amplifier.

Display

The objective of the display investigation on Mini was to demonstrate the maximum versatility for drawing pictures on a simple CRT display with the least amount of additional circuitry. (See Figure 4.) The circuitry, packaged on only three cards, provides for two 8-bit digital-to-analog converters, two 8-bit deflection registers, and standard output commands for loading these registers from the data flow. In addition, there is control for blanking the beam.

The 8-bit horizontal deflection register specifies on which of 256 horizontal positions the beam is to come to rest. An identical vertical-deflection register similarly selects one of 256 vertical positions. For example, if both registers contained the value of zero, there would be a spot at the lower left corner of the tube. If both registers were then set to 256 simultaneously, the beam would travel to the upper-right-hand corner tracing a line along the way. Thus, both horizontal and vertical registers must be changed simultaneously if other than horizontal and vertical lines are to be drawn. The hardware-function tradeoff selected allows the upper four bits of both horizontal and vertical

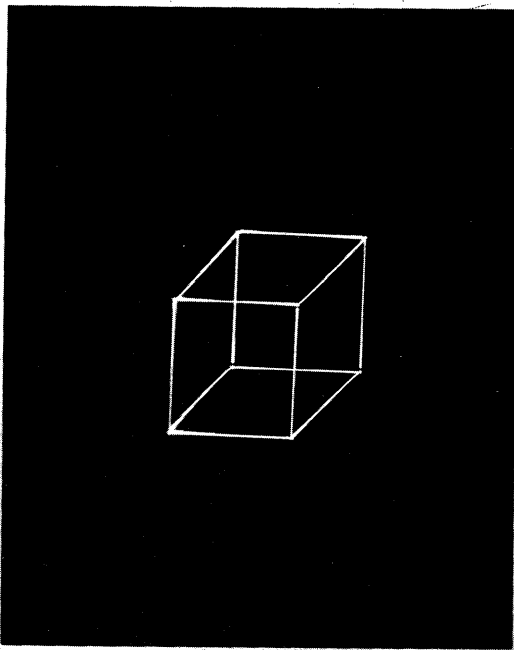


FIGURE 4—Demonstration of display versatility

registers to be set together, and the lower four bits to be set together. This provides the capability of connecting points on either a large or small 16-by-16 point grid.

A program was written to generate an alphanumeric display. The most challenging problem associated with the program was developing a subroutine for tracing each of the individual characters—a function which is generally provided by special-purpose hardware dedicated to generating a specific set of characters. For each character to be drawn by the program, a string of elements is provided. Each element defines a new point on the display. As the points of the string are successively connected, the desired character is drawn. To provide the most general character generator, each element of the string requires 8 bits, four bits for one of 16 vertical positions and four bits for one of 16 horizontal positions.

An alternate arrangement is to provide a more restricted array of end points so that each element can be specified with fewer bits. Figure 5 illustrates one such restricted set of points. Figure 6 shows the character set obtained. This set of 15 points can be specified in four bits, which reduces the core requirements of the character generator by a factor of two.

A program was written to accept characters from a keyboard and display them onto the face

.1	.6	.9
.2		.A
		.B
.3	.7	.C
		.D
.4		.E
.5	.8	.F

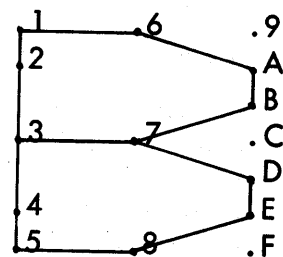


FIGURE 5a—Alphanumeric display generator matrix

FIGURE 5b—Example of how a letter (or number) is drawn from matrix

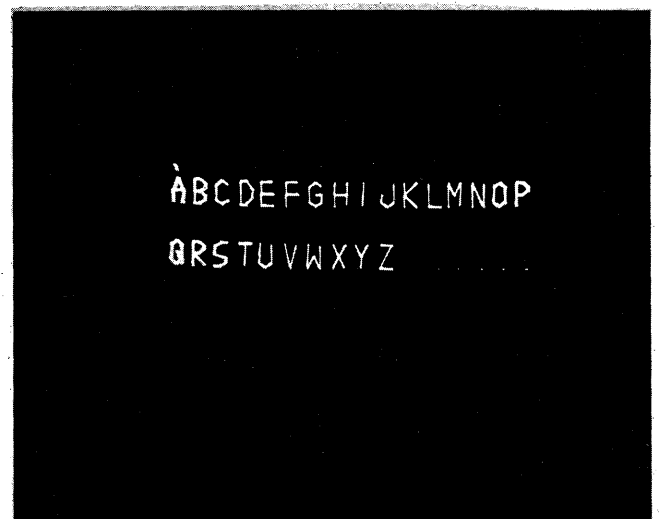


FIGURE 6—Alpha character set

of the tube. The character set consisted of the 26 capital letters and 10 digits. A cursor was provided to indicate where the next key to be depressed will be displayed. The space key moves the cursor incrementally along the line and the carriage-return key returns the cursor to the left and down one line. A reset returns the cursor to

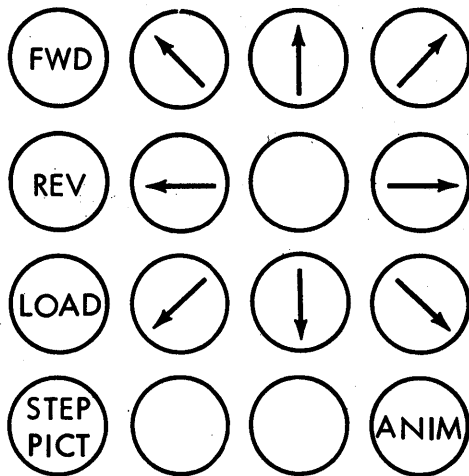


FIGURE 7—Mini keyboard interpreted for drawing program

the upper-left-hand position of the tube. Only 352 bytes were required for the programs and tables for this program.

A program was also written to permit an operator to draw pictures on the tube. This program interpreted the general purpose, 16-key keyboard as shown in Figure 7. By depressing one of the arrowed buttons, the operator can move the last end-point of his picture one increment in the direction indicated by the arrow. When the operator is satisfied with the position of the last point, he can freeze that point by depressing the load key. This stores the coordinates of the last point and creates another moveable point at the same location.

The forward and reverse buttons allow the operator to modify previously fixed points. Up to 16 different pictures containing a total of 128 end-points can be stored in the available buffer space. Animation can also be obtained by rapidly flashing the 16 pictures. This program requires only 256 bytes of program and 128 bytes of buffer.

Printer/Keyboard (IBM 1052/1053)

The Mini-keyboard interface consists of six Binary Coded Decimal data lines, a "key depressed" signal line, and a "keyboard reset" line to reset the keyboard between key depressions.

To accept data from the keyboard, the Mini program tests the "key depressed" line. When the "key depressed" line is up, the program enters a delay subroutine to allow the data lines to settle to a steady state. Then, the program transfers the information on the data lines to storage and issues

a reset signal to prepare for the next key depression. This sequence requires only nine bytes of storage in addition to the delay subroutine.

The Mini-printer interface has six data lines which position the print element; seven control lines corresponding to the functions of print, shift up, shift down, tab, carriage return, space, and backspace; and a "carriage in motion" line. The positioning information specified by the program is held in a special output register for the duration of a delay subroutine.

To use the printer, Mini raises a control line and holds it active for the duration of the delay subroutine. For the print function, the program first transfers the proper combination of bits for the character to be printed from storage to the output register. For tab or carriage return, the program tests the "carriage in motion" line and waits until it is no longer active before proceeding.

The Keyboard/Printer requires five cards.

A program to couple the 1052/1053 keyboard and printer together for normal typewriter use has been written. This program accepts data from the keyboard and immediately outputs it to the printer so that the data keys print and the function keys perform as in a normal typewriter. As an intermediate function, the program translates the BCD input code to the required output code before setting the output register. The program requires 54 bytes in addition to 45 bytes for subroutines and tables.

Sixteen-Key Keyboard

The keyboard used on Mini contains 16 general-purpose keys. Four bit-line outputs are available to the program, as well as an output directly from the zero key. The separate zero-key output, coupled with some additional programming, eliminates both extra keyboard hardware (no second contact needed on each key to indicate a key is depressed) and extra circuitry (no keyboard strobe).

To control this keyboard, Mini first tests the four data-line outputs. If no information is available, the program tests the zero-key output. If all lines are inactive, the program then repeats the above tests. If any data line or the zero-key line is active a delay is taken to insure steady-state values on the lines. The information is then trans-

ferred to storage, or if the zero key is active, a zero is inserted into storage.

The program requires 15 bytes in addition to the delay subroutine.

Communications

A standard common-carrier data set was attached with three cards. A program was written to enable communication between Mini and an IBM 1050 terminal.

Programming

Address modification is used extensively in programming Mini. The following example illustrates its usefulness:

Location	Operation	Address
x	FETCH	Increment Table
	STORE	x + 1

The first instruction is a short-format FETCH with op code at half-byte location x and address at half-byte location x + 1. This instruction addresses an 8-byte increment table at a fixed location in storage. The position in the table is determined by the address at location x + 1. The table is as follows:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Address
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	Data

The second instruction is a short-format STORE. It inserts the half-byte fetched by the previous instruction into the instruction stream at location x + 1, thereby modifying the address portion of the FETCH operation.

Assuming the half-byte at x + 1 is initially "0", the above sequence fetches a "1" to the Data Register, then modifies itself to fetch a "2" next time. Now, using the BRANCH ON NOT ZERO instruction, a loop can be programmed:

Location	Operation	Address
x	FETCH	Increment Table
	STORE	x + 1
	BRANCH	x
	NOT ZERO	

The BRANCH ON NOT ZERO instruction tests the Data Register and branches back to the start of the sequence as long as the Data Register contains a non-zero quantity. This will be the case as long as the half-byte at location x + 1 is not an "F." Once this becomes "F" the FETCH instruction will load "0" into the Data Register and the branch will fail. Assuming a "0" in this location initially, the loop will be executed 16 times with successively higher numbers being loaded into the Data Register.

The table used in this example is one of two which can be accessed with the short-format FETCH instruction. The other table is organized for decrementing and is as follows:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Address
F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	Data

The counting example could just as easily have been programmed using this decrement table. It would then count through the loop 16 times but would load successively smaller numbers into the Data Register.

The programming techniques used with Mini will not be described in detail. However, an example of the use of stored tables and the basic instruction set will clarify the manner in which processing is accomplished. The following routine performs binary addition on two half-bytes:

Location	Operation	Address
x	FETCH	Increment Table
	STORE	x + 1
y	FETCH	Decrement Table
	STORE	y + 1
	BRANCH	x
	NOT ZERO	

Let us assume that the two numbers to be added have been loaded into locations x + 1 and y + 1 in the instruction stream. The first number is incremented, the second decremented and tested for zero. This sequence continues, the first number being counted up and the second counted down, until the loop is ended by the second reaching zero. At this point the sum is at location x + 1.

The increment/decrement tables used in the

previous examples are the most commonly used in Mini and are therefore stored at fixed locations addressable by the short format FETCH instruction. Other tables, however, are constructed for special purposes and are accessed by the long-format FETCH. Examples of such tables are those used for code translation, shifting, decimal incrementing and decrementing (or of any other base, for that matter), bit testing, overflow detection, masking, etc. Other examples will no doubt suggest themselves to the reader.

From this discussion, it is obvious that Mini is a table-driven system. Because of this reliance on stored tables for basic data manipulation and because of the very limited storage area available, it was imperative that tables be kept as compact as possible. This explains why half-byte addressing is used with storage. Four-bit table addresses allow the use of 8-byte tables such as the increment and decrement tables described previously. Had storage addressing been at the byte level, these tables would each have required 128 bytes!

CONCLUSIONS

Mini, a small, experimental computer with a surprising amount of capability, has been built

around only two classes of instruction—move and branch. Arithmetic and logic functions are performed by table-look-up subroutines. Digit addressing makes it possible to use much smaller tables than ordinarily used, conserving memory.

No additional instructions or hardware are necessary to operate the computer. Mini has thus demonstrated that it is possible to build a computer with no arithmetic or logic hardware! Further hardware savings—in memory—have been made by using one-byte instructions (part of address being hard-wired or implied by operation code).

In the I/O area, Mini has shown that special-purpose hardware can be replaced by general-purpose data flow and that—once you have the data flow—little additional circuitry is needed to control I/O devices. The Mini-computer, because of its inherent simplicity, makes it possible to approach an I/O task—often quite complex—as directly as though it were a CPU operation.

Mini is smaller than most of the terminals and I/O units used with present-day computers, even though it wasn't built from miniature components. Mini may have indicated the shape of computers to come.

The Lockheed hybrid system—A giant step

by CONRAD K. BEDIENT and LARRY L. DIKE

Lockheed Missiles and Space Company
Sunnyvale, California

INTRODUCTION

View of the future

Gazing into our crystal ball we see you seated at the analog computer preparing for a scheduled hybrid run. You alert the digital computer through your remote display. The digital computer recovers from its permanent files the data and source files associated with your run. These files include a description of your problem in equation format, a description of your problem in block diagram format as mechanized on the analog computer, the source hybrid problem, and data decks. After logging on the system, a complete set of hybrid software is available to you. You may run your program all digital using the equation input and the analog mechanization input to verify proper analog implementation of your equations. Using the outputs of the digital simulation ranges of variables may be determined for automatic scaling. The digitally simulated analog mechanization can be used for a patchboard and equipment verification of the actual analog connectives. Static checks may also be generated. Using the outputs of your digital simulation, you can test the various transfer blocks synthesized on the analog computer in a dynamic mode.

Using a set of diagnostic software, you interact with the digital computer until your analog computer is set up and your problem is debugged. You now request the system to load your production program. You select the options of the program you desire, and make your production run. All files and data required for the run are maintained on the mass storage device. When your program is not in execution, it is rolled-out onto the mass storage where it may be interrogated and modified as though it were in central memory. All source files may be modified both from re-

mote or by update card decks. The system maintains all these files and makes them available to you on request. The digital computer is remoted from the customer and it furnishes computing capability as though it were a power source always responding to your demands. The flexibility gained from the display gives you the feeling of having control of the computer.

This description indicates the developmental direction of the Hybrid Laboratory at Lockheed Missiles and Space Company (LMSC) located at Sunnyvale, California. The large digital computer system approach to hybrid computing may permit new flexibility and efficiency that will allow a large digital system to be used economically in hybrid facilities. The only thing that seems to stand in the way of such an operation is the development of the software systems to support such a laboratory.

Problems in hybrid computation

When LMSC's system became operational in June 1967, the primary problems of concern were the problems of modifying a batch processing system to allow for real time computation, the sharing of the central processing unit between two hybrid problems, and the writing drivers for the linkage equipment—these problems although severe were solved and allowed the development of hybrid simulations. The principal step into the future was the development of a remote keyboard entry device. This development paved the way for a glimpse of what a future large hybrid system should be.

In debugging a large hybrid problem, batch processing techniques of debugging may be discarded as virtually useless. A hybrid program is a complicated program which cannot normally be

halted at a point for examination but must continue its dynamic solution. This complicated flow of information between the continuous running analog computer and the serial digital computer requires more refined methods of debugging. The size of the codes being written for simulation is large, since these programs are generally an attack at a large complex system which merged the complications of the predecessor multi-rack analog simulations with the large digital simulations. The total problem has become fantastically complicated. Large amounts of data must be handled both flowing from the simulation and into the simulation. Large programs and data decks must be handled. The programmer not only faces the task of solving the problem for a single run but also of controlling the job to allow for the proper sequencing of runs. The program must prepare the results from the runs directly for reports so that the number of runs may be increased without the burden of a large amount of data reduction. The development of hybrid codes is a large effort. The hybrid lab which can support two production hybrid jobs has many hybrid jobs under development. The jobs under development must be able to be sandwiched between production runs utilizing both real or simulated equipment. All the debugging techniques for the hybrid jobs must be available for jobs under development.

Use of the digital computer

The digital computer should be used efficiently. However, with hybrid simulations the problem is in the operate condition less than 50% of the time. Although the time must be scheduled, the very nature of the analog computers in simulation techniques make it impossible to determine when the digital computer will be required.

Yet, when the digital computer is required, it must be furnished immediately, since the basic cost of the system is the analog equipment. Many systems solve this problem by batch background processing. This procedure may have several appealing merits, but it is probably dangerous. The customer's diversified and dissimilar organization goals and interests, and the increased costs of equipment to support both batch processing and hybrid simulations plus the complications to the system software may compromise the hybrid requirements. Basically in a hybrid lab containing many analog computers even a large digital computer is not the high cost item in the operation; therefore, the main effort should be to use the

digital computer to increase the efficiency of the analog portion of the overall operation.

These are the problems that face a large hybrid lab. These problems require programming systems completely beyond the capabilities of those currently furnished by hybrid computer equipment vendors.

Reading guide

The material presented in this paper has been organized to accommodate both the reader interested in a general description of the Lockheed Missiles and Space Company (LMSC) Hybrid Computer system and the reader interested in studying the system in detail. For a general description of the main features of the system it is suggested that the sections I through III and VII be read. For those who wish to study the system in detail, Sections IV through VI are provided.

We have gazed into the crystal ball of the future. Now let us take a look at the organization and implementation of the LMSC hybrid computer system.

Hardware description of the LMSC hybrid computer system

The hybrid computing system at Lockheed Missiles and Space Company,* Sunnyvale, California consists of the following:

- Control Data 6400 digital computer system
- Two Comcor Intracoms (linkage equipment)
- Four Comcor Ci-5000 analog computers.

A block diagram of the system is presented in Figure 1.

Digital computer

Figure 2 shows the organization of the Main Frame. At the center is a large floating point processor, the central processing unit (CPU). The multi-register organization of the CPU allows fast execution of Fortran programs. The CPU communicates into 32K of 60 bit memory. Surrounding this central memory are 10 peripheral processors (PPs). These PPs are identical and operate simultaneously and independently as stored-program computers. They act as system control

*Hybrid computation at LMCS began in 1966 with a system comprised of a CDC 3200 digital computer system and two EAI 231R analog computers interfaced by means of a Comcor Intracom.

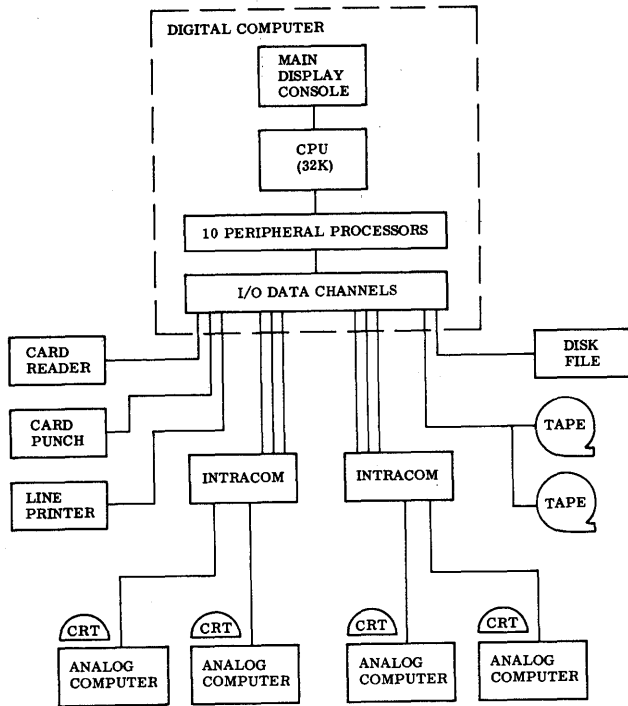


FIGURE 1—LMSC hybrid computer system

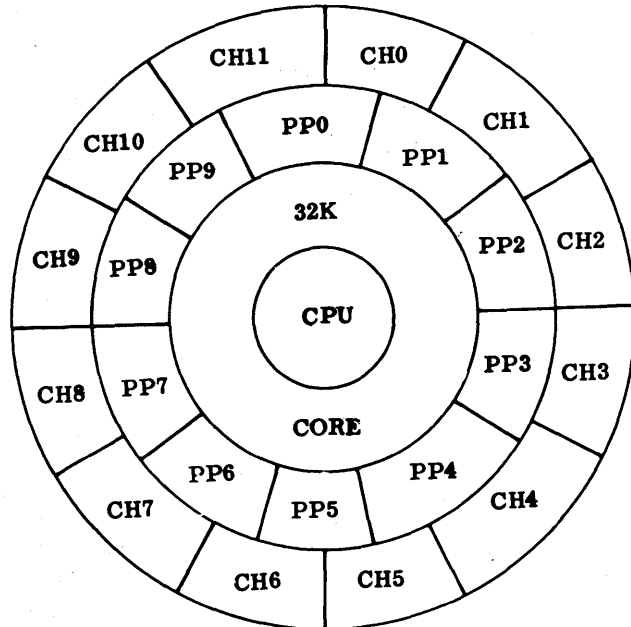


FIGURE 2—Organization of main frame

computers and as I/O processors with a 1 microsecond memory cycle time. The PPs have an instruction set which includes integer addition and subtraction but not multiplication and division. These PPs each have 4K of 12 bit memory and

may communicate directly with the large central memory or external equipment over 12 bi-directional data channels.

Table I specifies the peripheral equipment associated with the CDC 6400.

<i>Peripheral</i>	<i>Description</i>
Disk file	74 million characters
Card Reader	1200 cards per minute
Card Punch	250 cards per minute
Line Printer	136 characters per line, 1000 lines per minute
2 Magnetic Tape Transports	800,556, or 200 BPI, 75 inches per sec.
5 Remote Visual Display Units	Cathode Ray Tube (CRT) display with 20 lines, 50 characters per line and with typewriter keyboard input
Main Display Console	2 display screens for the operator of the digital computer

TABLE I—CDC 6400 peripheral equipment

Linkage equipment

The communication link between the CDC 6400 digital computer and the Ci-5000 analog computers is the Intracom which is shown in block diagram form in Figure 3. The LMSC hybrid computing system has two identical Intracoms with each Intracom normally controlling two analog computers. Each Intracom is divided into three main systems: I/O, Interrupt, and Analog. Each of these three systems is connected to its own CDC 6400 data channel for communication with the digital computer.

The I/O system is subdivided into six subsystems as follows:

1. I/O interrupt subsystem—which contains 24 priority interrupt lines and logic for handling these priority interrupt lines for processing by the Hybrid Input-Output PP program.
2. Discrete subsystem—which contains 68 discrete read and 68 discrete write lines.
3. D/A subsystem—which contains 40 digital-to-analog converters and 8 independent transfer controls.
4. A/D subsystem—which contains an analog-to-digital converter with 32 input channels

(can be expanded to 64) and 8 sample/hold controls.

5. P.I.G. subsystem—which contains a precision interval generator (real time counter).
6. Intracom mode subsystem—which contains logic for controlling and slaving the modes of the Intracom and the analog computers.

The Interrupt system contains logic for handling to 24 priority interrupt lines (previously mentioned) for processing by the Hybrid Monitor PP which controls the priority subroutines to be executed.

The Analog system contains the logic necessary to link (interface) the analog computer to the digital computer and allows for the following functions to be accomplished:

- Setting or reading a mode
- Setting or reading an address register
- Reading a DVM
- Setting a Reference Dac
- Setting a Potentiometer
- Reading status conditions of the analog console.

Each Intracom also contains a patchable logic group which is comprised of the components listed in Table II.

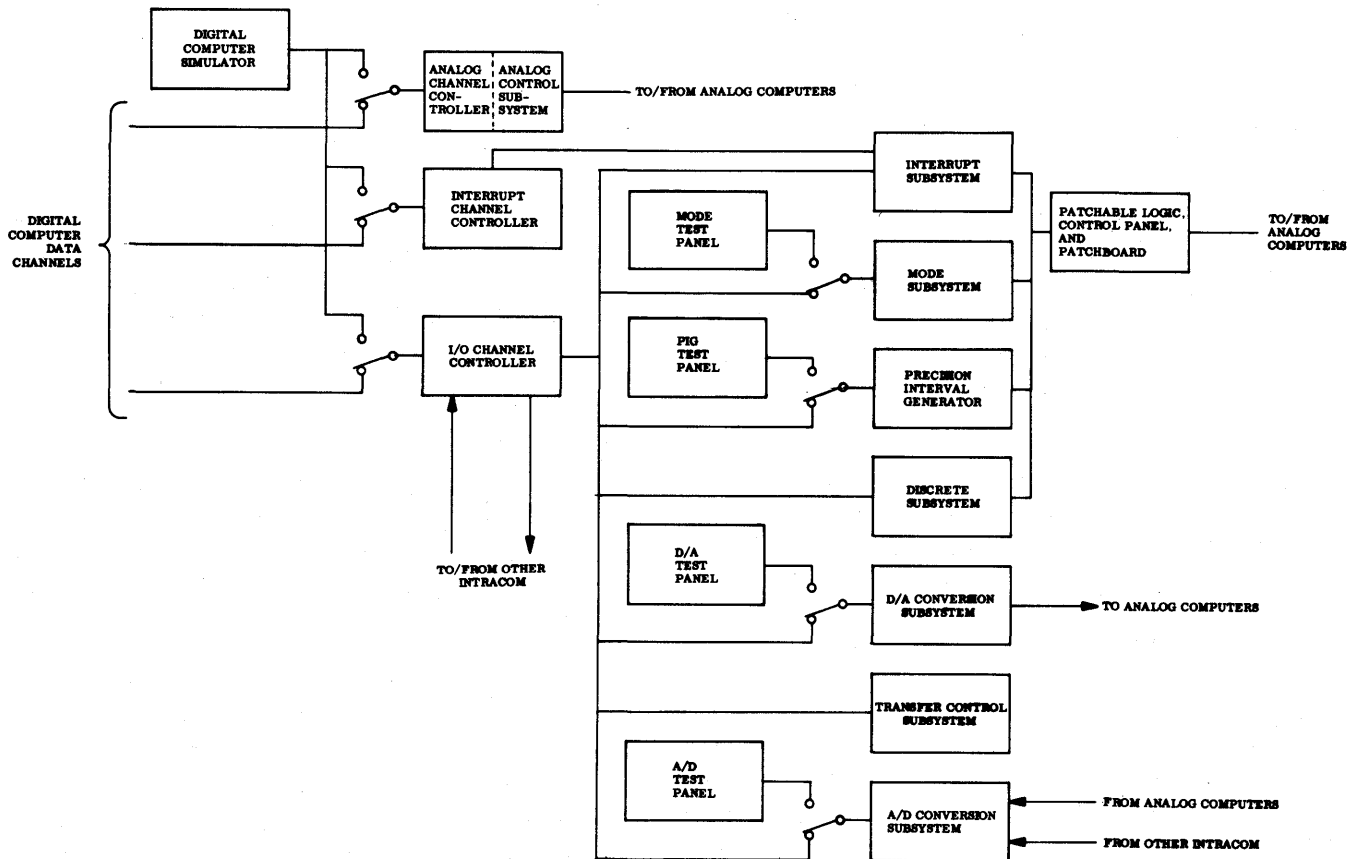
<i>Component</i>	<i>Quantity</i>
General purpose counter ----- (4 bit binary or BCD Up/down preset counter)	36
General purpose flip-flop -----	72
Variable delay flops -----	12
Logic function switches -----	24
Lamp drivers -----	48
System clock (100KC) -----	1
Nand gates -----	228
Inverters -----	72

TABLE II—Intracom patchable logic

Analog computer

The analog computing elements are solid state and operate over the range of ± 100 V. Solid-state

FIGURE 3—Ci-5000 intracom system



switching and digital logic circuitry is used to obtain high-speed computing operation. Each of the Ci-5000 analog computers is comprised of the components listed in Table III.

<i>Component</i>	<i>Quantity</i>
Summing amplifiers -----	60
Integrating and/or summing amplifiers -	60
Inverting amplifiers -----	16
Servo set potentiometers -----	144
Digital attenuators (HODADS) -----	48
Three terminal manual potentiometers -	32
Multipliers -----	66
Resolvers	
sin-cos -----	3
multipliers -----	24
Card set function generators -----	22
Card set surface generator -----	1
D/A switches -----	30
Comparators -----	30
Relays -----	24
Function switches -----	16
Feedback limiters -----	30
Patchable logic	
Up-down counters (4 bit) -----	8
or general purpose flip-flops -----	32
Counters (4 bit) -----	4
Shift registers (4 bit) -----	4
Delay flops -----	9
Inverters -----	24
Nand Gates -----	48
Cycle counter -----	1
Iterative clocks -----	3
DAC output trunk patchboard	
terminations -----	20
ADC input trunk patchboard	
terminations -----	16
Recorders -----	3
Oscilloscope -----	1

TABLE III—Ci-5000 analog computer components

Hybrid computer system and software capability

The operating system for the LMSC hybrid computer system is CLASH,* a modified and aug-

*The acronym "CLASH" originated by combining the initials of the companies who supplied programming personnel to the LMSC software development: Control Data Corporation, Lockheed Missiles and Space Company, Astrodata/COMCOR, Spectrodata. (The "H" originates from Hybrid).

mented version of SCOPE 2.0 which is a standard operating system for CDC 6400/6600 digital computers.

A measure of the effectiveness of the operating system is its ability to handle more than one problem and its ability to immediately process background digital work when there is a break in the hybrid action. An effective display system greatly simplifies program debugging plus allowing on-line debugging without sacrificing system efficiency. Extensive debugging can be done on a hybrid program utilizing the display system without the digital program being resident in core storage. Programs can be developed rapidly since their development is in Fortran language with a complete set of library routines allowing flexible communication to the linkage and analog equipment. The use of the PPs to augment the standard system in recording of data and driving of special interface or simulation equipment enhances the processing capability of the system. Continued expansion of the debugging capabilities can take place since these capabilities are largely implemented on PPs which do not reduce the basic computing power of the CPU.

Selection of the digital computer

The main criterion for selection of the digital computer and the design of the software system was the digital computer's capability of rapidly executing a scientific "Fortran" program at high interrupt frequencies, with low digital computer overhead for priority interrupt processing.

The central processor within the CDC 6400 digital computer has the capability of transferring between two programs by use of an exchange jump command. This command can either be issued by the CPU itself, or by a PP. By executing this command, the CPU can be switched between two programs at a very rapid rate. By utilizing this capability in a priority interrupt scheme, LMSC has effected high speed frequency computation of the digital portion of a hybrid program at low system overhead.

Parallel computation and I/O

A powerful feature of the computer is its multi-processor organization. The CPU, with its high speed 60 bit floating point arithmetic, is used for computation. The PPs are used for control and transfer of data in and out of the digital computer. This multi-processing structure has been implemented in the hybrid system to allow

computation within the CPU to proceed simultaneously with input/output of data via the PP (parallel computation and I/O). This capability along with the capability of rapidly switching the CPU between priority computational levels, allows extremely effective use of the CPU.

A computational frame normally includes input of data from the analog computer, computation on that data, and output of the data back to the analog computer. Two of these functions, the input and output of data, can be done effectively with PPs. The computational portion of the cycle must of course be done by the floating point CPU. If the inputting of data from the analog computer is initiated completely without intervention of the CPU, then time normally used for control of the I/O channels in a standard system is saved. By assigning a PP to each job, processing capability can also be increased. Each of these PPs can be performing input/output in parallel with one another and with the computation being performed within the CPU.

Figure 4 shows the utilization of the CPU and the PPs when we have two hybrid jobs in execution. Program A and Program B each have an individual PP, dedicated to the input and output of data, associated with them. If we look at the operation of the processors on these programs, we see parallel CPU computation and I/O transmission. For example between 1 and 2 milliseconds both PPs are transferring data while the CPU is processing the highest priority program. This assignment of PPs to each hybrid job also insures that the computation will occur someplace within the required frame time. Therefore if outputs can always be updated at the next frame time, we can guarantee uniform performance

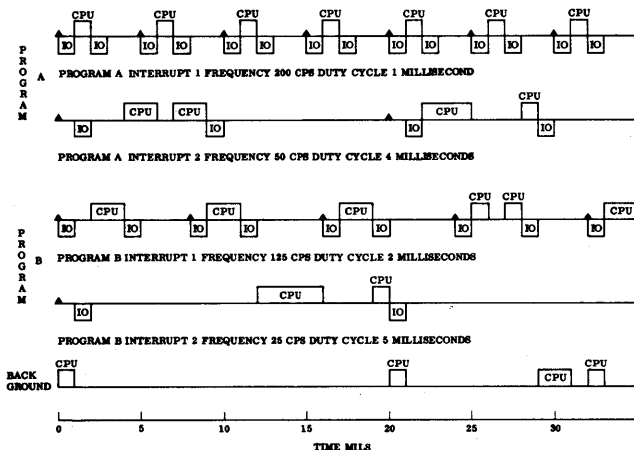


FIGURE 4—Parallel computation and I/O

with the job operating by itself or operating in concert with the other hybrid job.

The multi-processing approach also leaves more time available for the CPU. For example, if we look at the highest priority interrupt, Program A—if the CPU was required to control the inputs and outputs to the linkage equipment, it would be required 60% of the time rather than the 20% shown. This would mean that there would not be enough CPU time to allow both hybrid jobs to run simultaneously.

Automatic roll-out

Continued emphasis in the LMSC hybrid system has been placed on the utilization of the digital computer's small memory. This has resulted in major modifications to the standard CDC 6400 operating system and major use to be made of the mass storage device. One new feature added to the LMSC system is the automatic roll-in and roll-out capability. This automatic roll-out feature is further augmented by the capability of using all the debugging features of the display system while a program is in the rolled-out condition. Whenever a program comes to a point where it can no longer continue without programmer assistance, it is automatically rolled-out onto the mass storage device. As soon as the programmer has input to the display device, the system automatically rolls the job in and it continues execution. Rolling hybrid jobs in will automatically roll-out background digital jobs thereby immediately making central memory available. However, if two hybrid jobs conflict for central memory, the requesting hybrid job is delayed until the amount of central memory requested is available. Since hybrid jobs are not in execution a large percentage of the time, automatic roll-in and roll-out has greatly increased the hybrid system's efficiency.

The display system

The heart of the usability of the LMSC hybrid system is the display system. The hardware of the display system includes a controller with five remote visual display units which allow input via a typewriter type keyboard. Each of the remote display units has the capability of displaying 1,000 characters on a 50 character by 20 line screen. Regeneration of data on the displays is carried out in the display system's controller by using line memories; thus, the digital computer is only required when an update of the

display is to be made. A peripheral processor, PP no. 6, drives all five displays in a time shared manner. A small amount of central memory is also used at control point 6. The display system software consists of a display monitor program, which controls the overall operation of the display system, and a number of tasks which have been implemented to operate under the system. The number of tasks will be increased as new techniques are developed. The display monitor allows for five tasks to be in operation at each of the display consoles. The partial results of the five tasks are maintained on the disk for each console; therefore, requests made in one task are not lost when the operator temporarily switches to another task. The tasks available on a display include display directory, job control, variable display and modification, program I/O, source modification, hybrid utility, day file display, preventive maintenance, and engineering aid.

Job control

The job control task allows general control over the operating system. All of the features which are available at the main console display are available from this task. Control of a program is easily and effectively implemented since the operating system itself was designed to be controlled from a display. This task includes:

- Display of memory in octal or mnemonic format
- Entering of control cards
- Advancing of control cards that have been read in with the job
- Breakpointing through a program
- Requesting the output files to be listed

Variable display and modification

The variable display and modification task allows the programmer to display and modify variables by their Fortran name. The symbol table is output by the compiler and is included in the binary deck of the program. When the loader is loading the program, the symbol table is processed and relocated while the program and a symbol file is written to the disk. This symbol file is used by the variable display task to allow any named Fortran variable to be called up from the remote display. Variables may be displayed in octal, integer, or floating point formats. Simple mathematical functions are available including add, sub-

tract, multiply, divide, sine, cosine, and log. Variables may be displayed and modified when the program is in central memory and also when the program is rolled-out on the mass storage device. This latter feature allows the programmer to do extensive debugging without utilizing central memory.

Program Input/Output

The program input/output task allows the programmer to communicate with the remote display as though it were a standard piece of peripheral equipment. In a Fortran program he may make read or write statements which when executed will write out results or read in control information to or from the display. This is the normal way in which program control is maintained in LMSC's hybrid system. The programmer normally selects most of his options by reading in information using the program input/output display task. Whenever a request to read is made to the display, the display system automatically rolls the program out onto the mass storage device. When the requested input has been typed in by the programmer/operator the display system will roll the program back into core and the CPU will continue execution of the program following the read statement.

Source modification

This task allows the programmer to make small modifications to his source programs, both data decks and program decks. Modifications to these decks are made by using insert, replace, and delete statements. This feature allows minor program changes without requiring the resubmittal of the program decks.

Hybrid utility

The hybrid utility task displays information required by the Hybrid Monitor and also results coming from the Hybrid Monitor. The task allows the programmer to display the frequency of each priority interrupt line, the expected length of execution of each priority interrupt line, the actual length of execution for each priority interrupt line, and the number of priority interrupts that have been processed on each interrupt line. Other information which can be displayed includes the exchange package sets for each priority level in the hybrid problem.

Day file

When system errors occur which are critical to the programmer/operator, they are output to him on a display in the form of day file messages. Day file messages include such information as potentiometer did not set, equipment is not in a ready condition, etc. The messages for all jobs also appear on the main operator display console to inform the digital operator of the current status of the digital computer. The messages which appear on the remote display apply only to the job to which the particular remote display is assigned.

Preventive maintenance

Preventive maintenance (PM) for the analog computer and Intracom is also controlled using the display system. The preventive maintenance task was implemented to allow the operation of a set of maintenance routines, by technicians who are not familiar with the operation of digital computer equipment. When the preventive maintenance task is selected, the directory of the various PM tests available is displayed. The technician may select one of the tests by typing in a two digit number associated with that test by the directory. When the particular test is loaded, the reminder of what setup must be made on the equipment before the test can be run are displayed, for example, what patchboards must be mounted on the equipment, a reminder to place the equipment in digital computer control, or whatever is required for the test.

Engineering aid

The engineering aid task is basically a maintenance support program. This task is also useful when connecting new or additional equipment to the hybrid system. Since the LMSC hybrid system exists in an analysis, design, and evaluation environment, it is periodically required to connect to various pieces of hardware not normally considered as part of the hybrid system. The engineering aid task allows scope type debugging of a piece of equipment, where the instructions to be executed may be keyed in directly on the remote display by the engineer debugging the experimental equipment. The display system uses an additional PP to execute these tests. This capability has allowed new pieces of equipment to be checked without interfering with hybrid system's operation.

Software organization

The software for the digital computer is organized as shown in Figure 5. First we will discuss the organization of central memory as used by the central processing unit (CPU). Then we will discuss the assignments of the peripheral processing units (PPs) in the system. Finally we will describe how a job is processed by the system.

Organization of central memory

The central memory is divided into two main areas: resident and program. Within the resident are seven control areas for 7 digital programs. These control areas are called control points. Storage in the program area may be assigned to the control points. An area immediately following the central memory resident is assigned to control point 1; control point 2 immediately follows control point 1 and so forth up through control point 6. The top portion of memory is assigned to control point 7. As storage is required at a control point, higher numbered control points are moved upward in memory.

Hybrid control points

Two control points, numbered 1 and 7, are used for hybrid jobs. This assignment is made in order that their storage areas need not be moved as new jobs are entered or deleted from the system.

System control points

Two control points, numbered 2 and 3, are normally used by the system. Control point 2

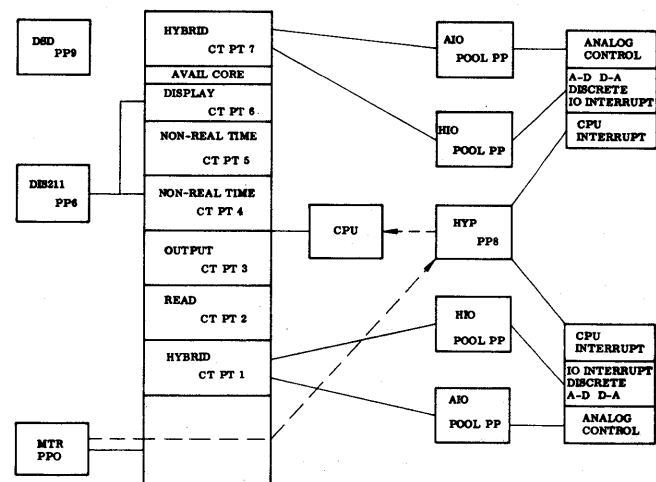


FIGURE 5—Software system organization

(READ) contains the software which processes inputs from the card reader. Control point 3 (OUTPUT) contains the software which processes all line printer and card punch files.

Non-hybrid control points

Two control points, numbered 4 and 5, are used for non-hybrid and non-real time jobs, including those of the straight digital or analog setup type which do not require real time response.

Display control points

A control point, numbered 6, is used to drive the remote display system. This control point is utilized to satisfy the CPU requirements of the display system.

Peripheral processor assignments in the system

Four of the PPs have permanent assignments while the others (6) are termed pool processors. Processors with permanent system assignment are loaded when the digital system is initialized and continually perform their assigned functions. Pool processors are assigned to perform certain tasks which may be either system tasks or input/output tasks. On completion of the task, they are returned to the pool.

Processors with permanent system assignment

The Montior Program (MTR) resides in PPo. This monitor program coordinates the operation of the entire hybrid system. It controls the assignment of central memory, coordinates the utilization of the digital computer's data channels by the other PPs, assigns tasks to the pool PPs, and controls the switching of the CPU between the control points of the system by passing exchange jump requests to the hybrid monitor in PP8.

The Dynamic System Display Program (DSD) in PP9 maintains a dynamic system display of all jobs on the main console that are in progress in the system. The CDC 6400 digital computer is unique in the fact that it has no hardware console, either for maintenance or for controlling its programming system. Therefore all of the control of the system must be exercised through a programmable Cathode Ray Tube (CRT) type-writer display.

The DIS 211 program in PP6 in conjunction

with the central processor at control point 6 drives the 5 remote displays.

The Hybrid Monitor Program (HYP) resides in PP8. Its function is to control the CPU. It directly switches the CPU real time computations and accepts requests from the MTR program in PPo for switching it between control points when real time requirements are satisfied.

Pool processors

Pool processors perform most system tasks and process input/output requests for the CPU. Two pool processor programs HIO and AIO are shown in Figure 5 and discussed below to emphasize the fact that they perform communication with the linkage equipment.

The Hybrid Input-Output program (HIO) is loaded when a hybrid program goes into real time and remains resident throughout the real time portion of the program. It processes communication requests as initiated by the CPU or the priority interrupt system in the linkage equipment. These requests are generated by a library of Fortran callable subroutines. At completion of the real time portion of the hybrid program the PP is returned to the pool.

The Analog Input-Output (AIO) routine processes all requests for non-real time setup of the analog computers. This program processes a buffer of requests generated for it by Fortran callable CPU subroutines. On completion, the PP is returned to the pool.

Job processing

Figure 6 shows the normal flow of jobs in the system. Jobs are entered into the system through the READ control point (2). A PP is periodically

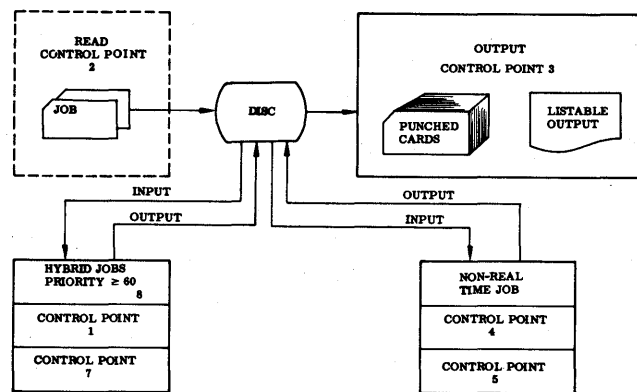


FIGURE 6—Job flow

assigned to the READ control point to see if any jobs have been placed in the card reader. If a job has been placed in the card reader the entire deck is read in and placed on the mass storage device. This file of data is then entered into the system directory as an input file with the priority that appeared on the job card. Control points which process jobs are; control points 1 and 7 for hybrid jobs, and control points 4 and 5 for non-hybrid jobs. Determination of whether a job is a hybrid job or not is made from its priority. Jobs with a priority equal to or greater than 60 are assigned to the hybrid control points (1 and 7) and jobs with priorities less than 60 are assigned to non-hybrid control points (4 and 5). Control points which are idle periodically have a PP assigned to them to determine whether they can accept a job at the control point. This PP scans the mass storage directory looking for input jobs. The hybrid control points (1 and 7) look for input jobs with priorities equal to or greater than 60 while control points 4 and 5 look for priorities of 60 or less. If an input file is found and the core storage required is less than the core storage currently available, the job is assigned to the control joint and PPs are called up to begin operations as indicated by the job's control cards.

While the job is in execution it may generate punched cards or listable output. This data are written on the mass storage device. When the job is completed, or under control of the remote display system, these files are tagged in the system directories as output files, and are released from the control point at which they were generated. The Output control point, control point 3, processes all of these listable and punched files. Periodically a PP is assigned to the Output control point, in order to search the system file directories for files which are designated as output files. When output files are found, they are printed or punched.

Program organization

The desire to run two hybrid simulations and to be able to do other digital background processing places emphasis on efficient utilization of the digital computer's small 32K core memory. Since the hybrid programs under development have an extended life of operation, they must be developed in such a way as to efficiently utilize core storage. The use of the remote display as the control device in a hybrid program also affects

the program organization. Jobs are loaded into the system and remain in operation on the system throughout the entire work shift. When a hybrid program is to be activated instructions are given to the digital computer through the remote display. A hybrid program therefore must be divided into a number of phases with these phases organized into overlays for more effective core utilization (See Figure 7). Since the real time overlay of a hybrid program is the largest overlay, emphasis must be placed on keeping this overlay as small as possible. This is done by insuring that there are no Fortran input/output statements, analog setup statements, or any other coding which is not absolutely necessary during the hybrid program's real time phase. Data which are required for the real time overlay preprocessed in the program initialization overlay so that they can be handled in a binary format during the real time operation. Data then are generated by the real time overlay and written to the mass storage device where they are later read and processed by the report generation overlay. By not using read and write statements, the large amount of core storage that is used by Fortran run time I/O is conserved and made available

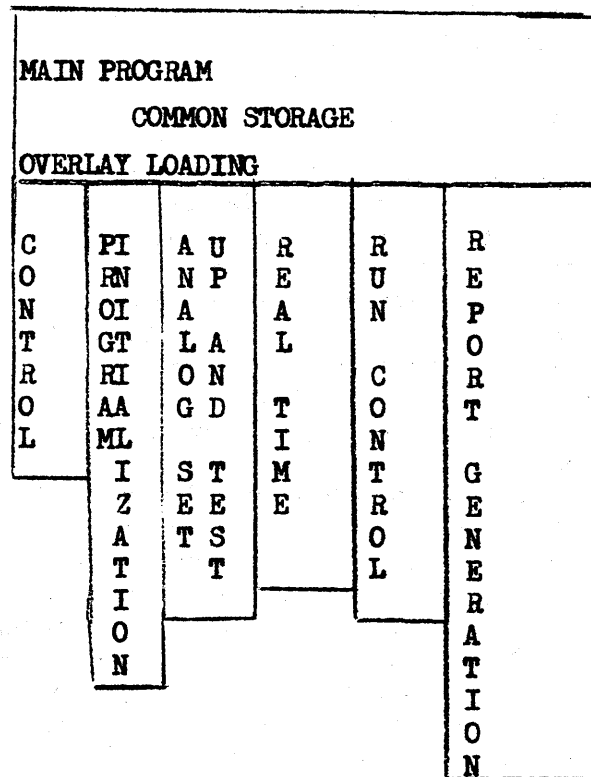


FIGURE 7—Storage allocation for main program overlays

to the real time portion of the hybrid program.

The main program contains common declarations for data that must be common to all overlays, allowing communication between them. The main program also contains calls to the overlay loader for loading the control overlay and other overlays which the control overlay may request (See Figure 8).

Programs are organized so that they use the control overlay for interrogating the remote display as to the next operation which is expected. The display task used for this interrogation is the I/O task. This task allows the programmer to communicate directly with the display as though it were a peripheral device. Thus, with Fortran read and write statements, he can read and write control information which determines what part of the program is to be entered next.

The control overlay performs a read instruction to the remote display. The remote display system has been programmed so that, when a read command is given to it, it will automatically roll the program out onto the mass storage device which makes the core storage used by the hybrid program available to the system. When the hybrid programmer has typed in the control information defining what he wishes to do next, the remote display system rolls the program back into central memory and continues execution of the control overlay which then determines from the parameters typed in what the operator has requested.

The control overlay now transfers control to the main program along with the proper parameters set for loading the overlay requested. When execution of the requested overlay is finished, control returns to the main program. The main program then reloads the control overlay and the display is read again. At this point the operator usually needs some time to analyze the results of his previous request. Since the hybrid program is rolled-out at this point, the system can immediately begin executing waiting jobs and continues to do so until the hybrid programmer is prepared to continue.

At this point it must be emphasized that the structure we are talking about is not explicitly defined in the operating system. Rather it is merely a structure for program organization which has been found to be effective in allowing maximum use of the display as a control device and which emphasizes the efficient use of core storage.

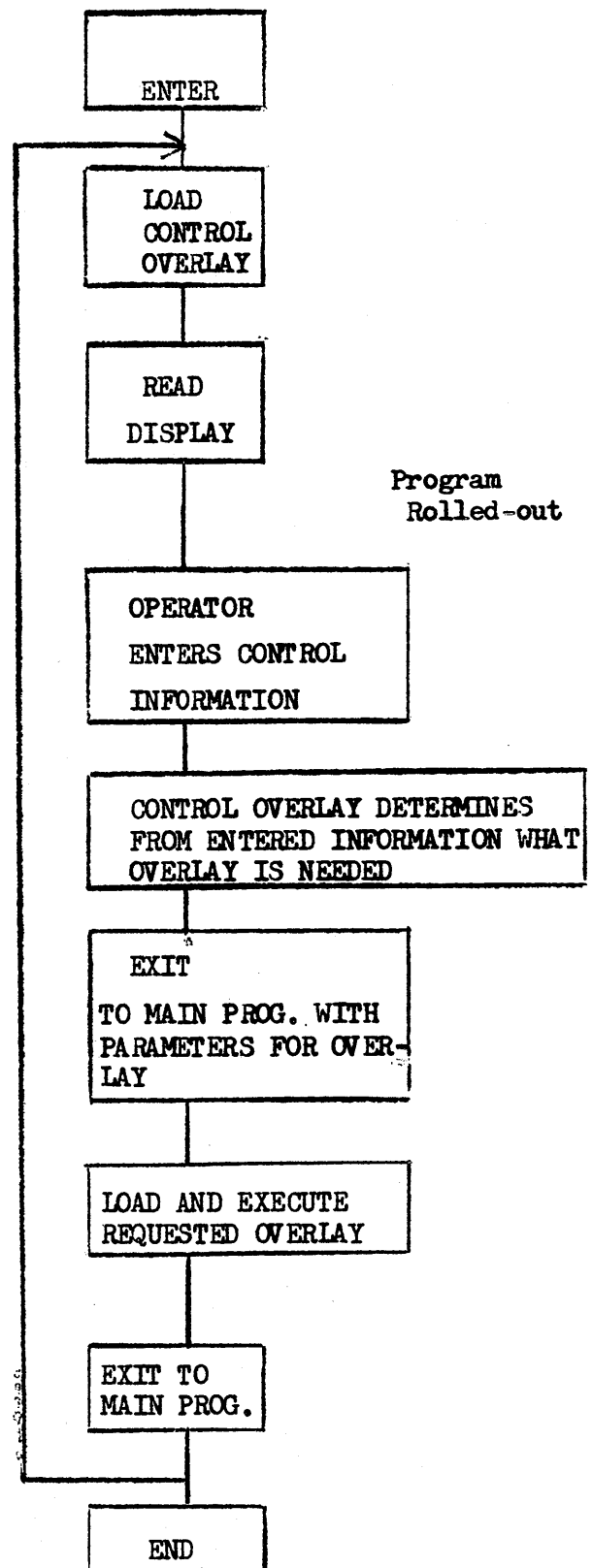


FIGURE 8—Flow diagram of overlay loading and execution

Several of the typical overlays that may be requested by the control overlay will now be briefly described. The initialization overlay is requested to initialize the program and to process all of the input data cards.

The analog set-up and test overlay is requested to set the potentiometers according to data that have been previously processed in the program initialization overlay. This overlay may initiate a number of tests to calibrate and/or dynamic test the analog mechanization.

The control overlay may request a hybrid program run or possibly a sequence of runs. If a single run is requested, the real time overlay would be loaded into central memory from the mass storage device, the simulation would be executed, and then the control overlay would be reloaded. If a sequence of runs is to be made, normally the hybrid program would contain a run control overlay which would vary parameters in the digital portion of the simulation, change parameter values on the analog computer portion of the simulation, and perform other non-real time functions between the real time phases.

The report generation overlay is requested to recover the data recorded on the mass storage device during the execution of the real time overlay. It generates listings and plots of data which has been gathered from one or more runs.

Communication with the linkage and analog equipment

In describing the LMSC hybrid computer system, we briefly discussed the communication linkage equipment (Intracom) connecting the digital computer and the analog computers (see Figure 3). Also in describing the software organization for the digital computer (see Figure 5), we briefly discussed the two pool PP program, HIO and AIO, which are used in communicating with the Intracom. In this and subsequent sections of this paper, we will describe in detail the important features of this communication both from the standpoint of the Intracom's hardware and the system software programs.

To facilitate hardware design and system software development, communication between the digital computer and the Intracom was grouped into the three categories: priority interrupt processing, Hybrid Input-Output (HIO), and Analog Input-Output (AIO).

The first category, priority interrupt processing, was desired in order that digital programs,

called priority subroutines, with varying priorities might be initiated by logic signals from sources external to the digital computer.

The second category, Hybrid Input-Output, was desired in order that values of analog or digital program variables might be input/output (I/O) at a fast rate to or from the analog computer, allowing for the possibility that the I/O defined in a predetermined pattern, might be initiated by a logic signal from sources external to the digital computer. Also the second category was desired to allow the digital computer to read or write logic signals (i.e., discrete lines) to or from the Intracom, to control a precision interval generator, and to control the mode of the Intracom.

The third category, Analog Input-Output, was desired in order that the digital computer might perform initialization, checking, and monitoring of various components within the analog computer. This category gives the same capability to the digital computer as the analog programmer has with the manual keyboard on the analog computer.

The first two categories are closely related in that sensing of logic signals occurring externally to the digital computer is necessary and also due to the desirability and/or necessity of having both I/O patterns and priority subroutines initiated by the same logic signal. Our attention in this section will be primarily focused on real time priority interrupt processing.

Priority interrupt processing

Interrupt capability

The CDC 6400 digital computer does not have a hard-wired priority interrupt capability; however, each of its PPs has the capability of commanding the CPU to exchange its registers with 16 central memory words. This exchange includes the transfer of all of the control, index, address, and arithmetic registers. Thus, by formatting in central memory exchange packages for each priority subroutine that is to be initiated by a priority interrupt signal, it is possible to rapidly switch the CPU between priority subroutines from a PP by executing an exchange jump command. This exchanging of registers requires less than three microseconds of CPU time.

By installing hardware in the Intracom which senses the occurrence of priority interrupts, determines the highest priority interrupt line active, and transmits this information upon request

by a PP to a data channel which is connected to the digital computer the ability to detect and respond to external interrupt signals can be realized by the CDC 6400.

By designing the Intracom priority interrupt hardware such that two separate but identical circuits (priority trees) may determine the highest priority interrupt active, installing a switch to transfer the interrupt signal from one circuit to the other, and connecting the output of each circuit to a separate data channel and PP, computation and input/output can be performed in parallel.

Peripheral processors associated with interrupt processing

The digital portion of the detection and response to external interrupt signals is handled by the Hybrid Monitor (HYP) and the Hybrid Input-Output (HIO) peripheral processor programs.

Hybrid monitor

It controls the operation of the CPU and also determines which priority subroutines should be initiated in response to priority interrupts detected by it. It is comprised of a PP program, HYP, which is loaded at dead start time and a CPU program which is part of the central memory resident.

When no priority subroutines are active or waiting to be processed, the PP program honors exchange requests from the system monitor to execute background jobs and maintains the job stack. Periodically the Hybrid Monitor checks to see if a request to initialize a hybrid program has been received. When such a request is received, it will read from central memory resident information necessary to set up and to execute the corresponding hybrid program.

The PP program (HYP) detects the occurrence of priority interrupts and activates the CPU when priority subroutines are to be executed. The HYP program periodically requests from the Intracom the highest priority interrupt line active and monitors the priority subroutine currently being processed by the CPU. Then based upon setup information which supplies the relative priority of the interrupt initiated priority subroutines determines whether an exchange jump to a higher priority subroutine than the one currently being executed is required. If an

exchange jump is made to a higher priority subroutine, the Hybrid Monitor will monitor the execution time of this priority subroutine to insure that it does not exceed the programs' allotted time as specified by setup data. If the execution time becomes larger than that specified, the monitor will abort that particular hybrid program.

Hybrid input-output

The HIO peripheral processor program processes analog to digital, digital to analog, and discrete information to and from the Intracom. It looks both to the central memory for requests coming from the CPU and to the Intracom's interrupt structure for commands to initiate transfer of data to and from the Intracom and/or analog computer.

The HIO program requests from the Intracom the highest priority interrupt activated and determines, based upon setup information, whether or not an I/O pattern is associated with that particular priority interrupt. If an I/O pattern is associated with that interrupt, then the communication codes of the I/O pattern are executed and the transfer of data is accomplished.

If both an I/O pattern and a priority subroutine are associated with a priority interrupt (leading edge I/O), the pattern definition is completely executed by HIO and then a function code is sent by HIO to the Intracom which transfers the interrupt to the hybrid priority tree and thus to the Hybrid Monitor.

Hybrid program organization

To obtain an understanding of how priority interrupt processing is implemented, the organization and various phases of a hybrid program will now be discussed. A hybrid program is organized with a main program, non-priority subroutines to the main program, priority subroutines, and subroutines to the priority subroutines. The typical organization for a hybrid program is shown in Figure 9.

In the following paragraphs we will present a step by step explanation of the main program structure of a hybrid program as it relates to priority interrupt processing (see Figure 10). Information deemed necessary to clarify the description will be inserted where needed.

The main program of a hybrid program initiates and controls the various program phases and calls into execution non-priority subroutines.

JOB Card
 HOOK control card
 Hook data cards
 Main program
 Non-priority subroutines to main program
 Priority subroutines
 Subroutines to the Priority Subroutines
 END

FIGURE 9—Hybrid program organization

	EXTERNAL priority subroutine name
Analog set-up phase	CALL CONSOLE CALL SETPOT CALL MODE CALL READ
Pre-real time phase	CALL SETUPFL (I/O pattern definitions) CALL STRTPAT (body of pattern) CALL WDACS CALL RADCS CALL ENDPAT CALL CONNECT CALL REALTIME CALL WAIT
Real time phase	(Priority subroutines, I/O patterns, and combination I/O pattern and priority subroutines which are interrupt initiated are in operation during real time under the control of the Hybrid Monitor and/or the HIO peripheral processor routines)
	SUBROUTINE Name (initialization code) (I/O pattern definitions) CALL FASTPRG (real time code) END
Post-real time phase	CALL RELEASE (post-real time analysis of data obtained during real time) END

FIGURE 10—Main program structure

The main program and its non-priority subroutines execute the analog set-up, pre-real time, and post-real time phases of a hybrid program.

SETUPFL

In general one of the first executable statements called in the main program is SETUPFL. This system subroutine performs the initialization needed to enable a hybrid program to communicate through the HIO peripheral processor program via a data channel to the Intracom. The HIO circular buffer is initialized to enable CPU-PP communication. Also the Intracom priority interrupt subsystem is initialized such that all

interrupts are cleared and disabled and such that all interrupt masks are cleared.

I/O patterns

The main program or the initialization code of a priority subroutine generates the I/O patterns during the pre-real time phase and they are executed when an associated priority interrupt signal occurs during the real time phase or when called from the main program, a priority subroutine, a non-priority subroutine, or by another I/O pattern. An I/O pattern is structured as shown in Figure 10. The pattern defines a sequence of computer instructions which perform input and output to the Intracom and/or the analog computer. A CALL STRTPAT statement must be the first statement of the pattern definition. This statement specifies a number (from 1 to 63) by which the pattern can be referenced in the hybrid program, and it informs the system that what is to follow is to be associated with this specified identification number. The CALL ENDPAT statement indicates to the system that the pattern definition is to be terminated. The body of the pattern definition consists of calls to system communication subroutines such as CALL RADCS, analog-to-digital conversion, and CALL WDACS, digital-to-analog conversion. The system subroutines called in the body of the definition cause instructions to be generated for the HIO peripheral processor program. These I/O patterns are resident in and executed from the PP which contains the HIO program.

Priority subroutines

A priority subroutine is a subroutine located in central memory which is to be executed only during the real time phase of a hybrid program and is normally initiated by the occurrence of a priority interrupt signal. The priority subroutine is structured according to the special format shown in Figure 11. The initialization code is executed in the pre-real time phase. The real time code however, is executed in the real time phase due to the occurrence of a priority interrupt to which the priority subroutine is associated.

Priority levels

Each I/O pattern, priority subroutine, or com-

(initialization code and definition of I/O patterns)	SUBROUTINE Name CALL STRTPAT (body of pattern) CALL ENDPAT CALL WDACS . . . CALL SMSK CALL FASTPRG . . . CALL SCALIN . . . CALL SCALOUT . . . END
(real time code)	

FIGURE 11—Priority subroutine structure

bination I/O pattern and priority subroutine (leading edge I/O) is assigned a priority level on the basis of its frequency of occurrence or relative importance. A priority interrupt line is then assigned to each of these priority levels with the interrupt line with the smallest number being assigned to the highest priority level.

Hook and hook data

For each priority interrupt line that has been associated with a priority subroutine, a Hook data card which contains the interrupt line number, maximum frequency of occurrence, and the maximum CPU execution time needed to process the corresponding priority subroutine is inserted in the program deck. The Hook system program is called by a HOOK control card at the beginning of a hybrid program deck. Using the data cards the Hook program informs the Hybrid Monitor as to which interrupts are to be used to initiate associated priority subroutines, the maximum frequencies of occurrence, and the maximum CPU execution times. Also the percentage of total CPU time needed per second to process each priority subroutine is calculated from the information on the Hook data cards and placed in the output file.

CONNECT

In the pre-real time phase of a hybrid program, calls are made to the CONNECT system

program which communicates to the HYP and to the HIO peripheral processor programs what should take place in response to the occurrence of the various priority interrupt signals. A separate CALL CONNECT statement must be made for each priority interrupt which is to be used by the hybrid program. The calling sequence of the CONNECT statement specifies the Intracom to be used, the interrupt line number, the I/O pattern number, the priority subroutine name, and priority subroutine parameters.

This call statement may connect an Intracom interrupt to an I/O pattern, resident in the HIO PP, which will communicate information to the Intracom during the real time phase of the hybrid program whenever its associated priority interrupt is triggered.

It may connect a priority interrupt to a priority subroutine which will be called into execution by the Hybrid Monitor's PP program (HYP) during the real time phase whenever the connected priority interrupt occurs. The CONNECT program executes or places in an HIO buffer for execution when RLTIME is called, the various initialization codes of the priority subroutine and terminates upon encountering the CALL FASTPRG statement; then during the real time phase whenever the associated priority interrupt is detected, the priority subroutine will be entered after the CALL FASTPRG statement and the real time code will be executed. The initialization code of a priority subroutine usually contains a SMSK call statement, whose purpose is to set the mask of (i.e., enable) the associated priority interrupt. The SMSK statement is placed in the HIO buffer to be processed at the time the RLTIME call statement is executed.

The CONNECT statement may connect both an I/O pattern and a priority subroutine to a priority interrupt. In which case, the I/O pattern will be executed completely and then the priority subroutine will be executed.

RLTIME

This system subroutine is called, following the initialization portion (pre-real time phase) of a hybrid program at the point in the main program where it is desired for real time processing of priority interrupts to be initiated. This subroutine determines whether the total CPU execution time required by both hybrid programs exceeds 80 percent of the available CPU time. A pause is initiated in the new hybrid program if

the total execution time needed does exceed the available CPU time. The routine enables the Intracom's priority interrupt subsystem. The prime functions of the RLTIME call statement are activating the Hybrid Monitor so that it may start real time processing of priority interrupts and causing the HIO program to be loaded into a pool PP. HIO remains resident in the PP during real time and makes it possible to communicate with the Intracom and the analog computer.

WAIT

Following the RLTIME call statement the WAIT subroutine is called by the main program to place the main program in recall status until requested to become active again by an ACTIVE or ABORT call statement in a priority interrupt initiated priority subroutine. The CPU at this point discontinues processing the slow or non-real time portion of the hybrid program.

I/O patterns, priority subroutines, and combination I/O patterns and priority subroutines continue to operate during this time, that the main program is in recall status, whenever, initiated by their associated priority interrupts.

I/O pattern body

The body of an I/O pattern is primarily made up of the RADCS and the WDACS statements. These two call statements control the flow of analog-to-digital and digital-to-analog data and perform the actual transfer of that data to and from the digital computer.

The function of the RADCS system routine is to read sequentially addressed analog-to-digital (A/D) channels, A/D channels originate from analog trunks located on the analog computer's patchboard, by means of the analog-to-digital converter (ADC) in the Intracom, transfer the data read from the Intracom's ADC to the HIO PP, and then pass the data into a specified buffer in central memory. In the RADCS call statement the address of first A/D channel to be read, the total number of channels to be read, and the buffer into which the data is to be stored are specified. Also, one of eight sample and hold controls on the Intracom patchboard may be specified; thereby, allowing the A/D channels equipped with sample and hold amplifiers to be placed in hold prior to their being read by the ADC, providing appropriate patching has been done on the Intracom.

The WDACS subroutine outputs to sequentially addressed digital-to-analog channels (DACs) in the Intracom data which has been previously prepared by the digital program (CALL SCALOUT) or data scaled, converted, and packed by the WDACS subroutine itself. The DACs' outputs are terminated on analog trunks on the analog computer patchboard. The WDACS call statement specifies the buffer from which data is to output, the address of the first DAC to which data are to be transferred, and the total number of sequentially addressed DACs to which data are to be output. If one of the eight transfer controls, located on the Intracom patchboard, is specified then, all data to be output are transferred to the DACs sequentially but are converted simultaneously to analog voltages. If the data to be transferred have not been previously prepared, a table of scales (conversion factors to convert from physical units to analog voltages) must be specified in the WDACS call statement.

The body of the I/O pattern may also contain calls to system subroutines which read or write logic signals to or from individual or groups of discrete lines which originate or terminate on the Intracom patchboard, change the mode of the Intracom, control the precision interval generator, set or reset interrupt masks, and etc.

Real time code of a priority subroutine

Normally in the real time code a call to the SCALIN system subroutine is made to unpack, convert, and scale the data that have been placed in central memory by the HIO PP as a result of a RADCS call in an I/O pattern. Computations based upon this data are then made in the priority subroutine using any legitimate Fortran statements and/or the data are stored on the disk for later analysis during the post-real time phase of the hybrid program. Data which have been generated by the priority subroutine are prepared for output to the analog computer by a call to the SCALOUT subroutine which scales, converts, and packs the generated data. Normally this data will then be output to the analog computer by a WDACS call statement in an I/O pattern.

Leading edge I/O

At this point the significance of having both an I/O pattern and a priority subroutine associated with a priority interrupt should be re-emphasized. Since the I/O pattern execution is

accomplished by the HIO PP, it is apparent that parallel computation in the CPU may be taking place while the input or output of data is being handled in the PP.

If, for example, two interrupts which have associated leading edge I/O occur simultaneously, HIO performs the processing of data associated with the highest priority interrupt and then, by switching the priority interrupt to a hybrid priority tree, causes the Hybrid Monitor to switch the CPU to the priority subroutine associated with this priority interrupt. HIO now processes I/O data associated with the other priority interrupt while the CPU is executing the priority subroutine associated with the highest priority interrupt. Then upon completion of this priority subroutine, the other priority subroutine will begin execution. Thus, parallel input/output and CPU computation is accomplished. This becomes very important when it is realized that if two hybrid jobs are running in parallel, then CPU computation and two I/O patterns may be proceeding in parallel.

Hybrid utility display

Information about the priority interrupts being used in a hybrid program may be displayed on a display scope by using the Hybrid Utility Task of the display operating system. Each interrupt's priority, HOOK duty cycle count, elapsed duty cycle count, and the number of times that the associated priority subroutine has been in execution when a higher priority interrupt occurred is displayed. This display allows the progress of the real time phase of a hybrid program to be monitored.

If a duty cycle of zero is placed on a HOOK data card, a maximum duty cycle will be assumed and only one hybrid program of this nature is allowed on the computer at a time. This feature assures that the hybrid job will not be thrown off due to duty cycle failures and it allows a method whereby the duty cycle of a priority subroutine can be evaluated by monitoring the actual duty cycle time on the display. Once the duty cycle has been determined, it may be entered via the display keyboard under Task H to modify the appropriate HOOK data card for subsequent runs.

RELEASE

When it is desired to terminate the real time processing of priority interrupts the RELEASE statement is called. This subroutine informs the

Hybrid Monitor to go inactive and causes the HIO PP to be dropped, thus ending communication with the Intracom and/or analog computer.

Priority interrupt hardware

The priority interrupt subsystem of the linkage equipment (Intracom) is comprised of an interrupt register, mask register, switching register, enable/disable flip-flop, two steering gates, and two interrupt priority trees. Each register consists of 24 flip-flops; thus, 24 individual and unique priority interrupts are provided for hybrid program usage. A simplified block diagram of the priority interrupt subsystem is shown in Figure 12.

A basic priority interrupt line consists of a mask, a switch, and an interrupt flip-flop. The interrupt flip-flop is set by a logic signal applied to the proper Intracom patchboard hole or by an INTRPT call statement from the HIO PP. Note that the interrupt flip-flop cannot be set unless the Enable/Disable flip-flop has been enabled by the RLTIME call statement from the HIO PP.

If both an I/O pattern and a priority subroutine are to be initiated by the interrupt, neither the I/O Interrupt Inhibit or the Hybrid Interrupt Inhibit are inhibited by logic signals on the Intracom patchboard. In this case, the switch flip-flop would be reset, having been reset if set by the changing of states of the Enable/disable flip-flop which occurred in response to the call RLTIME statement. Then assuming that the Mask flip-flop has been set by a SMSK call statement from the HIO PP, the I/O steering gate is enabled and passes the interrupt signal to the I/O priority tree where it is coded.

The priority tree generates a coded number, which is read by the HIO PP, containing the highest priority interrupt currently active in the Intracom. The HIO PP executes the I/O pattern associated with that priority interrupt and sends a function code to the Intracom which sets the switch flip-flop. The setting of the switch flip-flop disables the I/O steering gate and enables the Hybrid steering gate which allows the priority interrupt signal to be passed to the Hybrid priority tree where it is coded. The HYP PP reads this coded number and if no higher priority interrupt associated priority subroutine is being executed, the HYP PP initiates an exchange jump to the priority subroutine associated with the active priority interrupt. Upon completion of the exchange jump to the priority subroutine,

the HYP PP sends a function code to the Intracom which resets the switch and interrupt flip-flops.

If only an I/O pattern is to be executed, the Hybrid Interrupt Inhibit is inhibited by a logic ground applied at the patchboard. The priority interrupt is processed as previously discussed except that upon completion of the pattern execution the interrupt flip-flop is reset by a function code from the HIO PP and nothing is done to the switch flip-flop.

To execute only a priority subroutine, the I/O Interrupt Inhibit is inhibited which sets the switch flip-flop and maintains it in a set state. The priority interrupt is processed as previously described.

Non-real time hybrid input-output

A number of system subroutines are provided for communicating with the various sub-systems of the linkage equipment. These subroutines use

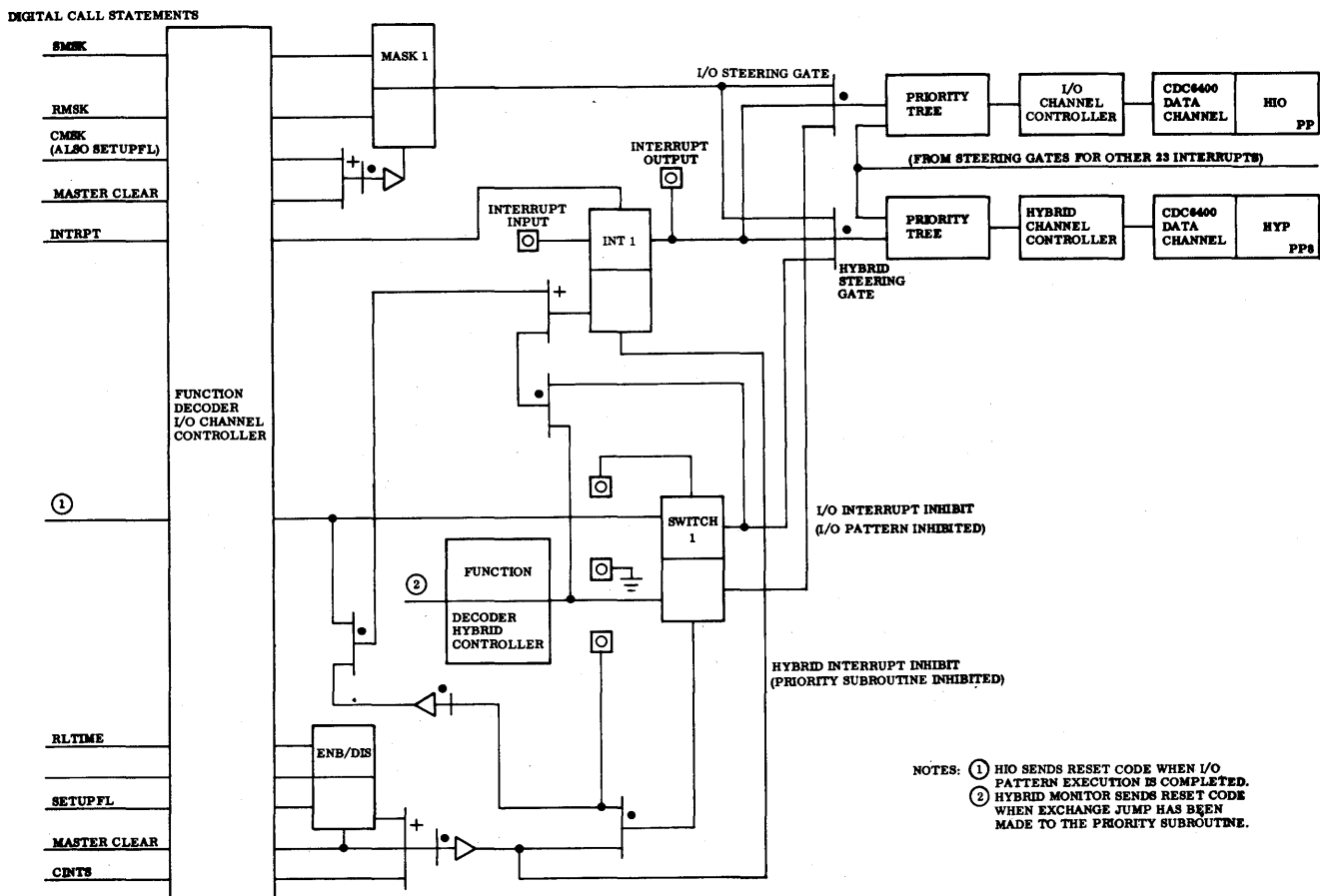
the HIO PP to perform the actual communication with the linkage equipment. They may be used either in the real time phase or in the non-real time phases of a hybrid program. Several examples of their use in the real time phase have been discussed previously.

To be used in one of the non-real time phases of a hybrid program a call is made to the LOADHIO subroutine which loads the HIO program into an available pool PP (see Figure 13). Once HIO has been loaded, then any of the subroutines listed in Figure 13 may be called. The DROPHIO call statement is used to inform the system that no further communication to the Intracom is to occur and HIO is then dropped from the pool PP.

Analog input-output

The digital computer can perform initialization, checking, and monitoring of various analog computer components by using AIO. The system

FIGURE 12—Intracom interrupt subsystem



```

SETUPFL
.
.
.
LOADHIO
.
.
.
(Intracom I/O Subroutines)

GINTS (clears interrupts)  RPIG (read PIG)
CMSK (clears masks)       SCALOUT (scale data for output)
INTRPT (set interrupt)    SMSK (set mask)
PATIO (execute I/O pattern) STORE (place sample/hold in hold)
PIG (change mode of PIG)  TRACK (place S/H in sample)
RADCS (read A/D channels) TRANSFER (DAC control)
RDISCR (read discrete line) WDACS (output to DACs)
RDW (read parallel word)  WDISCR (write to discrete)
RMODE (read Intracom mode) WDW (write to parallel word)
RMSK (read masks)        WMODE (set Intracom mode)
                          WPIG (initialize PIG)
.
.
.
DROPHIO

```

FIGURE 13—Non-real time HIO call order

subroutine, CONSOLE, is called to specify the analog computer with which communication is desired. The MODE, READ, and SETPOT subroutines are used during the analog setup phase of the hybrid program to change the mode of the analog computer, to read analog components, and to set the potentiometers. These analog computer setup commands are processed much like those of HIO. A pool PP program, AIO is called up to process the requests that are placed in the central memory circular buffer, and when the requests have been processed, the PP is returned back to the operating system.

Since the data channel, connected to the analog controller of an Intracom is only reserved during the time that an actual communication is being made, two AIO PPs may be operating via the Intracom's single analog controller to different analog computers. Communication can be performed simultaneously to the analog computer and to the Intracom because separate PPs are used to transmit information to the analog controller and to the I/O controller of the linkage equipment. This feature permits a hybrid problem and an analog problem to be run using the same Intracom and at the same time without interfering with each other.

Automatic problem verification

Static checking of the analog computer portions of a hybrid problem can be performed using the Automatic Problem Verification (APV) digital program which resides in the central library stored on the mass storage device. APV may also be used as a request package to operate the analog computer if desired.

APV uses source program statements, similar in syntax to Fortran, to process commands to select an analog computer or an Intracom, to set an analog computer or Intracom mode, to set a potentiometer, to set the precision interval generator, to set a digital to analog converter, to read analog components, and to test readings from an analog computer against the evaluation of an arithmetic expression. Each source statement is processed immediately after it is read from either the remote display or the input file. If an error is detected, the nature of the error detected is output to either the remote display, or to the output file.

Testing is performed by statements of the form: system variable (a string of characters representing the value of a particular analog component) = expression (string of constants, system variables, etc., separated by arithmetic operators). When such a statement is encountered, the analog component specified by the system variable is read, the expression is evaluated with other system variables being read if necessary, and if the two values obtained lie within a specified tolerance, the test passes. Tolerances can be specified by the programmer or standard built-in tolerances may be used. If the test fails, the value read for the system variable being checked, the value of the evaluated expression, and the difference between these two values are output to the display and to the output file.

Step-by-step static checking of the analog computer under the programmer's control is accomplished by allowing the APV program to be controlled from the remote display.

SUMMARY

This paper has speculated on the blue sky hybrid systems of the future, and discussed the general features and some of the details of implementation of the Lockheed Missiles and Space Company (LMSC) hybrid computer system. Most of the features described were implemented by modifying and augmenting a digital batch processing system.

At the time of the writing of this paper there have been at least six implementations of hybrid or real time systems using CDC 6000 series digital computers. All of these systems are different. Each system has been implemented independently and the emphasis guiding the development of each system has been focused on completely different areas. The desire to use the system as a batch processor has tended in some systems to override the hybrid requirements.

The background of people responsible for the software development has also varied. Those with predominately analog background have been satisfied with simple system design which requires the hybrid programmer/operator to use complicated methods of program organization and control. Those with batch processing experience have compromised the hybrid requirements and have been more concerned with protecting the system from the hybrid programmer/operator rather than giving him more capability and assistance.

Interface hardware varies in each of these systems from a single level interrupt source to a Hardware Hybrid Monitor.

Most of the systems which have been delivered are now maintained by the customer. The independent development of these systems continually widens the gap between the compatibility of the various systems. This trend has been aggravated by the tendencies of people to defend their own implementations to the point of ignoring other system approaches that would satisfy their particular hybrid requirements.

The reason for all of this disunity is the lack of experience and the resulting definition of the requirements of an efficient hybrid computing system. Hybrid computation is at the point that batch processing was when each installation developed their own operating systems. What hybrid system software needs is a definition of what a hybrid system is and guidelines to implement it.

The definition of a hybrid system would set requirements on hybrid equipment vendors to furnish software to meet a certain standard. They would then be required to develop and maintain a

system of software that would meet the requirements of most customers. This definition of a hybrid package would also allow the prospective customer to make meaningful evaluation of a hybrid system regardless of his experience.

Guidelines should emphasize characteristics of a hybrid system that will handle most of the problems that a general hybrid facility will face. In general the implementation of a hybrid system has been built around the system's initial requirements. After this has been done the capability of the system is set and future requirements must fall within that capability. Systems, however, should be designed to exploit the capability of the hardware as the only limiting factor.

If we review our crystal ball gazing and our current implementation, we can say we have come a long way towards our ultimate goal. We have established communication to the hybrid programmer/operator at a location remote to the digital computer and close to the operations center of the hybrid program. We have allowed him complete system flexibility giving him extensive control and monitoring capabilities. We have harnessed a large CPU and taken advantage of a multi-processor organization. The system is effective as a hybrid computer allowing two hybrid jobs to run simultaneously and is still capable of processing background digital work when possible. Our interactive Automatic Problem Verification Program (APV) is an effective step towards the software package described.

The "blue sky" set of software described at the beginning of the paper is beyond the resources and capability of a single existing installation, and can only come about by utilizing all the resources of vendors, universities, and users in designing and implementing comprehensive and efficient hybrid software systems.

We at LMSC feel that we have made a giant step forward. The rest of the way is clearly possible with the hardware available. The last step will come about when software resources are better marshalled to pursue the common objective, "the ultimate hybrid system."

A priority interrupt oriented hybrid executive

by GERALD N. SOMA and JOSEPH D. CRUNKLETON

Astrodata, Incorporated
Anaheim, California

and

ROBERT E. LORD

Montana State University
Bozeman, Montana

Hybrid simulation requirements

Prior to discussing the design and implementation of an executive system for hybrid simulation, it is desirable to discuss the particular requirements which hybrid simulation places upon the digital computer and its operating software. Of course, the designers' opinions of how the solution of a hybrid problem can best be achieved have a great effect on the design goals and subsequently the implementation of the system.

Priority interrupt structure

There has been substantial discussion in the past about the necessity of a true priority interrupt structure for hybrid simulation. It is not the objective of this paper to establish that a true priority interrupt structure is essential. Suffice it to say that the design of this hybrid executive, henceforth referred to as RAD-5075, is based upon the premise that for the class of digital computer considered a hardware priority interrupt structure, and routines in the executive system to support the hardware, are desirable for the solution of hybrid problems. Based on this premise, a major design goal for RAD-5075 was to provide a convenient means for the user to link high priority tasks in his problem to external priority interrupts in the digital subsystem of the hybrid system. The ability to have dynamically programmed changes in the relative priority of tasks is also of importance. However, providing the user with a flexible and convenient means of dealing with the priority interrupts is

not sufficient. The response of the digital subsystem to asynchronous interrupts from other subsystems of the hybrid system must occur within a time period which is not significant relative to the frequencies of problem variables or events. The priority interrupt structure must assume the responsibility of preserving and restoring the environment of the interrupted program. This achieves the desirable feature of multiple users not having to concern themselves with the operations of each other.

Re-entrancy

The concept of a priority interrupt structure in a hybrid system is very closely related to that of re-entrancy. In an environment where interrupts can occur asynchronously, it will often happen that the routine which has been interrupted will be used by the interrupting program. This can occur with math or library functions or with input/output operations. For cases such as this, it is not sufficient that the interrupt structure (hardware and software combined) preserve only the registers and central processor status of the interrupted program. It is also necessary that the intermediate data created by the program and stored in a memory location be saved by a technique such as "stacking" where a last-in, first-out algorithm applies to operations to and from the reserved storage area for a particular data value. The ease of achieving re-entrancy of programs is a function of the instruction repertoire and the hardware of the respective digital computer.

Input/Output

Input/output requirements for hybrid simulation are specialized. It is necessary to provide the capability for the digital computer to communicate with devices which are quite different from the standard digital computer peripherals. Differences exist in the formatting of the data, in the rate of transfer, and in the type and extent of error checking which is done. The goal here was to make the use of the specialized input/output operations as convenient as communication with the standard peripherals, but to incorporate the differences in philosophy which are dictated by real-time hybrid simulation requirements. An example will be used to illustrate the latter point. If in a standard digital computer magnetic tape operation, an error condition is encountered, the operation will be retried a set number of times before reporting the error condition and taking an action such as aborting the job. For hybrid input/output it is desirable to have predictable time delays associated with the overhead, and the time critical nature of the communication normally precludes the use of time consuming error recovery routines. Thus, for hybrid transmissions for the majority of cases, all that the software can do is report error conditions which have been detected.

The goals of speed and convenience, for hybrid input/output present conflicting criteria. Whereas the writing of stand-alone routines will in almost every case result in the most efficient code for input/output operations, the supervising of input/output by an executive system removes a substantial programming burden from the general user. Thus, a hybrid executive intended for general purpose problem solving capability requires a rather unique compromise between the two conflicting criteria.

Multiprogramming

The nature of the hybrid environment and the cost and capabilities of third generation digital computers dictated the next property which was one of the designers' goals, that of multiprogramming. It is characteristic of hybrid operation that a relatively large percentage of time is spent in the "hold" or "idle" state which generally results in very inefficient utilization of the digital computer's central processor. An executive system that will support more than one core resident job will certainly make more economical use of the digital subsystem in a hybrid environment.

There are many considerations and complexities which enter into the design of a multiprogramming system. The multiple users must be protected from one another and be able to execute completely independent of one another. The allocation of input/output devices between or among jobs must be taken care of by the executive. Also, job scheduling and job accounting tasks are taken care of by the multiprogramming executive.

Interactive capability

Another goal in the design of this hybrid programming system was to provide the "hands-on" quality which has always been available to the user of analog computers. This facility is provided by an interactive capability allowing the user to stop his problem at run-time and monitor or modify analog or digital problem variables. Together with the capability of restarting his problem, this gives the hybrid programmer the same level and flexibility of communication which the analog computer user has traditionally had. Also included within the scope of this feature are such digital computer debug capabilities as snapshot dumps and inserted "patches." A closely related capability is the establishing of a source file on a device, such as a disc, which can be updated at the source level from a device such as a remote CRT display. Having updated the source file, the user can recompile without resubmitting his job.

As has already been pointed out in the discussions of interrupts, re-entrancy, and input/output operations, hardware has repercussions on the amount and appearance of software necessary for a hybrid executive system. This point will be discussed further in the implementation section at which time some discussion of the hardware in the interface subsystem will also be given.

The digital computer

The considerations in this section concern hardware, however, as has already been pointed out, the characteristics of the digital computer hardware have major repercussions on the nature or amount of software required for hybrid operation.

In a hybrid environment, the digital computer not only must have the capability of responding to demands of other components of the system in a very short period of time, but it also must be capable of keeping up with the other components in terms of computational speed. The size of the

Sigma 5/7 instruction repertoire, the capability of directly addressing any memory location with a memory referencing instruction, fast floating point arithmetic hardware, immediate operand accessing instructions, and a floating index capability which allows convenient manipulation of other than word size quantities are some of the features which make it possible for the Sigma 5/7 to meet the above two requirements.

Priority interrupt structure

The priority interrupt structure of the Sigma 5/7's is very powerful and flexible. There may be up to a maximum of 224 external interrupts each identified by its own dedicated memory location. There are two levels of control of the interrupt system. The accepting of interrupts by any one of the three groups (counter, input/output, or external) can be inhibited by setting one of three bits in the program status doubleword (PSD) in the CPU. The external interrupts can be individually armed, disarmed, enabled, or disabled making it possible to have programmed dynamic reassignments of priorities.

Re-entrancy

Operation in an asynchronous interrupt environment requires re-entrancy as one of the properties of the executive system and all library and input/output routines. In order to achieve re-entrancy, it is necessary to preserve the machine environment upon entering a routine and restore it upon exiting. The Sigma 5/7's have two features which greatly facilitate the implementing of re-entrancy.

The first feature is the possibility of having multiple blocks of the 16 general-purpose registers. When an interrupt occurs, the exchange of program status doublewords (PSD) can also effect a change to a different general register block for the interrupt program. This automatically saves the 16 registers for the "old" program and they will be restored when the interrupt program has been completed and the "old" PSD is restored.

The second feature is the presence within the instruction repertoire, of push-down stack instructions. These provide the capability, by the execution of one instruction, of saving all 16 registers in a reserved stack area that is maintained by the system or established and maintained by the user. These instructions also allow the saving of the contents of memory locations, other than general

registers which have been set to some intermediate value by a routine.

Input/Output

The Sigma 5/7's have an input/output channel that provides a full word parallel capability. It is possible to read a 32 bit value from an external device into a general register by the execution of only one instruction (a Read Direct) or write a 32 bit value from a general register to an external device by the execution of a Write Direct. This capability has the effect of making other elements of a hybrid computing system appear to be simply extensions of the digital subsystem.

The Sigma 5/7's also have an automatic, buffered sequential character transfer channel (Input/Output Processor), and it has its uses in a hybrid system.

The basic digital computer operating system

Description

The SDS developed Real-Time Batch Monitor (RBM) was chosen as the basis for the hybrid executive. Principal reasons for this choice were:

1. Disc orientation of the system.
2. Small residency requirement of the system (<4,000 words).
3. Job stream processing capability.
4. Fast servicing of user requests.

RBM will be very briefly described to provide the reader with some knowledge of the foundation of RAD-5075.

The system processors, copies of the resident Monitor and the Control Card Interpreter, and the library, reside in two files (M:PROC and M:LI) on the RAD (Rapid Access Data File). These two files can be accessed only by the operating system and it is possible to execute a dead-start boot of the system from the RAD at any time. From the end of M:LI to the physical end of the RAD, the user can establish his own files for use at execution time. However, at compile or assembly time the RAD, starting at the end of M:LI, is used as the binary output (BO) device if no other explicit device assignment has been made. If binary input (BI) is assigned to the RAD at the time that a load command is encountered, RBM assumes a compile-and-go situation in which compilation was the last operation which referenced BO. Figure 1, RBM Disc File

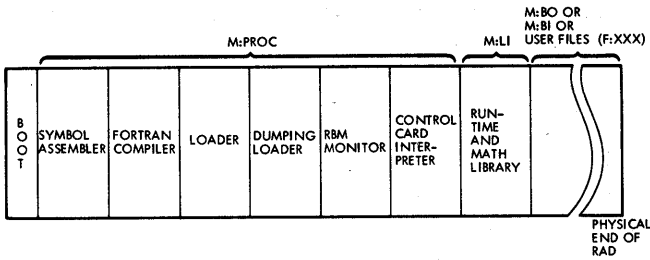


FIGURE 1—RBM disc file organization

Organization, indicates the RBM organization of RAD files.

The individual processors are in absolute on the RAD, and when the Control Card Interpreter (CCI) encounters a control card requesting one of them, the processor is loaded at some predetermined system bias.

Figure 2, Memory Map of RBM, illustrates a memory map of RBM during the time that the system is in idle awaiting a job.

The Control Card Interpreter is transient in that it is brought in from the RAD by the Monitor between execution phases of a job. Some of the job control cards which CCI recognizes under RBM and brief descriptions of the resulting operations are shown in Appendix A.

If the FIN control card is included at the end of a job, the operator must use the control panel interrupt to initiate an unsolicited key-in which

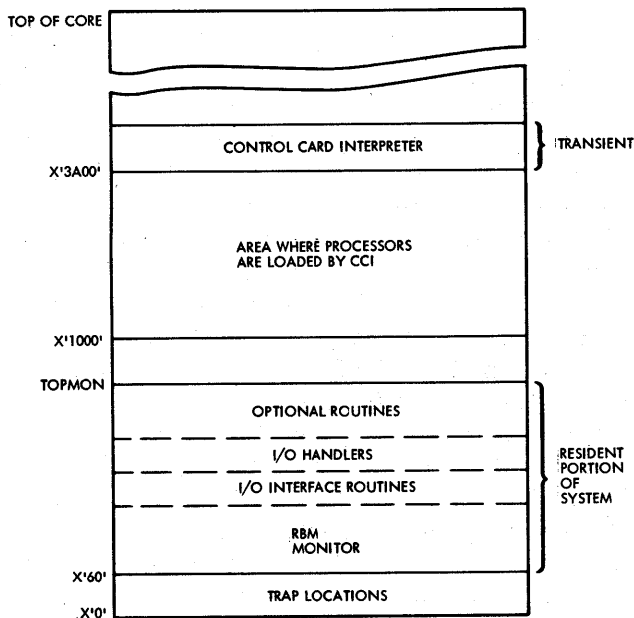


FIGURE 2—Memory map of RBM

will remove the system from the idle state and cause CCI to go to the card reader (control command input device) for the next control command. However, if no FIN cards are used in a job stack in the card reader, every time a JOB card is encountered the previous job will be terminated and signed off and the subsequent job will be read into memory. Thus, RBM does provide a job stream processing capability.

The disc file management capability provided by RBM is minimal. Under the standard system the user can utilize RAD space outside of M:PROC and M:LI for scratch storage only. There is no provision for protecting his file from subsequent jobs.

The Symbol Assembler which operates under RBM is a one-pass assembler; the compiler is SDS's FORTRAN IV-H which is also a one-pass processor, and the loader does not provide the capability of linking or overlaying. Since the assembler is only one-pass, the loader must define all forward references and literals and must satisfy all external references.

The FORTRAN Run-Time Library, the Math Library, and all of the input/output under RBM are re-entrant.

The fact that the core residency requirement of RBM is small and that it provides relatively straightforward and fast services to the user make it attractive as a basis for a hybrid executive.

Deficiencies

The following paragraphs point out some deficiencies of RBM related to its operation in a hybrid environment and the next section will describe what was designed and implemented to eliminate the deficiencies.

1. In a real time and hybrid environment the transmitting, receiving and processing of data is often dependent on the occurrence of an external event. Quite often the receive, process, and transmit loop is time critical. Therefore, a convenient method must be provided to allow the user to service external interrupts and connect them to portions of his program. The overhead incurred at the time of occurrence of the external event must be minimized. Under RBM no software exists for handling external interrupts.
2. The general software provided under RBM, although easily adaptable to a hybrid system, does not provide for specialized com-

munication with analog computers, ADC/DAC subsystems, display scopes, or various other hybrid display devices. It is essential that the software that handles these external devices be totally integrated into the operating system for user convenience and that it be re-entrant.

3. The normal mode of operation of RBM is for the user program to operate in the background area and in the slave mode. For clarification, the slave and master mode refer to the operating program's capability to execute certain privileged instructions (all I/O instructions can only be performed by a master mode program). The background area refers to the portion of memory that is outside the area required by the resident Monitor. The Monitor area can only be written into by programs that have the correct write key in their program status doubleword. Under RBM, the Monitor operates in the foreground area (master mode) and there is no provision for having multiple slave mode user jobs resident concurrently. That is, there is not a user oriented multiprogramming capability.
4. In considering a hybrid operating system, a close man-machine interface is needed not only with the analog console but with the total system. An interactive capability must exist that allows easy communication between all major subsystems of the complex and the user program. The implementation of this capability required modification of several of the standard RBM processors.

Implementation of a hybrid operating system

The additions and modifications made to RBM to produce the hybrid operating system RAD-5075 will be described in the following order:

- 1.) Priority interrupt processor
- 2.) Hybrid input/output
- 3.) Multiprogramming capability
- 4.) Interactive capability
- 5.) Hybrid error processor

Priority interrupt processor

As has already been described, the Sigma 5 and 7 have excellent hardware priority interrupt structures. However, the routines in the executive which are necessary to support the hardware do not exist in RBM and, therefore, were a neces-

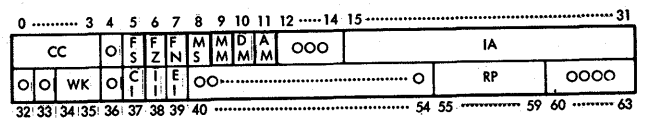
sary addition in the implementation of RAD-5075. An overhead time for entering a user's interrupt routine of less than ten microseconds was achieved.

The functions which the executive routines must perform to support the interrupt hardware are: preserve and restore the machine environment for the interrupted program, allow the user to establish a link between a given external priority interrupt and one of his programs or subprograms, and allow the user to arm, disarm, enable, disable, or trigger the interrupts individually.

Preserving and restoring of the machine environment is accomplished by utilizing additional blocks of general registers and the stack manipulation instructions which exist within the instruction repertoire.

Each priority interrupt has a dedicated memory location which contains the first instruction to be executed after the occurrence of the respective interrupt. The instruction stored into the memory locations is an exchange program status doubleword (XPSD). The program status doubleword (PSD) consists of 64 bits of information in the CPU which contain the critical control conditions of the CPU. The information contained in the PSD is illustrated in Figure 3, Sigma 5/7 PSD.

The XPSD instruction points to a four word area in memory and results in the current PSD being stored in the first two words and a new PSD being loaded from the last two words into the CPU. The instruction address portion of the new PSD, of course, points to the first instruction



- CC - CONDITION CODES
- FS - FLOATING SIGNIFICANCE CONTROL
- FZ - FLOATING ZERO CONTROL
- FN - FLOATING NORMALIZE CONTROL
- MS - MASTER/SLAVE CONTROL
- MM - MEMORY MAP CONTROL
- DM - DECIMAL MASK
- AM - ARITHMETIC MASK
- IA - INSTRUCTION ADDRESS
- WK - WRITE KEY
- CI - COUNTER INTERRUPT GROUP INHIBIT
- II - INPUT/OUTPUT INTERRUPT GROUP INHIBIT
- EI - EXTERNAL INTERRUPT GROUP INHIBIT
- RP - REGISTER POINTER

FIGURE 3—Sigma 5/7 PSD

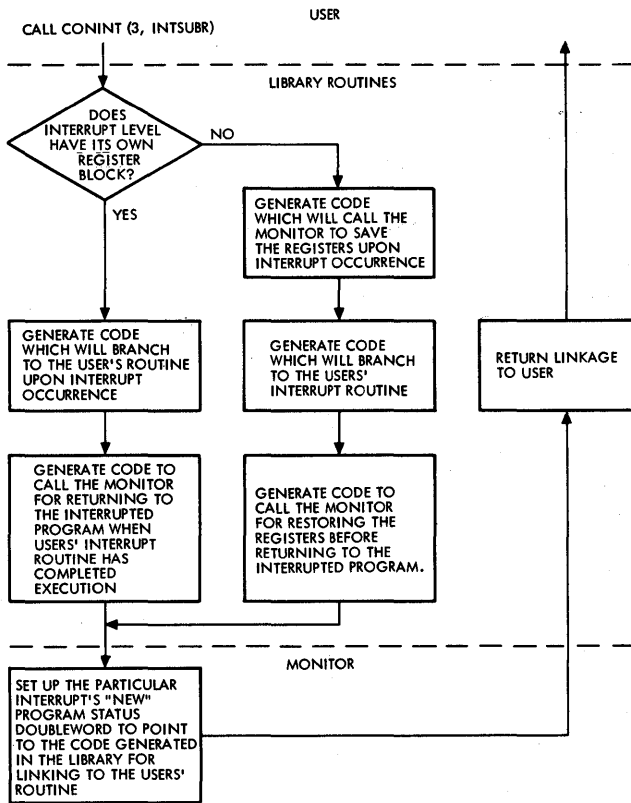


FIGURE 4—Connecting interrupts

to be executed by the CPU under the control of the new PSD. Also, if the register pointer field in the new PSD is pointing to a different block of general registers than were being used under the old PSD, the contents of the old register block will not be disturbed by the new program. Thus, with additional register blocks, the execution of the XPSD instruction at the interrupt location results in saving the old PSD and assuring that the registers used by the interrupted program are preserved.

If an external interrupt does not have a dedicated additional register block, the IA of the new PSD must point to a routine which saves the general registers before going to the user's interrupt routine. This saving can be done with a Push Multiple (PSM) instruction. All that is required to save the registers is to load a number from 1 to 16 into the condition codes and execute the PSM instruction. After the user's interrupt program has been executed, the old register values must be restored with a Pull Multiple (PLM) instruction before returning to the interrupted program. The linking of interrupts to user programs and servicing of interrupts are accomplished by the user through his calling of special library routines

which, in turn, call the routines in the executive which service the user's requests. Figure 4, Connecting Interrupts, illustrates the mechanics of connecting a given priority interrupt to a user's program.

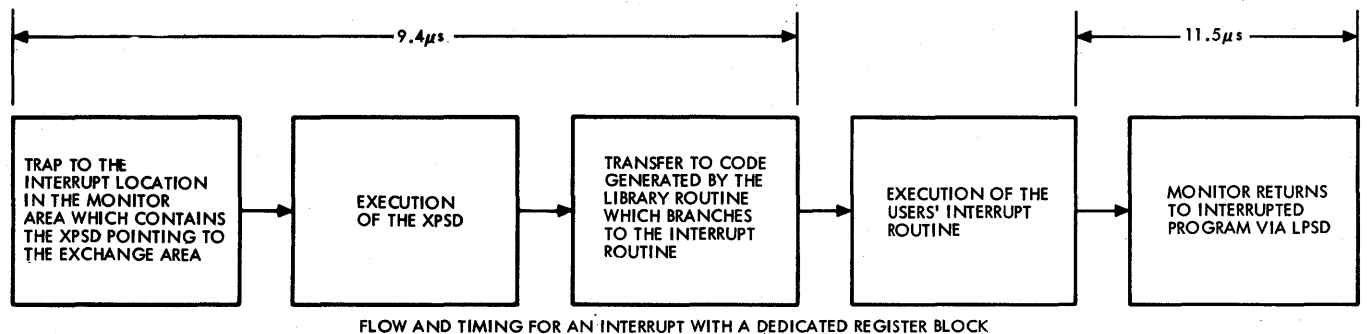
The following describes a typical sequence of events for the setting up and use of an external interrupt:

- 1.) At the time that the system is booted in from the RAD, the locations beginning with hexa-decimal address 60 and continuing through the number of external interrupts for the respective system are loaded with XPSD instructions pointing to the interrupt PSD table.
- 2.) The interrupt PSD table consists of four word areas for each external interrupt and is initialized such that if an interrupt occurs without having been connected to a user routine it is disabled, disarmed and a return to the interrupted program results.
- 3.) A user program will request that a specific routine of his be connected to a specific external priority interrupt. This results in the calling of a library routine which checks arguments and, in turn, calls the routine in the Monitor area which sets up the PSD area for the appropriate interrupt number to point to an area in the library routine. This library routine has also determined whether the interrupt, which is the subject of this user call, has its own dedicated register block.
- 4.) Next, the user will request that the interrupt be armed and enabled. This results in going to a library routine which checks the arguments and, in turn, calls the executive routine which appropriately services the specific interrupt. Note that steps 3 and 4 are most commonly executed during the non-time critical or set-up portion of a hybrid program.
- 5.) The user may then go into execution of the time critical portion of his program, a part of which is his interrupt routine, or he may allow the executive to fall back to the processing of a non-real time job in the background. In either case, the following sequence of operations results upon the occurrence of the external interrupt. Figure 5, Flow of Interrupt Processing During Time-Critical Execution, illustrates the sequence of events after the occurrence of the interrupt and presents timing data.

- a.) First the XPSD instruction in the interrupt location is executed saving the PSD of the interrupted program and loading the interrupt PSD into the CPU. The IA portion of the new PSD points to a set of instructions in the library routine which connect the interrupt to the user's program.
- b.) The code in the library routine is dependent upon whether the interrupt has its own general register block or not. If it does, the XPSD has accomplished all of the saving of the environment of the interrupted program and a branch to the user's interrupt routine is executed. If the interrupt does not have its own register block, the code in the library routine calls the Monitor to save the registers in a stack with a PSM instruction. The Monitor then returns to the user's interrupt routine.
- c.) When the user's interrupt routine has been executed it returns to the library

routine which in turn calls the appropriate routine in the monitor area for returning to the interrupted program. If the interrupt has its own register block, the return is accomplished by a Load Program Status Doubleword instruction from the interrupt PSD table. This results in restoring the PSD and changing the register block pointer. For an interrupt not having a dedicated register block, the registers for the interrupted program must be pulled from the stack before executing the LPSD and returning to the interrupted program.

- 6.) The user can at any time request the connecting of an interrupt to one of his programs or subprograms, the releasing of an interrupt from one of his programs, or the servicing of an individual interrupt.
- 7.) The exit from a job always initializes the contents of the interrupt PSD table such that no interrupts are connected to user subroutines.



FLOW AND TIMING FOR AN INTERRUPT WITH A DEDICATED REGISTER BLOCK

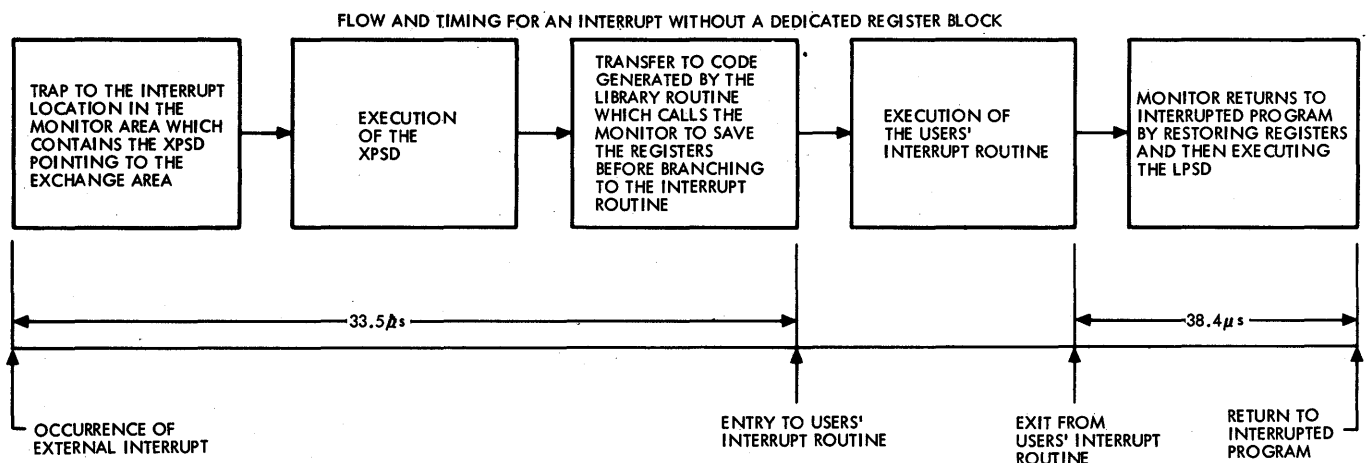


FIGURE 5—Flow of interrupt processing during time-critical execution

As shown in Figure 5, Flow of Interrupt Processing During Time-Critical Execution, for an interrupt with a dedicated register block, the time between the occurrence of the interrupt and entering the user's interrupt program is 9.4 microseconds. For an interrupt without a register block the elapsed time is 33.5 microseconds.

The total overhead of entering and returning for the first case is 20.9 microseconds and for the second case is 71.9 microseconds.

Hybrid Input/Output

The Sigma digital computers have two input/output channels both of which are utilized for hybrid communication. The Multiplexer Input/Output Processor (MIOP) is the standard buffered channel. It transfers data sequentially one byte (8 bits) at a time and can service up to 32 devices simultaneously. The MIOP requires that a command list be established in memory, that general register 0 point to the command list, and that the CPU execute a Start Input/Output (SIO) instruction in order to initiate an input or output transfer. After the CPU has issued the SIO instruction, the addressed device controls the input or output operation by making service calls to the MIOP until the transfer has been completed. Thus, having been initiated, input/output through the MIOP is independent of the CPU until an I/O interrupt occurs requiring servicing by the CPU.

RBM contains a routine called the CAL1 Processor which is a part of the resident Monitor and which supervises all of the input/output operations to the standard digital peripherals through the MIOP. This channel and a slightly modified CAL1 Processor are used for hybrid communication where sequential character types of transfer are appropriate such as controlling the analog computer. The MIOP is also used for block transfers of information from the analog-to-digital converter and to the digital-to-analog converters.

The other channel on the Sigma 5/7's is called the Direct Input/Output (DIO) channel. Through the DIO, with the execution of only one instruction, it is possible to read 32 bits of information into a general register or write 32 bits of information from a general register. The instructions are the Read Direct (RD) and Write Direct (WD), respectively.

The system as supplied by the digital manufacturer makes only internal use of the RD and WD instructions. Through the DIO channel, under

RAD-5075, it is possible to control the analog computer, read the analog-to-digital converter, write the digital-to-analog converters, read sense lines, write control lines, and initialize, read, or control the modes of the precision interval generator in the interface system. This full word, parallel channel has the effect of making the other components of a hybrid system appear as extensions to the digital subsystem.

The SIO, RD and WD instructions are what is termed "privileged instructions" in that they can only be executed by a program which is in the "master" mode. General users' programs, of course, operate in the "slave" mode whereas the Monitor operates in the "master" mode. One of the design goals as stated earlier was that of making the specialized hybrid input/output fast and convenient for the user. This was achieved during implementation by adding routines to the Monitor which will perform hybrid input/output operations in the master mode through the DIO channel. Thus, the user, a "slave" program, can request an input/output operation resulting in a library routine being loaded which checks and processes the arguments. The library routine in turn generates a call to the Monitor to execute the actual input/output instruction before returning to the user.

The input/output services are available to the user through FORTRAN calls or the generation of the standard FORTRAN calling sequence in a

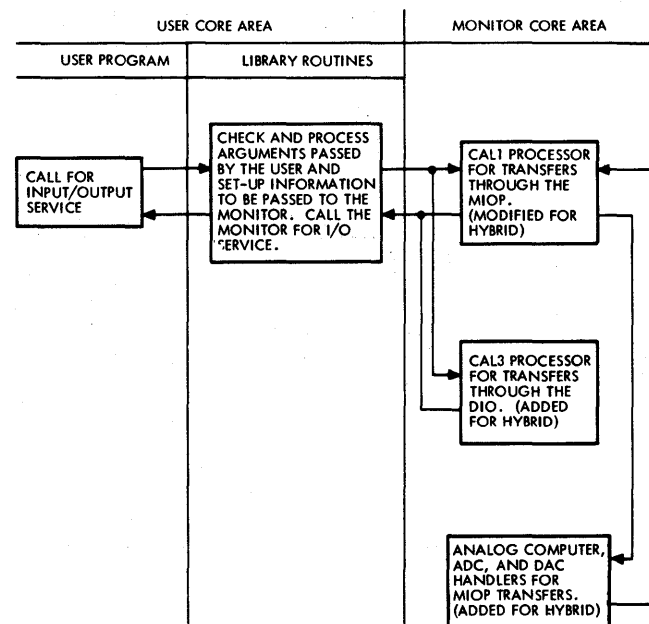


FIGURE 6—Hybrid input/output organization

Symbol program. Figure 6, Hybrid Input/Output Organization, illustrates the general flow and organization for hybrid input/output through both the MIOP and DIO channels.

Utilizing the standard calls for hybrid input/output service through the MIOP requires from 200 to 300 microseconds of overhead time after which data transfers can take place at the rate of 200,000 fifteen bit words per second. Through the DIO channel, after 80 to 100 microseconds of overhead time, data transfers from the analog-to-digital converter or to the digital-to-analog converters can take place at an average of 25 microseconds per transfer.

Table I, Channel Assignments of Hybrid Input/Output Operations, shows which hybrid input/output operations can be accomplished through the two channels. Note that the reading and writing of ADC's and DAC's through the DIO are indicated as being floating point operations. A unique feature in the interface hardware converts the 15 bit, fixed point output of the ADC to the normalized floating point short format before presenting the data to the DIO channel. There is also hardware which allows a floating point number to be written out of a general register to a

DAC; the hardware accomplishes the floating to fixed conversion. Accomplishing the conversion in hardware rather than software results in a two to one speed advantage in going analog-to-digital and a six to one advantage going digital-to-analog.

Referring back to the timing information given for the DIO channel, the presence of the hardware conversion equipment means that after 80 to 100 microseconds of overhead a normalized, floating point data value can be read into the user's memory or written from his memory at an average rate of one every 25 microseconds.

In contrast the data read or written through the MIOP channel are in fixed point format and can be stored in memory as full words or packed half-words. Also, the MIOP, once the operation is initiated, operates independently of the CPU, whereas the DIO channel requires the CPU for execution.

Patterned input/output is a unique feature of the hybrid input/output system. There are many instances when one or both of the following two cases exist in a hybrid simulation:

- 1.) The overhead associated with doing the actual I/O operations as shown in Figure 6, Hybrid Input/Output Organization, and listed above, can't be tolerated at execution time.
- 2.) The same series of input/output operations is repeated several times during execution of a hybrid program

Patterned input/output may be used in these cases. The user has special FORTRAN calls (CALL STRTPT and CALL ENDPAT) available to bracket a series of input/output operations for the purposes of defining a pattern. The calls which the user makes for the specific input/output operations that he wants in the pattern are very similar to those that he makes for a normal I/O service request. However, rather than calling the Monitor to provide the I/O service, the library routine generates the minimum amount of code necessary for the input/output operation and stores the instructions in a buffer in the user's area of core. When the ENDPAT call is encountered, all of the instructions which have been stored into the buffer in the user's area are moved into a reserved buffer in the Monitor area. The definition of the pattern and moving it into the Monitor area are done during the non-time critical portion of a job. Once the input/output pattern is in the Monitor area, it can be executed during the time critical portion of the hybrid program

MIOP	DIO
Control of the analog computer	Control of the analog computer
Read the analog computer	Read analog computer status
Read the analog-to-digital converter (fixed point, word or halfword)	Servicing of interrupts
Write the digital-to-analog converters (fixed point)	Read the analog-to-digital converter (floating point)
	Write the digital-to-analog converters (floating point)
	Read sense lines
	Write control lines
	Initialize the precision interval generator
	Read the precision interval generator
	Control the modes of the precision interval generator

TABLE I—Channel assignments of hybrid Input/Output operations

by calling it or connecting it to an interrupt.

Figure 7, Patterned I/O, illustrates patterned input/output. For this example, during the set-up portion of his hybrid job, the user defined a pattern which consisted of reading the analog-to-digital converter, writing a control line and writing a digital-to-analog converter. Library routines were loaded which processed arguments passed by the user, set-up addresses, and generated the minimum number of instructions necessary to perform the input/output operations. These instructions were temporarily stored in a buffer in the user's area of memory. Note that to read the ADC, a read direct (RD) and store word (STW) instruction were generated. Thus the data are stored into the user's address. To write the control line or the DAC load word (LW) and write direct (WD) instructions were generated. Still during set-up of the hybrid problem, the ENDPAT call will result in the generated instructions being transferred to a reserved buffer in the Monitor area where it will be executed.

During real-time operation, the input/output pattern can be called to execute through the PATTRN routine which may be in a routine connected to a priority interrupt. The overhead time in PATTRN is about 40 microseconds after which

the ADC can be read every 18 microseconds and DAC's can be written every 10 microseconds.

Multiprogramming capability

The multiprogramming or foreground/background capability provided by RAD-5075 allows two jobs to be in memory concurrently. Only one of the jobs can be real-time, that is, use priority interrupts. The real-time job is loaded into a segment of memory the size of which must be specified by the user. Background jobs are processed in a batch mode after the real time job has executed all of its pre-real time operations and has released the CPU to service possible background tasks. The occurrence of external interrupts brings the CPU back to the real time job when its services are required. The following control card is the only special consideration necessary for loading and executing a job in the foreground area.

!FJOB Proj. no., Job name, No. pages

Indicates that this job is a real time job and is to execute in the foreground area. The foreground area is in high core and consists of the number of pages (512 core locations per page) indicated. Project number and job name are optional specifications.

One library routine was added to allow the foreground job to release the CPU. This release allows the monitor to bring in the background job(s) if any are present. If release is called by a background job it returns to the Monitor for normal job termination.

The information below will be contained in the foreground's Job Information Table. It is during status 3 that concurrent foreground/background processing is possible.

Real Time Job Status

- 0—No status
- 1—Background mode (Compiler, Assembler, Loader and etc.)
- 2—Foreground mode exclusive
- 3—Foreground mode with background jobs operating
- 4—Job complete—requires sign-off.

The sequencing of the real time job status is controlled as follows :

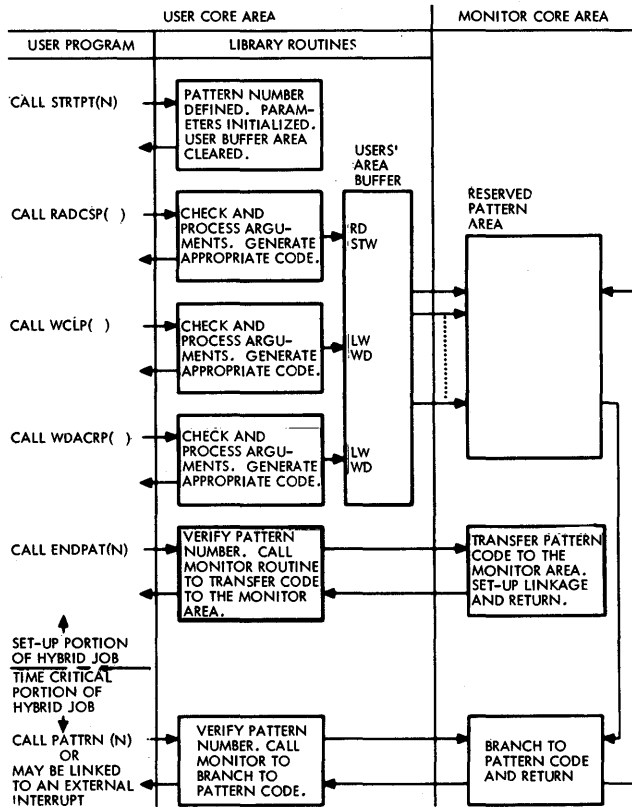


FIGURE 7—Patterned I/O

STATUS 0 to 1—Done by the Control Card Interpreter upon encountering the !FJOB card.

STATUS 1 to 2—Done by the Monitor upon a successful load operation.

STATUS 2 to 3—Done by the foreground job by execution of CALL RELEASE. This transition is not made if the core requirements of the foreground job exceed the space available between the top of the Control Card Interpreter and the top of the core.

STATUS 3 to 4—Done by the Monitor upon an exit (CAL1,9 n) from the foreground job.

STATUS n to 4—Done by the Monitor upon an error or abort exit (CAL1,9 2 or CAL1,9 3) from the foreground job.

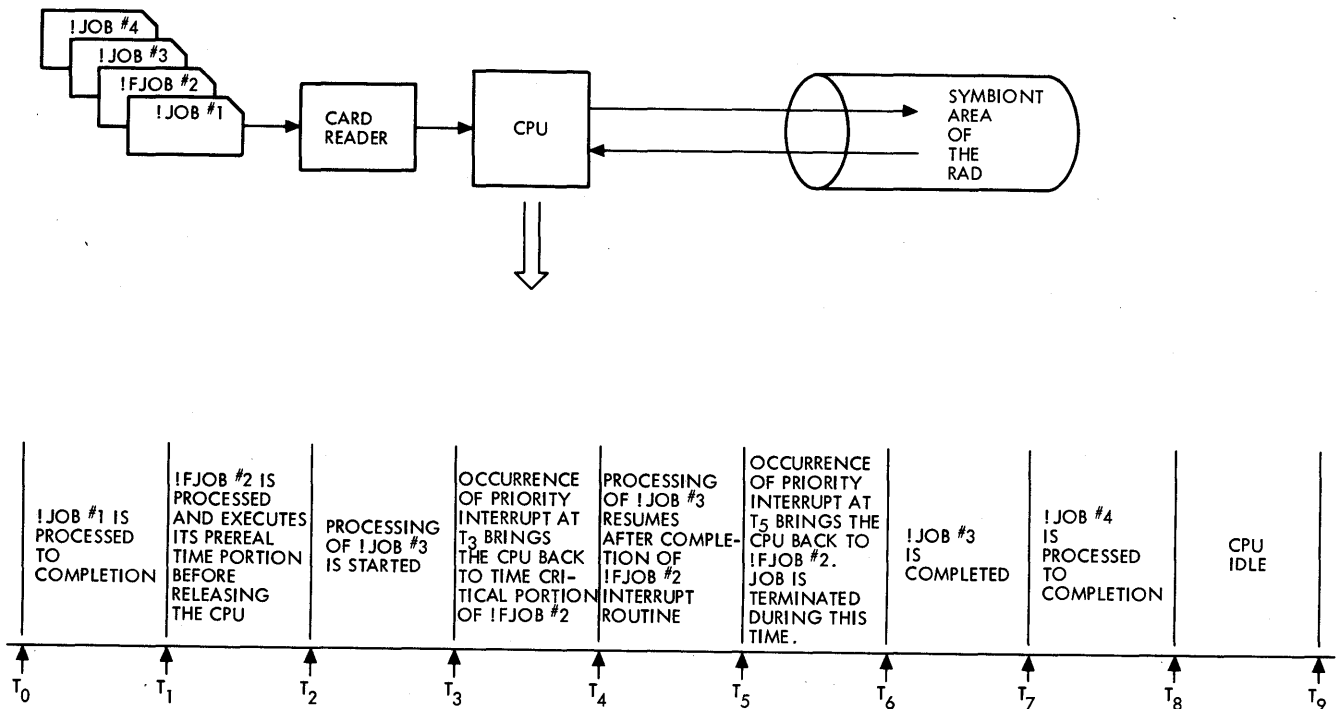
STATUS 4 to 0—Done by the Control Card Interpreter upon completion of job sign off.

The possibility of conflict between the two jobs in accessing digital peripheral equipment such as the card reader, card punch, and line printer is eliminated by the fact that RAD-5075 operates with a full symbiont capability. All input and output information passes through a reserved symbiont buffer on the disc.

To illustrate the sequence of events that occur during the foreground/background operation of RAD-5075, refer to Figure 8, CPU Time Allocation to Jobs.

!JOB #1 is a FORTRAN/Symbol job. This job is compiled, assembled, loaded into core and run to completion. !FJOB #2, the next job in sequence, is a hybrid simulation problem which utilizes priority interrupts. Upon completion of !JOB #1, !FJOB #2 starts its processing at time T_1 . Since this has been labeled as a foreground job by the job card, the compilation is performed in the background and the program is loaded into the foreground area. After being loaded, this job starts execution of its pre-real time tasks. This would include linking interrupts to user sub-routines, performing initializing functions in other subsystems of the hybrid system, and defining input/output patterns if they are required. Upon completion of execution of the pre-real time portion of the problem, the CPU may not be required immediately to support the time critical calculations of the program and can be released to the background to process the next job. At this point, the processing of !JOB #3 is initiated. While !JOB #3 is assembling, the occurrence at T_3 of an external interrupt returns the CPU to service an interrupt routine in !FJOB #2. Background processing is halted until the foreground task releases control of the CPU to the background at T_4 at which time !JOB #3 is resumed.

FIGURE 8—CPU time allocation to jobs



At T_5 , another priority interrupt occurs in !FJOB #2 which brings the CPU back to it, and during this time, the hybrid job terminates normally. After signing !FJOB #2 off, the CPU falls back to complete !JOB #3. Another background job, !JOB #4, is processed next, and upon its completion, the CPU goes to the idle state since there are no jobs requiring its service.

Complete accounting information is maintained for each job by the Monitor. The times required to compile or assemble, load, and execute plus the total time for the job are supplied to the user in his sign-off information and stored in a system file for accounting use.

Interactive capability

The man-machine communication required in hybrid simulation is provided by the Interactive Package. The Interactive Package allows the user to monitor or change problem variables at run-time in all subsystems of the hybrid system. Thus, from a control point such as a typewriter or CRT scope display, the user can read or change memory locations and component outputs in the interface equipment or on the analog computer. Memory locations can be referenced either absolutely or by symbolic name. The latter capability required modifications to several portions of RBM. The FORTRAN compiler was modified to save the program allocation information, the Loader was modified to save the load map information, and the Monitor was modified to maintain files on the disc which contain the above information.

In order to have access to the Interactive Package, the user must reference it which results in it being loaded with his program. At run-time, control can be transferred to the Interactive Package through either a user call or the occurrence of a priority interrupt which is dedicated to it.

The following lists and describes some of the commands of the Interactive Package:

1.) Read Command

- a.) READ ANALOG, COUNT—Reads a sequential number of analog or interface components starting with the designated address, (i.e., READ A/105,5).
- b.) READ MEMORY, FORMAT, COUNT—Reads the memory location or locations specified. This location can be an absolute address, a main program symbol or a symbol in a subroutine. When reading a symbolic name in a subroutine, the

subroutine and symbolic name are given. Arithmetic (+ or —) operations can also be used in specifying the address. The location is read and displayed as an integer, hexa-decimal, EBCDIC or floating point number dependent upon the specified format. An example of this command is: READ BETA+10,X,7. This would display seven locations starting at BETA+10 in a hexa-decimal format.

2.) Set Command

- a.) SET ANALOG, VALUE—Sets the specified analog or interface component to the desired value. These components can be potentiometers, digital attenuators, digital-to-analog converters, control lines or the precision interval generator.
- b.) SET MODE, X—Set the specified computational mode X.
- c.) SET, MEMORY, VALUE, COUNT—This allows several memory locations to be set to a specified value starting at the absolute or symbolic name designated.

3.) Run Command

RUN MEMORY—A memory location is specified in absolute or hexadecimal form and the program execution starts at that point.

4.) IFIX Command

This command enables the user to insert a series of values (instructions) between two memory locations. An AREA command sets up the locations where values (instructions) are to be stored and the linkage to this area is automatically made. An example of this command: IFIX ALPHA, VALUE, VALUE, . . . The location ALPHA would be set to a branch to a predefined area. The original instruction in ALPHA would be inserted in this area followed by values from the IFIX command. A branch back to the next location in the original program would be the last instruction in the new area.

5.) Snap Command

SNAP LOCATION, MEMORY, FORMAT, COUNT—This allows the user to obtain a snapshot dump during program execution. After issuing this command, a number of memory locations (COUNT) starting at the specified address (MEMORY) will be

dumped according to the specified format when the execution of the user's program reaches the specified memory location (LOCATION).

6.) Update Command

Through the use of special control cards, the user can have his program source information saved in a special file on the disc. The use of the UPDATE command in the Interactive Package allows the user to modify his source if he desires. The following operations can be performed under the Update Command.

a.) Change Command

CHANGE FROM, TO—This allows the user to change an area in his source program, as specified by the (FROM, TO) argument, to the instructions that follow this command. The original instructions are deleted and the new ones are inserted.

b.) Insert Command

INSERT LINE—This inserts a series of lines between two existing lines in the user file. The insert command would be followed by the instructions to be inserted.

c.) Delete Command

DELETE FROM, TO—This deletes the specified area in the user source file.

d.) Go Command

This returns to the Monitor with the symbolic input indicator pointing to the updated file. Thus a recompilation, load, and execute can be achieved from the updated file.

Hybrid input/output error processor

As mentioned in the Hybrid Simulation Requirements section, error checking for hybrid input/output operations differs from normal I/O error checking. The approach taken by the designers was to provide an error checking service which would be flexible for the user.

The Error Processor is a part of the resident Monitor. It is called only by the routines which were added to the library for hybrid operation. As indicated in previous discussions, the library routines do the great majority of the error checking. If one of the library routines detects an error, it calls the Error Processor passing it an error type code, the contents of the location where the error was detected, the name of the routine in which the error was detected, and the analog component address, if applicable. Having had an error condition reported to it, the Error Processor examines the states of two sense switches. Table 2 shows the possible sense switch settings and the corresponding action which will be taken by the Error Processor after being called by one of the library routines.

SENSE SWITCH STATES	ERROR PROCESSOR ACTION
SS #1 Off and SS #2 Off	Type error message and return to the user
SS #1 On	Return to the user without typing an error message with the error code stored in ERTAG
SS #2 On	Type error message and abort the job

TABLE 2—Hybrid error processor actions

The Error Processor will set a labeled common location with the error type code which allows the user to establish his own debugging environment if he so desires. For sense switch number one on, the user must do his own testing for the occurrence of errors after returning from a hybrid call and establish his own message format and conditions for printing.

Operating system summary

The organization of the total operating system

is shown in Figure 9, Operating System Organization. The Monitor is shown together with the features which it has for hybrid operation.

On the level below the Monitor, the hybrid utility programs are shown. These are individual processors that operate under the hybrid Monitor and are used in implementing, debugging, and checking hybrid simulation problems. Very briefly they have the following functions:

CSSL/COMCOR—

A digital simulation language which has

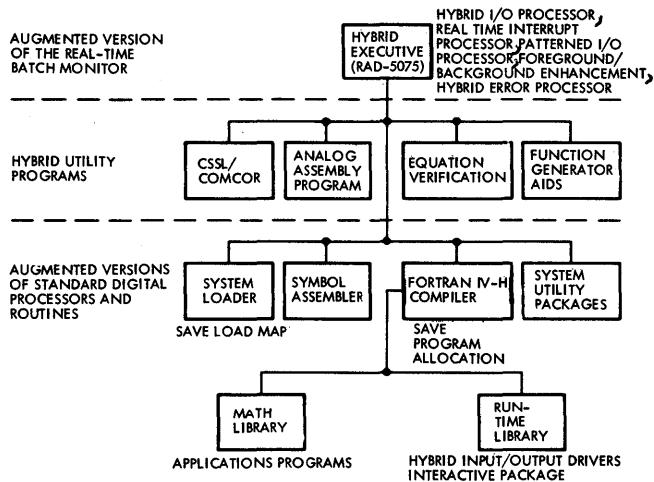


FIGURE 9—Operating system organization

been implemented according to the Simulation Council, Inc., specification of the Continuous System Simulation Language. At the present, CSSL/COMCOR is used to obtain all digital check solutions for analog or hybrid problems; ultimately it will be the hybrid programmer's real-time language.

Analog Assembly Program (AAP)—

This program achieves the system component assignments for a hybrid problem and generates patching instructions for the analog portion.

Equation Verification (EV)—

This program checks a hybrid problem both statically and dynamically back to the original unscaled differential equations.

Function Generator Aids (FGA)—

This program optimally selects breakpoints to fit a straight line segment approximation to user data. If the function generators are computer set, FGA sets them up; if they are of the card-set type, FGA punches the card.

On the next level are the augmented versions of the standard digital computer processors. The comments outside the blocks indicate what modifications or additions were made in the implementation of RAD-5075.

An example is included in APPENDIX B which illustrates what a user must do to write a hybrid program utilizing various of the features of RAD-5075.

CONCLUSIONS

The requirements of hybrid simulation on the executive or monitor of the digital subsystem of a hybrid system are such that the standard software supplied by the digital computer manufacturer is not, in general sufficient. The RBM monitor, as supplied by SDS for the Sigma 5 and 7, has many features which are desirable for the hybrid environment such as being small, fast, and disc oriented. However, it was necessary to make numerous modifications and additions in order to make it a satisfactory executive for hybrid simulation.

In implementing RAD-5075 the following design goals were achieved:

1. Small memory residency requirement — < 6,000 words.
2. Fast and flexible priority interrupt structure — < 10 microseconds to enter a user routine connected to an interrupt.
3. Flexible hybrid input/output services for the user — the Monitor supervises and executes all input/output operations for the user.
4. Full word parallel input and output of data in normalized floating point format—this makes other portions of the system appear as extensions of the digital subsystem.
5. The capability of generating highly efficient input/output patterns—after 40 microseconds of overhead time normalized floating point data can be read into memory at the rate of 18 microseconds per reading and normalized floating point data can be written from memory at the rate of 10 microseconds per output operation.
6. A user oriented multiprogramming capability—two jobs, only one of which can be real-time, can be in memory being serviced by the CPU concurrently.
7. A run-time interactive capability.
8. A flexible hybrid error checking facility.
9. Full symbiont input/output operation.

The following paragraphs list some improvements to RAD-5075 that are planned for the future.

The fact that RBM, and therefore, RAD-5075, is disc oriented is very desirable for hybrid operation. However, the disc file management capability of RBM is minimal and will be enhanced in the future for RAD-5075.

The FORTRAN IV-H compiler which runs under RAD-5075 does not generate re-entrant code. As mentioned previously, the run-time library, math library, all input/output, and the Monitor are completely re-entrant. The fact that the compiler does not generate re-entrant code requires the programmer to use redundant copies of routines that may be interrupted and used in the interrupting program. A real-time FORTRAN is planned for RAD-5075 in the future.

The RAD-5075 Loader does not have a linking or overlay capability. This capability will be added.

The expansion of RAD-5075 to be the executive for a hybrid system including multiple Sigma CPU's is being planned. The major motivation for this consideration is the achievement of a multiple real-time job capability together with background batch processing within one unified system.

APPENDIX A

CONTROL CARD	RESULTING OPERATION
!JOB Specification	The JOB command specifies the completion of the previous job and the beginning of a new one. The Job Information Table (JIT) in the Monitor is initialized and CCI returns nominally to the card reader for its next control card.
!ASSIGN Specification	If the user wishes to use any of the various peripheral devices differently than the Monitor's nominal assignments of functions, he may so specify with an ASSIGN command. The appropriate information will be put into the specified Data Control Block (DCB).
!SYMBOL Specification	When CCI encounters a SYMBOL control card, it does an absolute load of the SYMBOL processor area. SYMBOL then goes to the nominal or assigned symbolic input device for its source and puts its binary output onto the nominal or assigned BO device.
!FORTRAN Specification	When CCI encounters a FORTRAN control card the operation is precisely as described for SYMBOL. When a processor such as SYMBOL or FORTRAN completes its operation it returns to the Monitor which calls in CCI to read the next control command.
!LOAD Specification	When CCI encounters a LOAD control card the loader which is resident on the RAD is loaded into core. The relocatable program that has been assembled or compiled onto the BO device is linked, made absolute, and loaded into core for execution.
!DLOAD Specification	The DLOAD control card (dumping loader) performs the same function as loader except the absolute binary output is dumped on the specified BO device rather than executed.

RESULTING OPERATION	CONTROL CARD
!RUN Specification	This control command causes the most recent program to be loaded to go into execution. A starting execution address may be specified on this card.
!PMD Specification	The PMD (post mortem dump) control card causes a dump of specified memory area if an error occurs during program execution.
!DATA	The DATA control card informs the Monitor that the following information is to be used in the execution of the presently running program.
!EOD	The EOD control card informs the user that the end of data has been reached. The user can use this information in his program. If a EOD is encountered with no user provision for its use, a message is listed on DO (diagnostic output) device.
!FIN	The FIN control card informs the Monitor and the user that all current jobs have been processed and the computer is entering the idle state.

APPENDIX B

A USER EXAMPLE

Figure 10, Second Order Loop Implementation Diagram, illustrates a simple implementation of the second order differential equation that describes a harmonic oscillator.

Figure 11, User Program Example, shows a sample program listing for setting up the problem, defining an input/output pattern, connecting and servicing interrupts, controlling the analog computer modes, and responding to the

interrupts. The problem as programmed would be in the compute mode for 100 seconds.

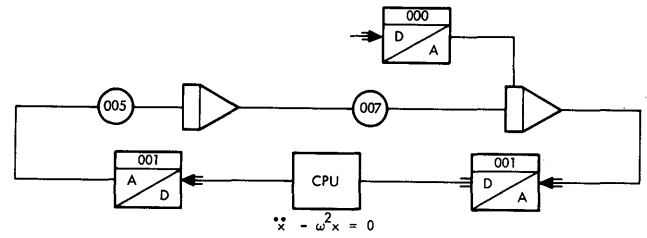


FIGURE 10—Second order loop implementation diagram

```

C PROGRAM LISTING FOR USER EXAMPLE
COMMON IFLAG,FDBCK
EXTERNAL CLOOP
IFLAG=0
KOUNT=1000

C THE FOLLOWING DEFINES THE I/O PATTERN FOR THE INTERRUPT ROUTINE
CALL STRTPT(1)
CALL RADCRP(001,FDBCK,0) { GENERATES CODE TO READ THE
CALL WDACRP(001,FDBCK,0) { OUTPUT OF MULTIPLEXER 1 AND
CALL ENDPAT(1,0)          { WRITE DAC 1 WITH THAT VALUE

C THE FOLLOWING INITIALIZES THE PROBLEM AND CONNECTS AND SERVICES INTERRUPTS
CALL SETPOT(005,.1,007,.1) SETS ATTENUATORS ON THE ANALOG COMPUTER
CALL MODE('R')             SELECTS THE RESET MODE ON THE ANALOG COMPUTER
CALL WPIG(KOUNT,MICRO)     SETS THE INTERRUPT INTERVAL AT 1 MILLISECOND
CALL WDACR(000,1.,0)       SETS THE INITIAL CONDITION ON THE INTEGRATOR
CALL WDACR(001,-1.,0)     THIS CLOSSES THE LOOP IN INITIAL CONDITION
CALL CONINT(2,CLOOP)       CONNECTS INTERRUPT 2 TO THE USER ROUTINE,CLOOP
CALL INTSER(2,2)           ARMS AND ENABLES INTERRUPT 2
CALL MODE('C')             SELECTS THE COMPUTE MODE ON THE ANALOG COMPUTER
CALL RELEASE                { RELEASES THE CPU TO POSSIBLE BACKGROUND
END                          { TASKS SINCE PRE-REAL TIME TASKS ARE COMPLETED

SUBROUTINE CLOOP
COMMON IFLAG,FDBCK
CALL PATTRN(1)              CALLS THE I/O PATTERN TO EXECUTION
IFLAG=IFLAG+1              { COUNTER FOR DETERMINING THE
IF(IFLAG.EQ.100000) GO TO 100 { DURATION OF COMPUTE
RETURN

100 CALL MODE('R')          RETURNS ANALOG TO THE RESET MODE
CALL MODE('P')             RETURNS ANALOG TO THE POTSET MODE
CALL INTSER(1,2)           DISARMS INTERRUPT 2
CALL RLSINT(2)             RELEASES INTERRUPT 2
CALL RLSPAT                RELEASES THE I/O PATTERN
RETURN                     JOB TERMINATED
END
    
```

FIGURE 11—User program example

Growing pains in the evolution of hybrid executives

by MARTIN D. THOMPSON

UNIVAC
St. Paul, Minnesota

INTRODUCTION

The development of a hybrid executive creates problems that go beyond those normally encountered in a digital executive. One of the biggest problems results from bringing two divergent human elements together in the same system. The analog user is "real-world" oriented with an extensive knowledge of the hardware involved in his problem. The digital user is often concerned with batch data processing and scientific computation that has been written in a high level compiler language.

While the digital user has some critical high priority programs to be scheduled, the main emphasis is directed toward completing the execution of as many programs as possible. Because of this, most digital executives stress orderly allocation of new programs, a minimum amount of re-allocation and effective utilization of all parts of the system.

The analog user generally wants control of the system on a demand basis and will retain a high priority level throughout the run. He occasionally requires direct contact with the operating system and will delay the run while deciding what action should be taken.

Part of the problem facing the hybrid community is to persuade the analog user to understand and appreciate the tools that are offered in a digital system. This will become a lesser evil as additional hybrid experience is obtained by the analog user.

The other problem is educating the digital systems programmer in the needs of the hybrid user. This is of immediate concern since the systems programmer will have the responsibility of modifying the digital executive in a way that will satisfy these needs and still retain an efficient operating system. In some situations, both worlds can be satisfied, but often there is a need for compromise.

The object of this paper is to show what happened when a digitally-oriented software group attempted to

bend an existing capability to provide a sophisticated hybrid executive.

A hybrid education

When first given the task of providing the necessary software to control a Hybrid configuration, we began a study effort to determine what would be needed to provide a good software package. Obviously the first move was to peruse any information obtainable that was related to existing software in an attempt to start as close to the "state of the art" as possible. The next step was a survey of a number of hybrid users and designers in an attempt to determine their future needs and desires as well as what objections they had to the systems that they had developed.

A return to the drawing board ensued in an attempt to determine how best to adapt standard digital hardware and software systems in a way that would maintain the capabilities that existed for digital processing and still provide the real world needs of a hybrid computer. Since the system to be developed would be used by yet-to-be-named users to solve problems that could be quite diversified, it was mandatory that the chosen executive would be very generalized and offer modularity such that the basic philosophies could be transferred from one user to another, or from one digital system to another without major overhaul.

Some of the more critical needs for a hybrid system were itemized. Primarily, there was the need for high speed initiation of hybrid communication as a result of some stimulation that had occurred on the analog side of the system. Examples of this would be a time-out from an interval timer or a fault condition. In addition to fast reaction time, it is imperative that any variations in this time should be minimized and, if possible, completely removed.

In many of the critical real-time problems solved on a hybrid computer, the digital portion can easily become

compute-limited. Therefore it is very important that the available compute time in a given cycle be as predictable and liberal as possible. The chaining of I/O commands and independent data transfer to memory seem to be essential for maximizing and stabilizing compute time. Also, special consideration must be given to any possible reduction in executive overhead during a real-time operation.

Since most simulation laboratories are very cyclic in their needs for a large scale hybrid capability, justification for a large scale digital computer can rarely be made if the machine is completely dedicated. It is therefore necessary to retain a powerful batch processing facility under executive control within the digital portion of a hybrid system to supplement the high priority hybrid operations.

The hybrid portion of the executive should be modular to allow a user to specify only those segments that will be required, thereby conserving memory and allocation of peripherals. It should be general enough to allow the user to adapt it to various standard digital executives without major modifications. It should also be easy to adapt the software to variations in the hybrid system configuration such as changes in the number of channels used, or additions and deletions of hybrid devices or components related to these channels.

In some installations there will be multiple analogs operating independently, and each must react as though it were linked to a dedicated digital computer. To maintain this type of environment, it is necessary for all hybrid routines to be re-entrant, which would also reduce the reaction time for processing high priority interrupts. Another useful tool for a multiple analog environment would be a remote console located with each analog and providing the user full communication with the executive. This could be used for initialization of hybrid runs, normal debugging procedures or communication with the analog computer, as well as other operations normally offered on an I/O console.

In order to retain the full flexibility of use by various systems, all hybrid software routines must be as equally accessible from a high level language such as Fortran as they are by the assembly language. This would include any executive support routines such as those provided on a system program library. The program library would have to contain an assortment of standard hybrid routines such as an arbitrary function generator, a continual simulation language, and fast-response math routines, and be expandable to allow the user to add any routines that would be used frequently in his system.

At another level beyond the normal priority structure in a digital system, there is a need for a secondary priority network that is flexible and related strictly to the hybrid environment. This network should incor-

porate the speed of hardware and at the same time allow modification and interpretation under program control. To satisfy this requirement, as well as others previously mentioned, leads to the investigation of more effective design of the potential capabilities of the hybrid interface box.

In surveying our resources, it was learned that a very solid [®]UNIVAC 1108 hybrid system was in operation at the Naval Undersea Warfare Center (NUWC) in Pasadena, California. (See Figure 1). Although their system was quite specialized for solving their particular problems, it was evident that a considerable amount of planning and insight to the needs of the hybrid user existed. They had grown from a smaller-scale digital and minimum analog configuration to a sophisticated large scale system, and had expanded their executive development during the transition. NUWC and Univac personnel had modified the standard 1107/8 executive, which was basically a batch processing executive, to provide the real-time features necessary for their hybrid operation.

The most difficult problem encountered was to retain an effective batch processing capability on a machine that was to be dedicated to hybrid simulation on demand. The NUWC approach to the problem was to develop a "batch shelving" routine to provide high speed swapping between the batch and real-time mode of operation.

The first function of the shelving routine is to load the hybrid simulation program into the resident area of the drum, set appropriate retrieval information in the executive and issue a notification of a successful load completion to the console typewriter. The loading routine will then terminate, allowing the system to continue routine batch processing.

The second function of the shelving process is to respond to a type-in or an external interrupt on a hybrid channel that indicates a desire to execute the hybrid simulation program. A number of error checks are made at this point. The status of the operational program is

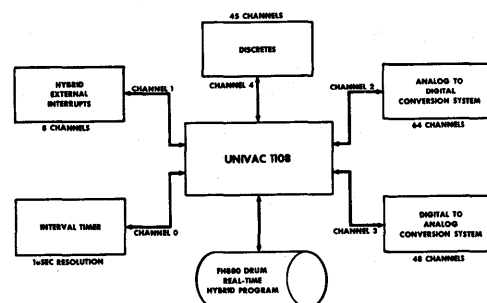


FIGURE 1

checked so that an attempt is not made to shelve a running simulation program and the status of the drum area is checked to verify that a simulation program is available for execution. If any errors are detected, the appropriate message is typed, the shelving request is ignored and batch processing continues.

If the verification is successful, the next step is to store all volatile flags and tables and control registers such as accumulators and index registers. After the environment is stored, the contents of user core are written on the drum and the simulation program is placed in user core. The console is notified that the real-time program is started and control is transferred to the starting address of the simulation program. The required time for the initiation process was determined to be less than two seconds.

The final function of the shelving process is the response to a type-in or an external interrupt requesting the termination of the simulation program. When this occurs, the simulation program is overlayed with the batch program and the batch environment is reset. The computer time used by the simulation program is calculated in a way that no simulation time is charged against the batch program. Less than one second is required from when the termination routine is entered until the batch program obtains control.

All device to device activity such as printing or punching cards from drum files is allowed to continue during the real-time operation. It is not necessary to suspend these operations since the CPU time used does not have a significant effect on the timing requirements of the real-time program.

In order to amplify the interface and provide a straightforward, timesaving software approach, the hybrid world was dispersed over five 1108 I/O channels. The dispersion was made as follows: the interval timer was placed on channel 0, unsolicited hybrid external interrupts were received on channel 1, the analog to digital (A/D) conversion system was on channel 2, the digital to analog (D/A) conversion system was on channel 3 and the discrettes were on channel 4.

The result of this breakdown was a reduction in the interface requirements as well as a reduction in the communication control and command formatting requirements placed on the executive system. Examples of this would include D/A or A/D conversions. The command would indicate which channel should be used, and no directional commands would have to be formatted for transfer to the interface, since the channel used would be self-explanatory.

For the type of operation existing at NUWC their system was very satisfactory, and it was felt that this system would provide a solid foundation for building a generalized hybrid executive.

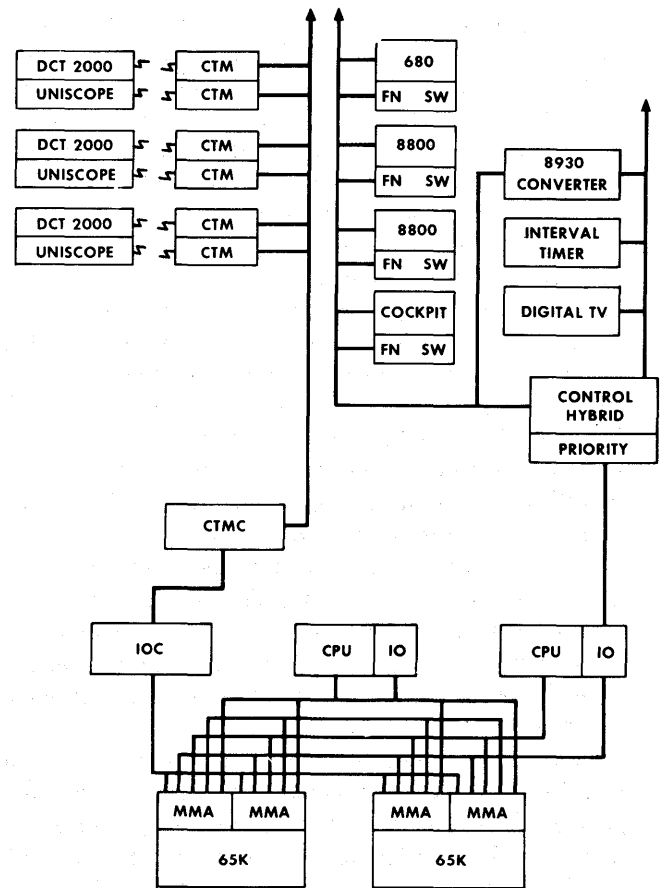


FIGURE 2

Subsequently, the NASA Manned Spacecraft Center in Houston, Texas indicated a desire to develop a hybrid facility using the 1108 multi-processing computer. See Figure 2. Having designed several dedicated systems, this type of operation offered many new challenges, but also provided the tools needed to create a much more powerful hybrid system.

The first and most obvious challenge was to blend the known hybrid software needs into the more sophisticated multi-processing executive. Although the pitfalls became more numerous in making modifications to an executive of this magnitude, it provided many solutions to system problems that had existed in converting a batch processing executive. It was a real-time executive with existing procedures for handling remote consoles while a real-time program was being executed.

The projected hybrid system at NASA will have up to six analogs operating simultaneously and independently. With this type of system, scheduling and allocation of facilities can become very critical. So it was necessary to extend the existing features for the assignment and release as well as the continual verification of the hybrid facilities.

The hybrid facilities were identified by labeling, as devices, such things as analog computers, groups of A/D and D/A channels, the interval timers and discrete devices. By identifying "hybrid real-time" as one of the facilities, the hybrid worker program could be assured that it would be the only hybrid program with critical real-time demands using the hybrid channel. By carefully assigning the proper priority to his real-time operation, and dynamically assigning the facility "hybrid real-time" only when the timing becomes critical, the user has the tools needed for a very efficient system.

The assigning and releasing of facilities can be done at run time through the control stream or dynamically by the operating program. The "Communication Routines" that verify and format the I/O packet establish the facilities to be used to execute the packet. The Hybrid Handler, the routine that executes the I/O packet, will validate the availability of facilities before execution. If the facilities are not available, the worker program will be notified and corrective action can take place. If a job is terminated for any reason, the standard executive procedures would release all facilities that had been allocated.

As would be expected with a multi-processing executive, most standard operations must be executed in generalized routines so that the executive will always have complete control of the system. Operations to be replaced in a batch type executive are the verification of an I/O packet, establishing memory restrictions and converting access control words from relative to absolute addresses, the saving of required environment in response to an interrupt, the queuing of interrupts when their priority is not sufficiently high enough to warrant immediate execution, and the queuing of I/O requests when the facilities required are not available.

By designing the executive modifications in such a manner, the hybrid runs will have no adverse effect on the operating system, and the central control structure of the executive will contain all necessary information on the status of all system components. This is especially desirable when there is external interrogation from conversational use of a remote device or when a programming bug or computer fault force an untimely abortion of the run.

It was during the NASA design effort that the technique was developed that made the greatest impact on the effectiveness of this over-all hybrid operating system. The 1108 has an interrupt lock-out feature that is set by hardware immediately after an interrupt is received. Any succeeding interrupts are suppressed until the executive does a minimal amount of housekeeping and releases the lock-out. This would normally take 40-60 microseconds. In addition to this time, an interrupt will not be recognized until the current instruction is com-

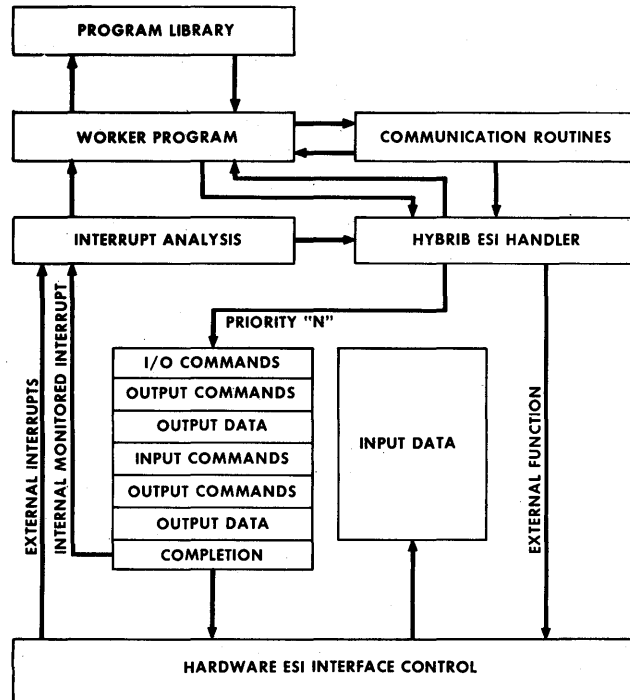


FIGURE 3

plete. In most cases the instruction time is 750 nanoseconds, but in an extreme case of double precision floating point divide, the instruction time is 17.25 microseconds.

The composite of these times created a variation in the interrupt response time and, in situations such as setting analog components in response to an interval timer, this variation represented an uncomfortable margin of error. In designing a dedicated system, this situation was tolerable since the hybrid program could usually keep enough control over the entire operation to be waiting for any cyclic interrupts to occur. In response to unsolicited interrupts, the variation was usually tolerated.

By placing a small amount of capability in the hybrid interface, a method evolved that provided immediate initiation of I/O under the control of the interface. This method included a variation on externally specified indexing (ESI). Details on the technique are included in the description of the basic executive.

General hybrid executive

The out-growth of these previously-mentioned experiences was a modular set of routines that could be merged with an existing digital executive. The primary goals were twofold: to create a system that would allow the hybrid user the means for tailoring and expansion, and to retain the capabilities offered by the digital

executive. In order to accomplish these goals, the software was designed in two sections, Program Library and Executive Control. See Figure 3.

The first contained an executive-callable group of subroutines that may be required to support the hybrid operations. These subroutines would be retrieved from mass storage by the executive when required for execution of a hybrid program. Modification of these routines or the addition of new routines could be accomplished with ease.

Since the Fortran program library already contained most of the mathematical routines required for hybrid computation, it was decided that the first routines for the library should include the use of these mathematical capabilities for hybrid application tools. The first subroutine was a completely digital arbitrary function generator for the evaluation of functions with a maximum of four independent variables. The next effort was a continuous system simulation package that would utilize, in addition to the conventional math routines and the function generator, routines which simulated the various analog components. The language syntax was chosen so as to improve its flexibility and usability for a wide range of problems and yet retain the simplicity of a problem-oriented language such as Fortran.

To supplement the application routines, a number of operator-controlled routines were needed to assist the operator in debugging or initializing the hybrid programs. Included among these was a "hybrid interface simulator" that would over- lay the Hybrid Handler and feed the I/O packets to a table or peripheral device such as a printer, rather than the analog computer. This routine provides the user independent debugging of a hybrid program on a digital computer.

To allow the operator to modify and interrogate component values in the analog computer, a routine was designed to interpret commands from the operator and, using the Communication Routines and the Hybrid Handler, execute the proper I/O packet and continue communications with the operator.

Another routine to improve the flexibility of the system provides the operator with a means to enter tables that indicate the initial conditions that the analog should have before the hybrid program can be executed. All values are set, a time-out is used when necessary to establish success, and the components are read to verify a normal completion. This routine can be executed during a batch processing mode of operation.

To make the basic debugging routines more comprehensive, a routine was provided that would, on breakpoint command from the worker program, read all of the components into a table of the same format as the one used to set up initial conditions. When the dump is complete, it is labeled and written on a mass storage

device. This table may then be used as a restart feature at a later time or may be printed for perusal during post-run analysis.

The Hybrid Executive Control Routines include an initialize package, communication routines, and the Hybrid Handler.

The initialization portion of the executive control has the responsibility of answering the command from an external source that calls for allocation and execution of a hybrid program. In addition to retrieving the hybrid program from mass storage, parameters must be read and verified that will establish the hybrid environment required for the program to run. This includes the setting of general purpose access control words, error recovery routines with associated entry points, clearing all priorities and activating the hybrid channel(s) if necessary.

The Hybrid Communication Routines (HCR) have the responsibility of verification and formatting of the I/O packet that will later be executed by the Hybrid Handler. It was determined beneficial to the system development to adapt these routines to those of the analog manufacturer, since most manufacturers have existing Fortran programs written to perform diagnostic and maintenance tests on the conversion and analog equipment. In addition to these routines, if the digital computer is to be interfaced with existing hybrid equipment, the terminology used is generally well-entrenched in any operational programs the user has developed.

Analog manufacturers are placing greater emphasis on providing a compatible interface to the digital computer. An example of a large scale analog computer with a digital interface that is easily adapted to the ESI philosophy for hybrid communication is the Applied Dynamics AD4. The digital I/O commands necessary for control and operation of the analog are designed to make the analog appear as a standard peripheral device. ADI had previously formatted a set of 43 routines that could be called by Fortran or assembly language programs. These routines provided the means for initialization, data transfer, sensing and setting of discretely and mode control.

Provisions were made to use the Hybrid Communication Routines in either a test mode or a run mode. When operating in the run mode, no error detection takes place since the emphasis is placed on speed. If the test mode contains an error return argument, the error detected is noted and control is returned for user processing. If the test mode does not contain a return argument, the error is printed on the I/O console and continuation of the run is left to the operator.

The executive will make all normal checks on the parameters for an I/O packet. It will also validate the hybrid addresses referenced, and will make special

checks to determine if the argument type is correct for the request, and determine if enough data is available to satisfy the request, or if the range of the data is exceeded. Another condition that may be considered an error is the busy mode from the analog console referenced.

The I/O packet created by the Communication Routines will contain the location of the output buffer, any input buffers, an executive scratch area and the return entrance for normal and error conditions. Also included are indicator fields for the buffer length, priority numbers, error options, status of operation and a bit-encoded word indicating the facilities required for this operation.

The output buffer must contain not only output data but also the I/O commands required to control all communication over the hybrid channel. The interpretation of the request parameters for creating the output buffer must be listed as the primary function of the Communication Routines. Since the packet will not always be executed immediately, all of the verification cannot be done in the one set of routines. Because of this, fields such as the buffer length, facilities used, and the error options must be forwarded to provide the execution routines the information required for assembling the absolute access control words, facility allocation and error recovery procedures. When an I/O packet is to be executed as soon as it is formatted, the Communication Routines will select the proper entrance and transfer the I/O packet directly to the Hybrid Handler.

The Hybrid Handler is the keystone of Hybrid Executive. This is an assembly language routine that provides the interface between the normal digital executive procedures and the hybrid world. It is the only new routine that is keyed to the hardware restrictions of the digital system and hardware capabilities provided in the interface. Because of this, portions of the Hybrid Handler become very specialized and would have to be rewritten for any major change in the interface or if a different digital computer were used. Even though the coding would change, the philosophies should remain stable.

The Hybrid Handler is a composite of re-entrant software routines that are necessary for the execution of the I/O packets developed by the Communication Routines for utilization by the worker programs. These routines control any resulting communication between the hybrid world and the worker programs. The following includes the general task that will be accomplished by the Hybrid Handler:

1. Initialization
2. ESI environment control
3. Verification of the I/O packet
4. Activation of internally-stimulated I/O
5. Programmable priority network

6. I/O completion
7. Error analysis and recovery
8. Interval timer

The initialization portion of the Hybrid Handler has the responsibility of establishing the entire ESI environment for the hybrid channel. This routine is used by the Executive Control initializing package to set up the access control words and error recovery routines for system-dependent error conditions and clearing all priority flip-flops related to the initializing program. The hybrid channel will be activated if necessary.

A special phase of the initialization will allow the program to set up any error or system priority routines needed for the program.

A definition of the hardware activity is necessary to show how the Externally Specified Index (ESI) channel is controlled by the Hybrid Handler. (See Figure 4). When the hybrid interface detects a priority (priority < 128) from the analog world, it will add this unique priority value to the hardwired ESI bias for output (2000₈), place this information in the lower 15 bits of the 36 input data lines, and raise the output data request line (ODR). This will cause the 1108 I/O control to form the appropriate pointer to the main memory access control word (ACW) related to this priority in the lower half of the input control register for the hybrid channel. The 1108 I/O will then transfer the appropriate pre-set block of memory to the hybrid interface. This block of memory may contain commands and output data.

If the commands in the output stream indicate input, the hybrid interface will set the lower 15 bits of the 36 input data lines to the hardwired ESI bias for input (2200₈) plus the priority number, and the requested input data will be entered in the upper 18 bits with sign exten-

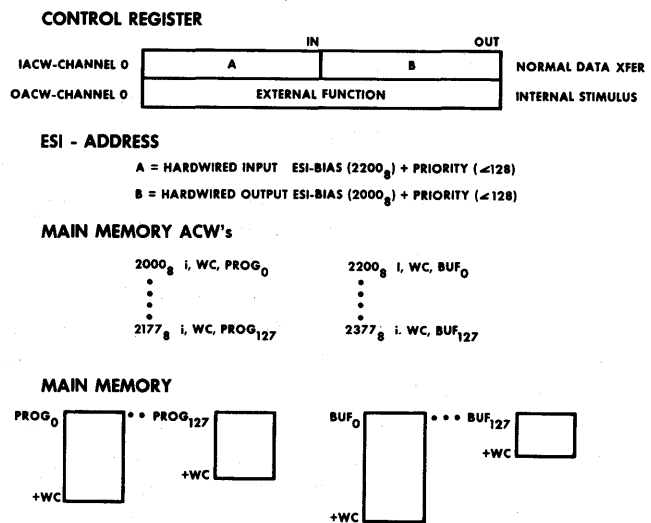


FIGURE 4

sion. Raising the input data request line (IDR) will cause the 1108 I/O control to form the appropriate pointer in the upper half of the related input control register, and the data will be transferred to the related block of memory. The completion of any data transfer operation will always be controlled by a command in the output stream. Execution of this command will cause an output monitored interrupt.

It can be seen that the ESI mode of operation offers immediate initialization of preset I/O commands as a result of an external stimulus. For initial condition set-up and check-out or setting the priority in coder flip-flops, or pre-setting the interval timer, the I/O must be initiated by the digital program.

Any internally-stimulated I/O follows the same procedure as the externally-stimulated I/O previously described. The stimuli to the hybrid interface are provided by a forced external function (EF). The EF may be sent in the output data lines with no conflict, even if a normal output data transfer is taking place. This will cause the hybrid interface to activate priority 0, which is reserved for internally-stimulated I/O. As soon as the currently active buffers are finalized, the highest level priority active (in this case, 0) will be passed to the 1108, and the internal operation will be executed.

The only hybrid operation that will behave in a manner other than that described above is an error condition. After detection of an error condition, the hybrid interface will freeze the priority network and bring the current input operation to a normal completion. An 18 bit status word will be set in the upper half of the input data lines, and 177_s (reserved for error conditions) is placed in the lower half. The IDR is raised, causing the status word to be stored in the 1108, independent of other operations. The hybrid interface will then request an output for priority 177_s, thereby causing a monitored interrupt. The I/O will continue requesting output on priority 177_s, but the freeze on the priority network will have to be released by command from the 1108. This will reinstate I/O to a normal status.

The hybrid interface will be idle, most of the time, waiting for a priority to become active. To allow the interface to handle the situation in an orderly fashion, priority line 176_s is reserved for the idle condition and is always active. When the interface detects line 176_s as the highest priority, no more output data will be requested until a valid priority is activated.

Software control of the Hybrid ESI environment places major emphasis on the speed with which certain operations can be executed. Included among these operation times are: 1) the amount of time that the hybrid channel is deactivated after the occurrence of an output monitored interrupt, 2) the response time from the moment a monitored interrupt occurs until control is

given to the real-time worker program, 3) the initialization time for internally-stimulated I/O, and 4) analysis time for error conditions.

The main advantage of placing the hybrid channel in the ESI mode is that of providing immediate, non-varying initialization of externally-requested I/O while maintaining effective I/O command chaining. The Hybrid Handler provides an effective means of keeping the access control words, the monitored interrupt return addresses, and the data buffers updated within a critical time cycle.

The Hybrid Handler is utilized to set up the execution of I/O in response to two executive requests: 1) Hybrid I/O and Proceed, and 2) Hybrid I/O and Wait. The Handler then verifies all critical fields such as the internal file name, the facilities used word, the error handling options, the I/O priority number; and any critical commands, such as resetting priority numbers, in the output buffer. An option is provided that allows the worker program a means of by-passing the verification phase in order to decrease the executive overhead during the Handler operations.

There are a number of situations where it is necessary for the worker programs to stimulate I/O. When such is the case, the worker program will call upon the Hybrid Handler in almost the same manner as if it was setting up a buffer to be initialized by an external condition. The exception is that it will always set the priority to zero. This will cause the Handler to force an external function on the output data lines to the Hybrid interface. After any currently active I/O is completed, the internal packet will be executed.

If any internally-stimulated I/O is being executed at the time that a new request is received, the new request is queued. The monitored interrupt caused by the completion of an internally-stimulated I/O operation will cause the Handler to initialize the next request before control is returned to the working program.

The majority of the priority network is executed by the hardware in the Priority Encoder. Associated with each priority level are two flip-flops, the ACTIVE and the MASK flip-flops. The ACTIVE flip-flop is set as the result of some condition that exists on an analog facility. It will enter the priority matrix for eventual transmission with the hardwired ESI bias.

I/O commands exist to allow the software to clear the ACTIVE flip-flop related to either the highest active priority or a particular priority. There is also a software capability to arm or disarm any of the priorities through the MASK flip-flop. The mask gates the input to the ACTIVE flip-flop.

Since the programmable portions of the Priority Encoder can be set through normal output commands, it is impossible to maintain an accurate map in the Hybrid

Handler. A routine provides for initialization of the priority system as well as entries for arming or disarming any particular priority or a range of priorities.

To supplement the priority hardware, the Hybrid Handler contains a programmable breakpoint priority where any monitored interrupts occurring with a lower priority will be queued until a command is received to lower the breakpoint to the level of the previous interrupt.

As previously noted, the normal completion of an I/O operation is activated by an output monitored interrupt. This causes the executive to store the priority and reactivate the hybrid channel. The Hybrid Handler will store status information in the I/O packet and for certain priorities the access control words are reset for succeeding interrupts. Interrupts are enabled and control is returned through the executive to the location indicated in the I/O completion field of the I/O packet.

Another method for completion of an I/O operation is by the executive request, Hybrid I/O Cancel. This command will cause the Hybrid Handler to force the word count of the access control word associated with the given priority to zero. An "ignore state" is placed on the monitored interrupt for this priority and, after setting the appropriate status, control is returned to the canceling routine.

Two types of errors are noted by the Hybrid Handler, the parameter error in the I/O packet and an error condition that is detected by the hybrid interface. If an error is found in the I/O packet, control will be returned to the worker program. If the error is detected by the hybrid interface the Handler will check the error options offered in the I/O packet to determine if the operation causing the error should be re-executed a set number of times (with a time delay) or if the operation should be aborted. An error condition option field will exist for every priority where an error detected by the hybrid interface could exist. This will also include any priorities that are completely under the control of the Hybrid Handler. A bit corresponding to each type of error will indicate whether re-execution should be attempted. After detection by the Handler, the priority reserved for errors (177₈) is cleared.

An example of an error condition that could be detected by the hybrid interface is an illegal I/O command. Another is to have the termination command

missing, which would happen if the word count in the access control word did not agree with the output buffer length.

Some situations, such as not having the power on the hybrid facilities referenced, will be referred to the operator.

The interval timer contains an active counter register and a holding register. The active register can be started, stopped and read. The holding register can be loaded under program control. When the active register is decremented to zero, the appropriate priority line to the interface is activated and, if a value has been entered in the holding register, the active register is reset and decrementation is initialized. A multitude of interval timers can be associated with the hybrid system.

The interval timer can be used to synchronize the updating of digital to analog conversions, placing the analog in the hold mode, timing programs or for timing the initialization of any other hybrid communication desired. It can be directed to decrement at seven different rates from one microsecond to one second.

SUMMARY

In conclusion, it is our feeling that we have designed a solid, basic hybrid executive that could easily be tailored to solve the problems of various hybrid users. The divergent philosophies of the analog and digital users had been successfully linked through the executive. The standard digital system, though occasionally compromised, remains a powerful tool. Additional efforts are always required to reduce the executive overhead and find ways to increase the effectiveness of the existing hybrid software. Hopefully, the program library will become more comprehensive as users expand their capabilities. As long as the hybrid executive remains dynamic, it can offer the hybrid user the capabilities that will be needed as hybrid systems are utilized in currently untapped fields.

ACKNOWLEDGMENT

The author would like to express appreciation to Eugene Guliani of Univac, Robert Roulette of EAI, J. T. McKinney of NUWC/Pasadena and Dr. B. B. Johnson of NASA/Houston, whose efforts greatly enhanced the design of the hybrid executive.

The Boeing /Vertol hybrid executive system

by DONALD A. WILLARD

The Boeing Company
Morton, Pennsylvania

INTRODUCTION

The Boeing/Vertol Hybrid Facility was conceived from the start as a state-of-the-art, integrated laboratory to solve engineering problems in design and research of V/STOL aircraft. Considerable experience was gained from The Boeing Huntsville Simulation Center (BHSC) where several generations of hybrid systems were evolved. Since these systems were highly successful both from an economic as well as scientific point of view, it seemed reasonable to pattern the Vertol Facility as much as possible after the Huntsville Facility. This had several immediate benefits. First, the specialized software packages would be compatible. Second, experienced personnel could be transplanted without dropping back on the learning curve. Third, future developments in systems and applications could be jointly undertaken, thereby avoiding duplication of tasks and saving The Boeing Company money.

Within the above ground rules, many features were adopted to provide a balanced, general purpose, scientific hybrid facility. No one feature is considered to be necessarily outstanding, but when all the features are taken in perspective, a truly state-of-the-art, workable hybrid facility has been realized. Some of the major features that have been incorporated are:

1. Thirty-two levels of Priority Interrupts fully supported.
2. A peripheral I/O Supervisor System, capable of supporting Apparent Real Time Tasking of Tape, Printer, and Card Read/Punch devices.
3. A user oriented scope I/O display system to allow "hands-on" control of simulations.
4. A dual interface concept allowing time

shared multiple hybrid application programs.

5. Software control of all analog devices, i.e., potentiometers, logic and mode control, enabling rapid analog system set-up and check-out.
6. A complement of hardware and system interconnection which is identical in each analog console, allowing dynamic logical assignment of any set of analog consoles in a given simulation.
7. The sophistication of Fortran, Level "G" compiler and a Macro assembler.

These features are defined and described in depth in the body of this paper. An example of a classical hybrid application program is also discussed to show how these features are utilized in an operational environment.

Configuration

The configuration of the system (Figure 1) is an IBM 360/44 and Applied Dynamics, Inc. AD-4. Although this paper deals primarily with the Executive System, a better understanding of this system is afforded if some statement of the configuration is made.

Digital

- 1—IBM 360/44 Digital Computer; 256K Byte Storage with high speed registers, priority interrupts, and the following peripherals:
- 2—2260 alphanumeric remote inquiry scopes
- 1—1053 Printer (2260 Hard Copier)
- 2—2315 Single Disk Storage Devices
- 1—2540 Card Read/Punch (reader = 1100 cards/min)
- 1—1403 1100 line/min printer

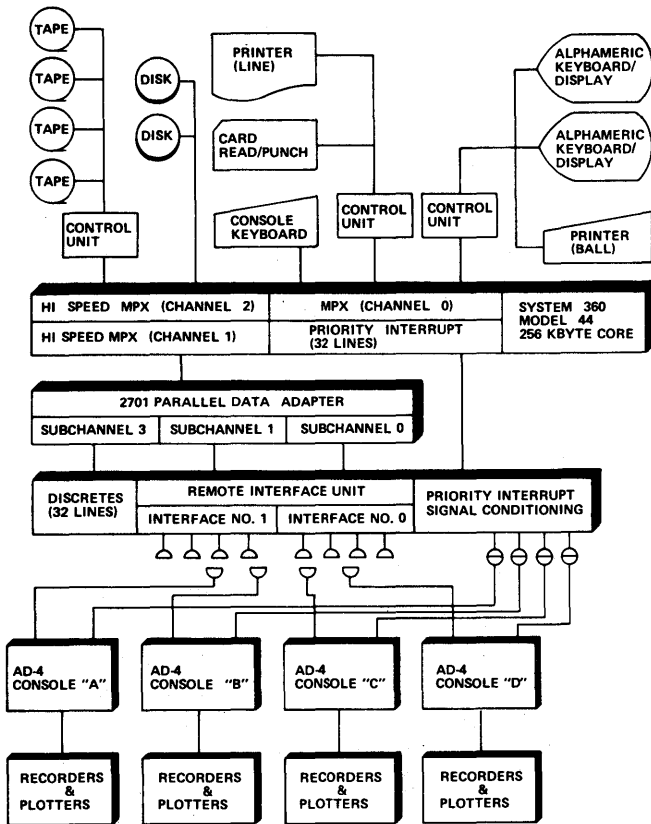


FIGURE 1—Boeing/Vertol hybrid configuration

- 4—2401 Mod 3 800 BPI 112 in/sec tape drives
 1—2701 data channel with 3-32 bit parallel data adapters

Analog and interface

- 4—AD-4, $\frac{3}{4}$ expanded analog consoles

Each console has a self-contained ADC and DAC System. The ADC in each console has a 100KC data rate. Each ADC is multiplexed to 32 lines, 16 of which are sample held. The data are converted sequentially from the 32 lines, but since the non-sample held lines are converted first, the maximum skew time for the 32 words is 160 microseconds. There are 32 double buffered DAC channels per console. The data are held in the first buffer until an appropriate operation code moves the data from the first buffer to the second. At this time the DAC attains the new output voltage.

The Remote Interface Unit (RIU) is shared by the four consoles. This device serves several functions:

1. The junction point for the 32 Priority Interrupt Lines. Each console has all 32 lines terminated upon the patchboard. Also, signal conditioning and buffering is necessary for compatibility between the 360/44 and AD-4.
2. Four sets of ADC buffers, each 32 words. These buffers may be filled simultaneously from the analog consoles, then dumped serially into the 360 core.
3. Two logically independent 24 bit interval timers. The minor bit represents 1 microsecond. They have several running modes described later in this paper.
4. Termination and signal conditioning of 32 discrete lines.

The system is designed to support two logically independent interfaces. Each interface can communicate with any number or combination of the four analog consoles. It is only necessary to select the consoles required for a program and then plug them into the console plugs of either interface. The plugs are numbered logically from "0" to "3" on Interface "0" or "1." The interfaces require that plug "0" always be used with subsequent consoles using the next higher number. Logical console "0" is then the master console and supplies the mega-cycle clock drive to the RIU timer on the interface selected.

The IBM 2701 Parallel Data Adapter is a standard device used by IBM to connect non-standard hardware to an IBM 360 system. Fundamentally, it is a multiplexer channel device capable of supporting up to 4, 32 bit subchannel data lines. Each line is capable of both "read" and "write" operations. The Vertol configuration requires only 3 of the subchannels. One subchannel is dedicated to 32 discrete lines which normally would be connected to devices external to the AD-4 analog consoles, e.g., a simulator. Each of the other two subchannels is dedicated to one of the interfaces described above. Further information on the logical functioning of the interfaces is detailed in the Analog Input/Output Section.

Configuration features

One major feature of this configuration is that it allows rapid "turn-around" of hybrid problems. Turn-around time normally is 15 minutes. This feature is only possible by forcing an identical complement and placement of hardware in each AD-4 console, and using the digital com-

puter to set up and static check under software control. The consoles are then logically addressed much the same as other digital peripheral hardware. Another feature of the configuration is that it allows the simultaneous setup of two hybrid programs on the analog consoles by assigning an interface to each program.

Application program structuring

In the Boeing/Vertol operational hybrid environment, an application program is usually broken into several major blocks which will be referred to throughout the balance of this paper. The purpose of this section is to aid the reader in a better understanding of these major elements and terms.

Three major elements exist. These are "Background," "Foreground" and "Real Time Tasks" (RTT).

The analog computer is in continuous parallel operation with the digital computer. Therefore, the digital must be capable of repetitious execution of tasks to support the analog simulation. This execution is called the Real Time Task. Since the RTT must occur synchronously with the analog, the instructions associated with the RTT will be entered via a Priority Interrupt Line. This implies that any task not driven by a priority interrupt, e.g., the foreground described below, may be interrupted by a priority interrupt. The usage of these interrupts depends upon the structuring of the application program. Normally, synchronism is established by using one of the interface timers to define a time frame. The timer generates an interrupt at the end of each time frame of computation. This time frame is required to be of sufficiently small duration to update the analog computer many times per cycle of the highest problem frequency.

The foreground portion of an applications program contains the necessary communications to the peripheral digital computer devices. During the execution of the program, needs arise for printer, tape, scope display and card read/punch services. These devices require relatively long periods of time to complete their input/output. Since having the Central Processing Unit (CPU) wait for I/O completion would be unacceptable in a hybrid environment, a Peripheral Input/Output (PIO) Supervisor has been incorporated in the Basic Programming System. PIO is described in depth later in the paper. This supervisor allows I/O to the peripheral devices to op-

erate in "Apparent Real Time." Normally, buffers are filled or used by the RTT. These core buffers are checked for status when the program is in the foreground mode. Then, I/O is initiated as required such that the necessary I/O is completed before the buffers are again needed by the RTT. Under this supervisor, control is always returned to the application programmer immediately after start of I/O. Hence, the peripheral devices are active during problem solution, and although the I/O action occurs asynchronously with the analog computer, the I/O looks like it is apparently in real time.

Background is defined as a low priority task such as "Compilation," "Assembly" and "Utilities." Presently, Background does not co-exist with Foreground and RTT under any operational system. It is necessary to reload the system to include Background with Foreground and RTT.

Hybrid compilation and assembly

Two IBM supplied systems are utilized. These are named by IBM as BPS (Basic Programming System) and PS (Programming System). BPS was released about one year before PS, and therefore became the base system for modification to support a hybrid environment. Many man years were expended to perform this task. The modified BPS is acceptable for a hybrid operating system but has several disadvantages:

1. Tape Resident
2. "E" Level Fortran Compiler
3. No Macro Facility
4. Unsupported by IBM as of 2/15/68

These disadvantages have largely been overcome by PS. In order to use the compiler, assembler and utilities of PS, but still execute under control of the BPS system as modified for the hybrid environment, it was only necessary to provide for the incompatibilities between PS and BPS in the object deck's External Symbol Dictionary (ESD). A routine called ASSET has been added to the PS Library and is executed via appropriate Job Control Language. This program inspects and modifies object modules produced by the 44 PS Assembler and 44 PS Fortran Compiler. The output consists of edited (new) object decks that are compatible with the 44 BPS Fortran Loader and Library.

The advantages of this dual system are:

1. "G" level compiler, providing better diagnostics
2. Faster compilations

3. A load and go system for digital simulation
4. Macro assembler
5. More extensive utilities
6. IBM supported with growth capability into Data Acquisition Multi-Programming (DAMPS)

Hybrid executive system

After the BPS object program is produced, execution is controlled by the Fortran System Director (FSD) and its subsets. This system (Figure 2) is a modification of the IBM supplied BPS. The modifications were jointly designed and implemented by IBM and The Boeing Company.

The principal paths of communication for hybrid are shown. Only the parts of the system that are specifically tailored for hybrid will be discussed as a complete treatise of the entire Hybrid BPS would involve many IBM published manuals. Each major module is treated in some depth with particular emphasis upon the AIO Supervisor and Library Subroutines. Although this module is very similar in function to the Peripheral I/O

Supervisor and Library, it is very configuration and conceptually oriented to the AD-4 interface.

The Hybrid Executive System's primary purpose is to effect real time and apparent real time I/O for the various devices in the system. This was accomplished through modifications to the Fortran System Director and IBCOM, plus the addition of two supervisors with appropriate calling routines. One of the major advantages to using the IBM supplied BPS is the ability to READ and WRITE under Fortran format control. This was accomplished by assigning to other routines some of the responsibilities of the FSD and IBCOM, such as posting and analyzing I/O completion status. In this manner, the nucleus of the operating system was "fooled" into believing that its environment had not been altered. The rest of this section is a description of the modifications and additions to this basic system and their effects upon the 360/AD-4 environment.

Fortran System Director (FSD)

The FSD is the nucleus of the BPS Fortran System. It is in residence at all times and provides a method whereby the compiler, loader, editor and object modules may be called in and executed. One of the principal tasks of FSD is to process channel interrupts. For the hybrid facility an I/O Active Table and an I/O Interrupt Filter were added to FSD. By checking the I/O Active Table, interrupts are analyzed to determine if they are analog (AIO) or peripheral (PIO). Linkage to the respective analog or peripheral supervisor is set up if a real time interrupt is felt. If it is a non-real time interrupt, linkage is established to provide a return to the standard FSD interrupt processor.

A second addition to FSD was the Real Time Communications Region. This was necessary to provide linkage to the Supervisor Routines and a save area for the Priority Interrupt Supervisor. The following information is contained in this region:

1. Hybrid Status and Indicators
2. Save area for 32 Priority Interrupt Supervisors
3. Pointers to Analog and Peripheral I/O Supervisors
4. Pointer to the I/O Active Table
5. The Priority Interrupts enable and disable masks
6. The 64 Program Status Words for the Priority Interrupts

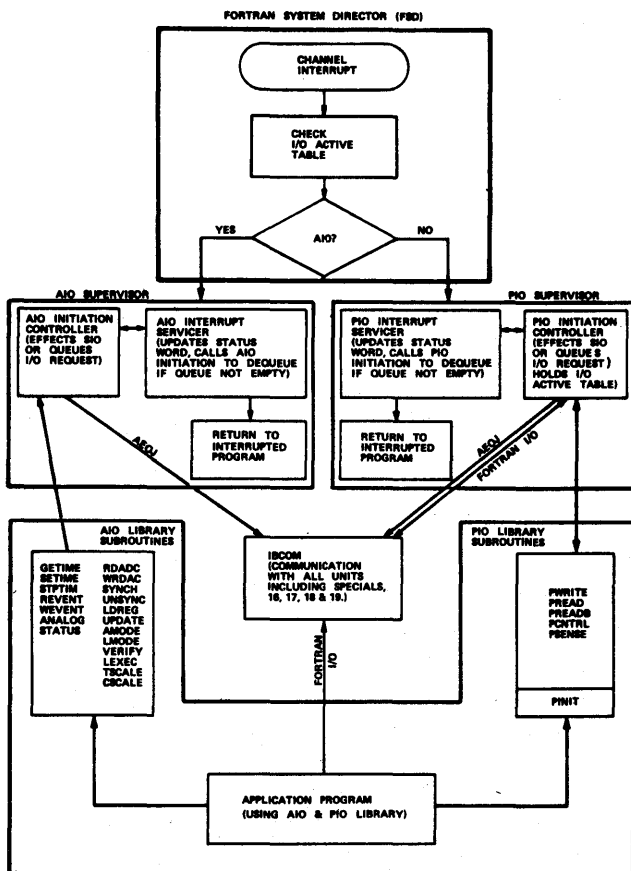


FIGURE 2—Hybrid operating system

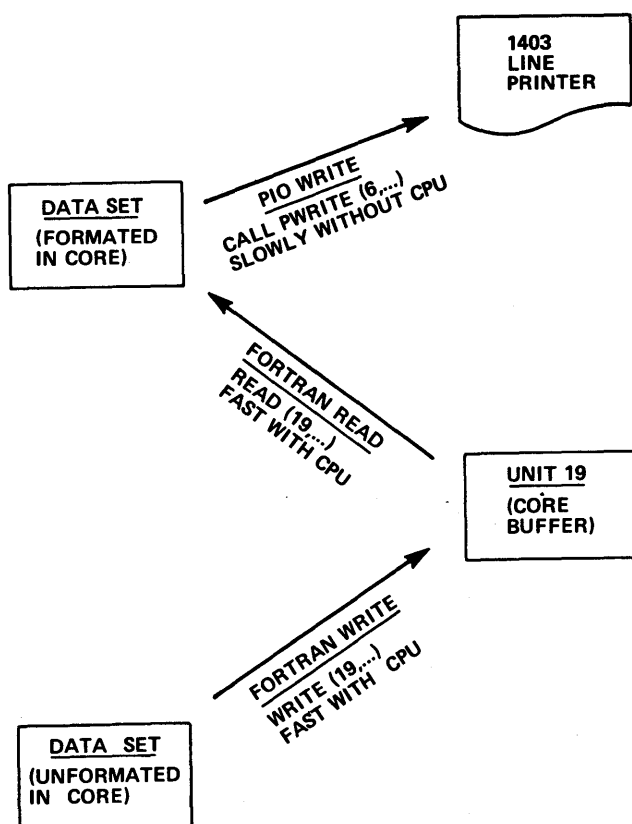


FIGURE 3—Unit 19 utilization

Communication between FSD and the PIO and AIO routines is necessary. Since FSD is an absolute module, blank words are internally reserved to serve as pointers to these routines. PINIT, a subroutine called from the applications program, is executed immediately after loading all the modules of the object program. This initialization routine picks up the entry addresses of the PIO and AIO routines and appropriately places them into the FSD communications region.

IBCOM and FIOCS

The communications system for Fortran I/O is in IBCOM. Hybrid I/O has made special demands upon I/O functions. Several special units (1053 ball printer, two 2260 alphanumeric scopes and an internal unit) are supported. These are logical units 16, 17, 18 and 19. Fortran Input-Output Control System (FIOCS) has been modified such that all device I/O is routed thru the PIO system described later.

A requirement exists to read and write the digital peripheral devices in apparent real time.

Since core to core operations are relatively rapid, an internal scratch, unit (19), of 132 characters has been added. Using this unit, the formatting features of Fortran may be used. As an example (Figure 3); if it were desired to do an apparent real time write on the line printer in alphanumeric format of floating point numbers in core, the data would have to be reformatted before being written using PIO calls, as these calls do not have format capability. Using Unit 19 as an intermediary, the data are written under Fortran format control on Unit 19; then read under Fortran to the application program buffer, and finally the printer is written by a PWRITE call of PIO. The status of the final write is checked by the application program to make sure that the buffer was dumped correctly before the process is repeated.

A second usage of Unit 19 is to transfer alphanumeric constants under Fortran from Format Statements to Data Areas. These constants might be used for comparison with key control words. Key words are necessary to allow the application programmer to choose options via 2260 keyboard inputs. This process is described in the Hybrid Applications Example later in the paper. A set of these control words would be generated prior to real time execution. Later, these words would be compared with keyboard inputs through logical IF statements. No other reasonable method of generation is available as the BPS Fortran System does not support literals or hexadecimal constants.

Analog Input/Output Supervisor

This system manages the flow of data to and from the analog devices. It consists of two major routines:

Analog I/O initiation controller

All I/O initiation to analog is generated in this routine. To perform this task in an orderly fashion, it does the following:

1. Bookkeeping
2. Determines the availability of the device
3. Queues the request if the device is not available
4. Starts I/O if the device is available
5. Handles device inoperative responses (I/O was attempted but could not be successfully initiated)

Analog interrupt servicer

When I/O on a device is completed, this routine processes the device interrupt return. This interrupt will have been sent by way of the FSD interrupt filter, described earlier. To accomplish this task the following actions are taken:

1. Field and post the status of the I/O completion.
2. Check the Queue Table for pending AIO calls. If other tasks are queued, the I/O Initiation Controller Routine is again called.
3. Load the Program Status Word (PSW) for return to the interrupted program.

A device active and queue table are maintained to facilitate the analog supervisor in its duties.

Analog Input/Output (AIO)

All requests for I/O under the control of the AIO supervisor arise from subroutines which are called from the applications program. These are:

SYNC	Synchronizes the DAC's and/or ADC's conversion with the timer.
UNSYNC	Removes control of the DAC's and/or ADC's from the timer.
RDADC	Read ADC buffer into main storage.
WRDAC	Writes the DAC's buffer.
UPDATE	Updates a set of DAC output buffers from the associated DAC input buffers.
REVENT	Reads discrete lines via an interface channel or via the discretized terminal station channel.
WEVENT	Write corollary of REVENT.
SETIME	Starts the timer.
GETIME	Reads the current value of the timer.
STPTIM	Stop the timer.

Other routines are available to control the modes and registers of the analog consoles. They implement all requirements for I/O via Channel 1. Of the three channels in use (0, 1 & 2), only Channel 2 with its tape drives has a higher priority in the servicing of I/O interrupts. Channel 1 utilizes a high speed multiplexer with three subchannels. One subchannel is for communication with a 32 terminal station which provides

for the autonomous input and output of discrete signals. Each of the two remaining subchannels communicate with its own remote hybrid computing interface housed in a Remote Interface Unit (RIU). The RIU was designed and built by Applied Dynamics, Inc. to Boeing specifications.

From the software point of view the discrete signal terminal station and the remote interfaces are three peripheral devices. All communication with them is by standard I/O commands. The AIO routines, therefore, effect all AIO operations by using a sequence of I/O records, many of which are constructed internally by the subroutines. I/O with respect to the discrete signal terminal station is simple and involves the writing or reading of a four byte, single record.

Apart from the subchannel addresses, the two interface subchannels are indistinguishable. An interface subchannel connects with a remote interface which provides four outlet plugs on the RIU corresponding to four logical analog console addresses 0, 1, 2, & 3. (Figure 1). An interval timer, which is an integral part of the remote interface, is driven by a clock pulse supplied from the physical console that is plugged as logical 0. Four plugs corresponding to four physical analog consoles A, B, C and D are also located on the RIU so that the physical units may be assigned to the logical unit addresses and interfaces as desired. Functionally, the plugs are indistinguishable.

Each plug connects both to an AD-4 console and to a 32 word ADC buffer using 16 bit words. The buffer is filled from the AD-4 Console when a conversion sequence for the console is initiated. A single ADC operation may initiate the conversion sequence for either a single console, or for all consoles simultaneously. Thus, the maximum time skew from start to finish of the conversion sequence is a function of a single console's ADC system. A conversion sequence may be initiated either by a program call or directly by the timer co-incident (overflow) pulse. When all conversions are completed (for one or all consoles) a pulse is delivered to the logic patchboards on the interface. In the hybrid technique using repetitive sampling, it is at this point that the real time task (RTT) may begin. Hence, this pulse is normally patched to a priority interrupt line (PIL) to initiate the RTT application program. In the RTT another AIO subroutine is then called to read into core as a single record either a single ADC buffer or all of them sequentially.

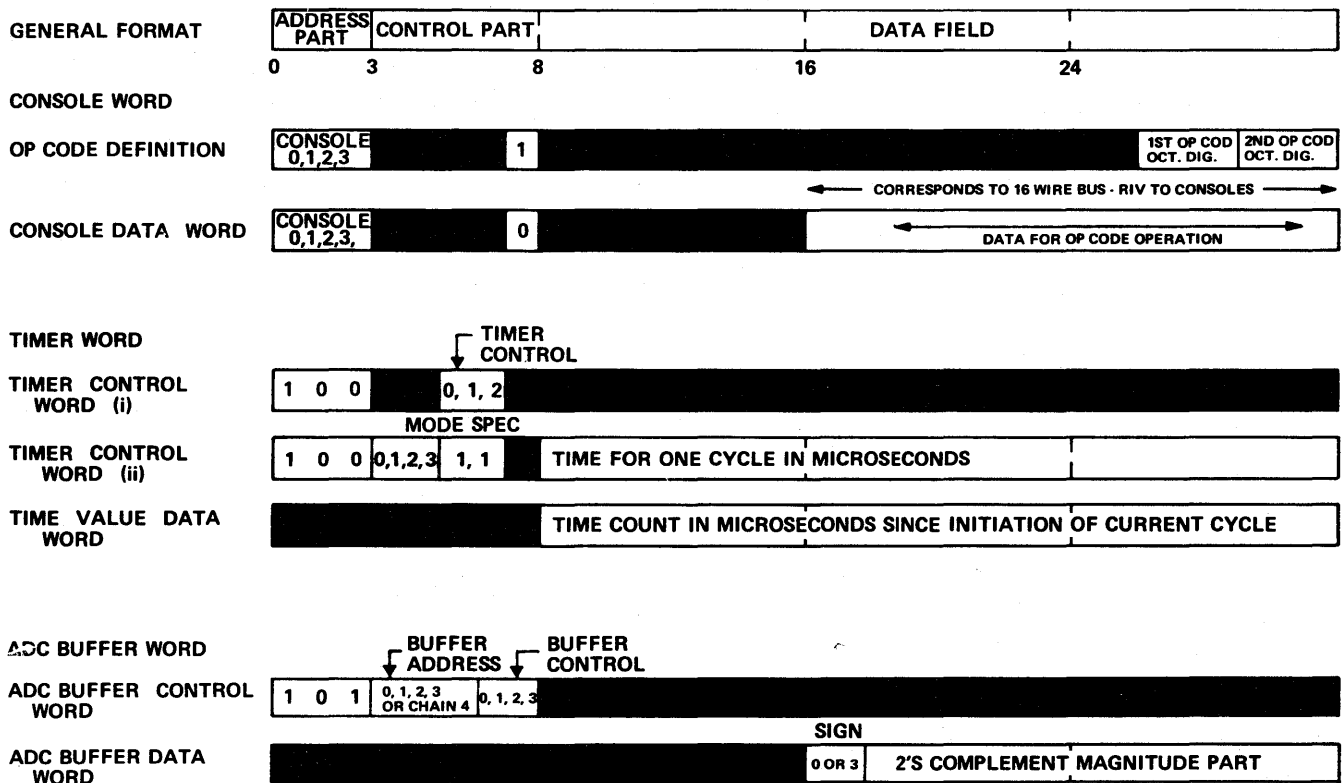
The RIU interval timer is the main controller for the interface when repetitive time control is desired. The timer can be programmed to define a single time interval and then stop or to recycle after each interval. It can also be programmed to redefine the interval time and initiate a new cycle either immediately or at the end of the current cycle. The count is from zero until coincidence occurs with a final value register. Upon this event a pulse is generated which is available at all the AD-4 console patchboards plugged to that interface. In the sampling technique used, in addition to initiating the ADC conversion sequence, this pulse may be patched to update the DAC final registers from the initial DAC registers which were loaded by a previous RTT application subroutine call.

All words (Figure 4) sent to the RIU carry an address referring to either the interval timer, the ADC buffer system or to one of the consoles. Those sent to the timer can control its mode of operation, its actual operation, or request the current value of the megacycle counter. Four

Fortran subroutines cover these operations. Words to the ADC buffer system carry a buffer address referring either to all buffers or just one of the four. Fortran subroutines control the buffer's mode of operation providing program or timer synched initiation of the conversion sequences and facilitate the transfer of the buffer contents to main storage.

For those words addressed to a console, the data part of the subchannel words is simply passed on to the console by the RIU. These half words of 16 bits pass either an operation code (OP code) of the AD-4 digital control system or data which will be processed according to the current OP code in the AD-4 OP code register. One bit of the first half word is used by the RIU to distinguish between OP code words (bit=1) and data words (bit=0), and to signal the AD-4 console accordingly. The repertoire of OP codes covers all the operations necessary for the control of the analog console and the selecting and passing of discrete data and digitized analog data to and from the RIU. Some OP codes cause in-

FIGURE 4—Formats of analog I/O words



LEGEND [] MEANS 'NOT USED'

crementing of addresses in the AD-4 addressing system. Records written to an analog console may have OP code words and data words mixed. Records are read from an analog console to take from it single 16-bit data words or sequences of such words under the control of the appropriate OP codes. The AD-4 control system is essentially digital in concept, and consequently it interfaces with the digital computing world in a straightforward and expeditious manner.

Peripheral I/O Supervisor

This module functions similarly to Analog I/O Supervisor previously described. It contains all PIO request queueing, request staging, interrupt handling and initialization logic. All interrupts are disabled when any part of the module is in control.

The main interrupt filter in the Fortran System Director turns control over to this module when an I/O interrupt occurs for a device supported by PIO, provided this module has been initialized by a Peripheral Initialization (PINIT) call from the problem program and the interrupting device is entered in the device active table.

PIO library subroutines

The routines of this library request appropriate interface queueing thru the Peripheral I/O Supervisor. They are totally re-entrant and can be called either from the foreground task or the real time task(s). They are:

1. PWRITE (output)
2. PREAD (input)
3. PREADB (read backward tape)
4. PCNTRL (control operations)
5. PSENSE (sensing)

The purpose of the PIO calls is to allow apparent real time programming of the peripheral devices. The FSD is forced to communicate with the devices thru PIO. When a standard Fortran call to a device is made, IBCOM establishes the status word for the device and waits upon the completion of the I/O. Since this would be unacceptable in a real time environment, direct PIO calls require that the application program establish the status word. Upon initiation of the I/O request, control is returned directly to the application calling program. It then becomes the responsibility of the applications program to

check the status word to determine I/O completion and whether or not an error was encountered.

Priority interrupt supervisor

The Priority Interrupt Supervisor consists of two special routines call ATTACH and DETACH. They are called from the application program. These routines allow the application programmer to specify the attachment and enabling of Priority Interrupt Lines (PIL's) or the disabling of same. This is the mechanism that allows the application programmer to specify the routines which will service his real time interrupts. ATTACH normally would be called at program initialization and DETACH whenever desired or at program exit.

Hybrid program control example

A block diagram for the digital control and execution of a typical hybrid program is shown in Figure 5.

After the program is first loaded, the analog consoles, interface, tapes and other equipment used can be logically assigned using key control words on the 2260 display keyboard (Block 1). With this fundamental task completed, the display keyboard acts as a distributor, giving the

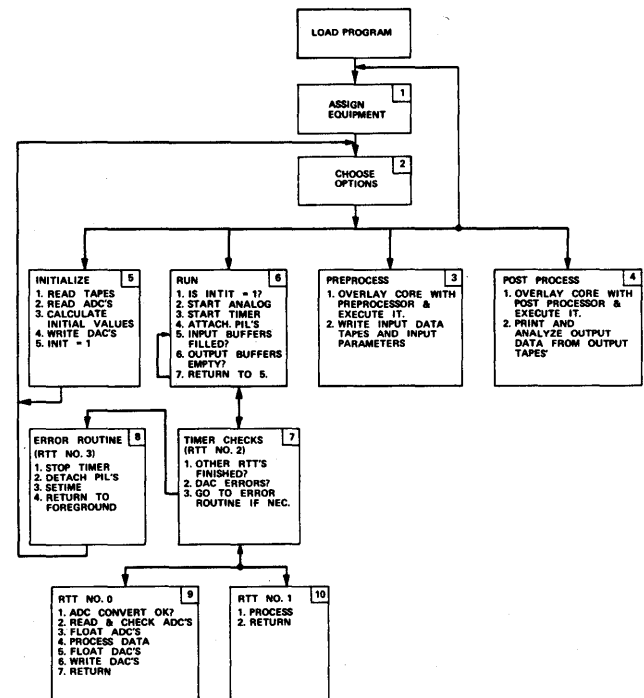


FIGURE 5—Example hybrid program block diagram

application engineer a choice of options (Block 2). This example allows the choice of initializing the program, running, preprocessing or post processing. One might also wish to include selections to display areas of storage, allow for changes in storage, etc. Blocks 3 and 4 of the flow chart allow the operator to set up tapes for use during running, i.e., input data for function generation, and to post process output data tapes that were created during a real time task.

The run option requires controlling a RTT in parallel with analog using the interface timer. Each time a RTT calculation is required, its associated priority interrupt will cause entry to the designated section of executable instructions. The highest priority interrupt will be serviced immediately and when the associated instructions are complete, return will be to the interrupted program. To explain the fabrication of the RTT controls, refer to Blocks 5 thru 10 of Figure 5.

Block 5 is used to initialize the DAC's by performing the frame calculations using the initial ADC values and any tape data as input. After this is performed, a status word is set to one to indicate acceptable completion of the DAC initialization and the program is again given a choice of running, preprocessing or post processing.

If the operator next chooses to run, a check of the initialization word is made to assure proper initial DAC values before proceeding to run. After this check, the analog is put in the operate mode. The timer is started and the priority interrupt lines are attached to the RTT's. Next, the CPU begins execution of an indefinite length loop in the foreground, which is interrupted by either a RTT or a program halt. This loop is shown in Block 6. The status of the I/O buffers are checked. If an I/O buffer becomes ready for data transferral, a request for peripheral I/O is made via the appropriate PIO subroutine call. If the requested device is not active, the start I/O command is issued and control is returned to the application program immediately. If the device is active, the request is queued, and will wait until the device signals completion of current I/O by a channel interrupt before being honored. (See the previous PIO section.) The I/O buffers are generally double, to permit the RTT to work into or out of one while the other is being filled or dumped by the peripheral devices.

Because of this PIO capability of starting

I/O and then returning to the application program immediately until the device I/O is complete, other devices can be started and overlapped in this foreground loop whenever a buffer requires service. Printer, 1053 typewriter and 2260 scope devices, for instance, can be used to give operator messages in apparent parallel operation with the analog.

Since the timer has been set and started before the loop was entered, it will overflow when its preset coincidence value is reached.

A timer pulse (available at the analog) which occurs at timer overflow can be wired so that it will start DAC ADC conversions at the end of a frame and cause a priority interrupt for Block 7. Block 7 will then check to see if the real time tasks have all been completed on time. If not, an error routine is entered and the run is halted. This check is accomplished by setting status words after the appropriate checks have been made.

After completion of Block 7, return is made to the Block 6 loop. The ADC's are now converting and when the ADC multiplexer is finished a pulse appears on the analog logic board. This pulse is patched to the priority interrupt line attached to the RTT of Block 9.

The first action of RTT #0 is to check the conversion and transmission of the ADC's to the interface buffer. Then the ADC interface buffer is read and the data converted to floating point format in core. The frame arithmetic calculations of the process cycle will be executed using the converted ADC data and the resultant data will finally be written to the DAC's. A return is then made to the foreground loop at the point at which it was interrupted. If while the RTT was being performed, a peripheral device had completed its read/write operation, it can interrupt the RTT and see if any more PIO operations are pending. If there are requests pending, the PIO supervisor starts them before returning to the RTT.

If another real time task is needed only once every several frames, it will be given a lower priority and will be executed whenever higher priorities are not controlling the CPU. In this fashion, portions of a lower priority task can be done piecemeal until the whole task is completed. The timer routine would check to see if this task has been completed only once every 10 frames, for example. Until this lower priority RTT executes its RETURN statement, it will al-

ways be serviced when no other higher priorities are requested.

Future hybrid executive system

As listed previously, the Hybrid BPS has several limitations. A joint Boeing Huntsville Simulation Center, Boeing/Vertol and IBM development is under way to produce the next generation Hybrid Executive System. This system when complete will be a modification of the Data Acquisition Multi-Programming System (DAMPS) to support the special requirements of Hybrid. Fundamentally, DAMPS is an extension of PS which will support a Priority Interrupt System and provide an apparent real time I/O scheduler. Features of this system beyond our present system include:

1. Disk residence.
2. "G" level Fortran Compiler load and go.
3. Background compiler, assemble and utilities.

4. IBM supported.

5. CSMP or similar block modeling language.

The final version of this system will be operational in 1969.

ACKNOWLEDGMENT

This paper was written with the assistance of John Rayner, The Boeing Company and edited with the aid of Carol Becknell, International Business Machines, and Larry Snyder, The Boeing Company. Their efforts were most constructive and are greatly appreciated.

The Boeing/Vertol Hybrid Executive System is the result of many man years of effort on the part of numerous individuals, both in The Boeing Company and IBM. Therefore, the Huntsville, Alabama Hybrid Team of Frank Abbot and Roger McAllister of The Boeing Company and William Pierce and William Maasberg of IBM as well as Bill Post and Carol Becknell of IBM, Philadelphia, should be recognized for their major contributions towards this final system.

FAMILY I: Software for NASA-Ames simulation systems

by EDWARD A. JACOBY

Basic Computing Arts, Inc.,
Mountain View, California

and

JAMES S. RABY

NASA-Ames Research Center,
Moffett Field, California

and

DONALD E. ROBINSON*

Applied Simulation Research
Redondo Beach, California

INTRODUCTION

The Ames Research Center (ARC) of the National Aeronautics and Space Administration has two major hybrid computer systems which are used to control piloted simulators of advanced aircraft, such as supersonic transports, and space vehicles, such as the Saturn V Apollo booster.

There are two fundamental problems in hybrid computation. The hybrid user/experimenter invariably lacks the background, experience, and inclination to consider the needs of the computing system. He focuses on his scientific problem, and justifiably objects if the computing system is so sophisticated that it becomes a separate problem. An equally important problem is the excessive programming and checkout time inherent in hybrid computation. Programs are large and divided, part in the digital computer, and part in the analog computer. Programs are complex, requiring extensive synchronizing and control logic. Experimentation requires continual changes to the program and its data on a trial-and-error basis.

The FAMILY I software system includes the

normal functions of a digital monitor, the executive control functions for the simulation program, and an interactive hybrid debugging system. The digital monitor handles language processing, program loading, and provides the running program with digital input/output and interrupt services. The executive controls the simulation. It is a *privileged mode* program that performs many of the functions normally reserved for *monitor mode* programs. It sets and tests discrete lines, responds to reserved interrupts, controls analog-to-digital (AD) and digital-to-analog (DA) data transfers, and performs other simulation executive functions. The debugging system provides powerful assistance in problem setup, static test, and program debugging.

Hybrid simulation has special scheduling requirements; large amounts of computing equipment in different worlds must be controlled and synchronized. With a pilot in the simulation loop, the response time of the computing system becomes critical. The executive controls scheduling and adds only a minimum overhead time to the computational loop.

The paper examines briefly the key participants in a simulation—computers, men, programs—and reviews in some detail the evolution of Ames hy-

*Mr. Robinson was employed by Electronic Associates, Inc. at the time this system was completed.

brid software, its use, its subsystems and their implementation.

The simulation

The Flight Simulation Laboratory at ARC supports a variety of projects. Typical at this time are: (1) studies of the handling characteristics of advanced aircraft (including SST and VTOL) especially during the landing and takeoff phases of a flight; (2) investigation of the ability of astronauts to control the Saturn launch trajectory; and (3) optimization of control systems.

The laboratory has six Electronic Associates Incorporated (EAI) 231R analog computers, two EAI 8800 analog computers and two EAI 8400 digital computers. FAMILY I was designed for use with the 8800 and 8400 computers. The configurations of the two digital computers are shown in Figure 1.

In addition the laboratory has a number of cockpit simulators ranging from fixed-based cabs to a full six-degree-of-freedom cab. A visual simulator is used to provide pilots with an out-of-the-cockpit view of a landing approach. The runways, hangars, and landscape are projected on a screen in front of the pilot so he can make visual approaches as well as instrument approaches. These simulators are all computer driven.

The skills of the individuals using the laboratory are diverse; however, the typical user is an aeronautical engineer with several years experience in programming analog computer simula-

tions. He is accustomed to an open-shop facility where the user does most of the computer programming, debugging and operation. The majority of debugging is done on-line. When the user is satisfied that the program is working correctly, he runs basically that same program for several months using different pilots, changing some aspect of the simulation in an attempt to obtain a "better" configuration of the aircraft. He makes this type of program modification on-line, by changing the value of some variable based on pilot comment and his own engineering judgment.

Software must provide this type of user with the same man-machine interaction that he is accustomed to having when using the analog computer.

The hybrid computer program

A hybrid simulation involves a large diverse complement of computing equipment. In digital memory, there are parts of the program written by the experimenter or his staff, and there are executive functions provided by the hybrid software system. On analog computer patchboards are other parts of the program. In instrument racks and the simulator cockpit, there are batteries of logic panels, timers and control lines which serve as logical extensions to the program; some capable of functioning asynchronously with other parts of the program.

The purpose of the complete computer program is to give the experimenter control over the entire complex of equipment, and to solve the calculations and differential equations in such a way that the total effect represents the characteristics of a physical system, such as an aircraft, in its dynamic states of change with respect to continuous real time. To accomplish this, the program in its total aspect must control the flow of data across lines between the computers and the simulator station where the pilot sits. When all is synchronized, the pilot receives cues in his cockpit that closely simulate the motions and visual responses he receives from a real airplane.

A significant melange of parts and pieces must be fitted together to build the digital section of a hybrid program. Figure 2 shows these elements in a helter-skelter arrangement which suggests the size and complexity of the programmer's task, and the reason that he can often become confused, and make mistakes, or omit key parts of his program by oversight.

As far as he is concerned, he would like to pro-

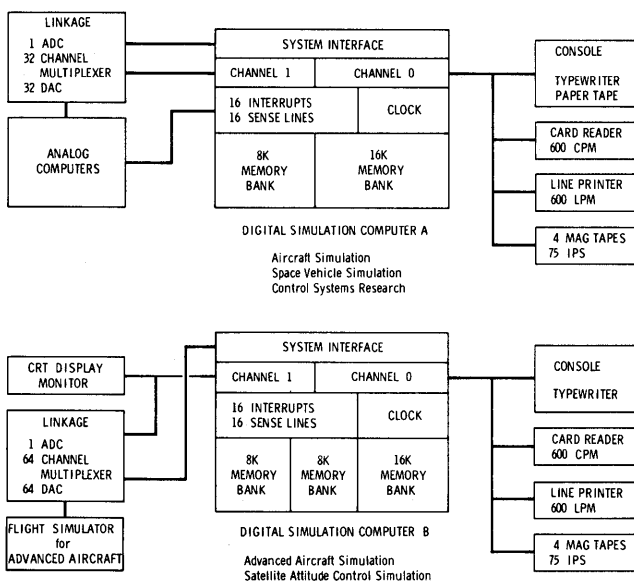


FIGURE 1—NASA-Ames flight simulation laboratory hybrid computers

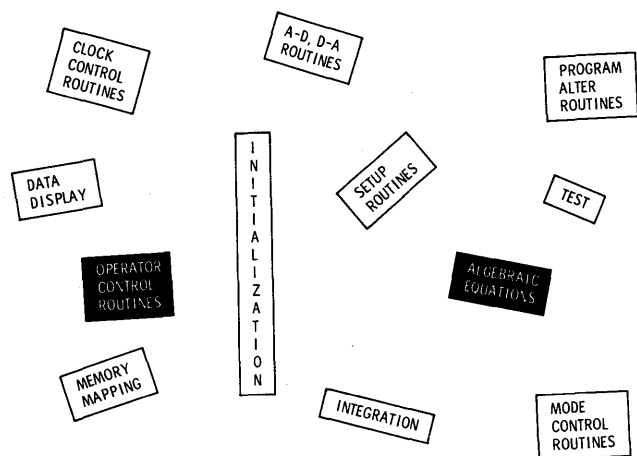


FIGURE 2—Typical hybrid problem elements

gram only those parts of his program shown in black boxes; these are the differential equations that represent the mathematical model of the physical system to be simulated, the interrupt routines that respond to special events occurring in the simulation, and the on-line utility functions he would like the computer program to perform in response to pushbuttons and switches. These utility functions include data displays, use of special initial conditions, printouts on a line printer, and similar functions.

The other parts in Figure 2 are equally necessary, which can be seen by briefly reviewing the typical structure of a hybrid simulation program.

Setup and test section

Before running a simulation, considerable initialization and testing of the whole program is needed. Analog potentiometers must be set and checked. The complete analog program must be checked against known static conditions to verify that the patchboard wiring and component settings are correct. Instruments may need to be calibrated. Certain digital memory locations controlling system operation need to be setup after the digital program is loaded into memory. Most of these tasks are relatively routine, and the programmer would rather not have to program them, but he must provide for their presence.

Computation section

There are three simulation modes of computation: IC (Initial Conditions), Hold, and Operate.

In *IC mode*, all variables in the simulation are set to their initial values by assigning initial con-

ditions directly to certain variables, and computing the equations of the system but without integrating the differential equations.

In *Operate mode*, real-time clocks are started and controlled to keep the entire system in synchronization. AD/DA transfers take place at precisely timed points. The differential equations of the system are computed and integrated. In the digital domain, special numerical integration routines are needed to perform this task. Data history must be recorded at timed points to provide a "movie" of the run. Often, several computational processes must execute at different frequencies, introducing a scheduling problem akin to multi-programming.

In *Hold mode*, real-time clocks and the analog computer are stopped, freezing computations at the values that existed at the instant of entering Hold mode. The experimenter then can spotcheck results, and make changes. To do this he requires a means to readout or set (change) any variable in the simulation, whether analog or digital.

Control section

The most basic control needed in a simulation program is the ability to change from mode to mode at will. With each change to IC, Hold, or Operate, all computing equipment, simulator controls, and instruments must change to the new mode simultaneously.

Other controls, decided upon by the experimenter, are invariably included in a comprehensive simulation. He wants interrupt buttons to impose special conditions, e.g., an engine failure. He wants easy ways to start again at Setup, or to end the run for the day.

In summary, the hybrid computer program is a sophisticated system, with heavy demands on it for easy changes, and flexible control to meet the needs of experimentation. Without software assistance, the hybrid programmer faces an enormously complex planning and programming task.

Software system specifications

The hybrid simulation has now been described; the simulators, the computers, the users, and the program. We have reached the conclusion that a software system is required to assist the user to create a valid physical simulation. The next step is to determine the specifications for such a software system. In order to make this determination we will review the several steps required to pro-

gram, operate and debug a typical hybrid simulation.

Programming steps

The first step is the planning phase. The experimenter must decide (1) which equations he will use to simulate his particular vehicle, (2) which portions of the equations will be solved on the digital computer, (3) how to divide the digital computer program into independent computational loops, and (4) what types of external control (interrupt, pushbutton, etc.) will be needed in the digital program.

The second step is for the simulation programmer to write his digital program. This step includes two separate problems. Problem one is to create a physically valid simulation. To accomplish this he must create a program which will compute and integrate the differential equations which describe the physical system. It must also compute the values of arbitrary functions and record data acquired from various sections of the simulation. Problem two is one of organization and control. He must consider mode control, linkage input-output, computations to perform in each mode, and utility functions needed.

The third step is to have the program key-punched, compiled and loaded.

The fourth step is to debug the digital program. The debugging includes making program patches, changing data (usually in decimal), setting breakpoints, and printing memory dumps.

The fifth step is the debugging of the hybrid program. Here the hybrid user wants to set potentiometers, run a static test, and readout and set both analog components and digital variables.

The sixth and final step is the execution of the debugged program. The experimenter is now ready to take data. However, he is continually modifying his program in order to obtain a "better" result, even after he determines that the computer is correctly solving the equations which describe the system which he is simulating.

Specifications

It is clear that the standard digital software system does not give the hybrid user assistance in many of the tasks described above. The software system must not only provide the hybrid user with a standard digital monitor, but also with a means of easily and conveniently communicating with and controlling his simulation computer program. The software system should assist the

user to organize his program, and lead him step-by-step through the thought processes he must make to achieve a completed program. In order to meet these requirements, the system should include the following items:

- 1) Problem-oriented languages which allow him to express the basic equations and control functions of his problem in terms familiar to him. FORTRAN, HSL, and other simulation languages are good examples. A language designed to meet a special need, e.g., a function generation processor, is another example.
- 2) Hybrid executive functions which are called by the simulation programmer through the problem-oriented language. These calls should be made during program setup, when time is not critical. The executive must optimize the simulation program. Operate mode in such a way that overhead is constrained to a minimum. The hybrid software should handle all multi-rate scheduling, all AD/DA transfers, mode control, interrupt scheduling, and schedule user utility functions.
- 3) Provisions to allow the simulation to be set-up and static tested under the control of a punched card setup deck, or similar medium. Furthermore, the system should allow reading out the state of variables at the end of a run, on punched cards or other medium, so that these can be re-entered the next day as new initial conditions. Convenient means of symbolically addressing and changing both analog and digital variables are essential.
- 4) Subsystems such as real-time numerical integration, arbitrary function generation, data history recording, and postrun data processing and analysis.

We have reviewed the steps which a hybrid user takes in creating a hybrid simulation and from this determined a set of functional specifications which a hybrid software system must meet. Now we will describe FAMILY I and see how it meets these specifications.

FAMILY I

The operating system software at NASA-Ames evolved to its present state under the influence of the needs of the experimenter. The implementation takes advantage of specific hardware fea-

tures of the EAI 8400 computer, but the design is computer independent. FAMILY I consists of several modules designed to give the experimenter the assistance he requires to create his simulation. These are:

1. DAD—Developmental Ames Digital Monitor.
2. MOTHER—Monitor Time Handling Executive Routine.
3. HYBRID CASPRE—Comprehensive Aid to Simulation Programmers and Engineers.
4. RNTNTN—Real Time Integration System.
5. RUNDUM—Simulation Run-Time Dump.

MOTHER and HYBRID CASPRE are the two major modules of FAMILY I which provide a significant advance toward the goal of easing the experimenter's burden in preparing and operating a hybrid simulation program. For this reason they will be described in considerable detail.

MOTHER

The original purpose in developing MOTHER was to provide a real-time scheduling solution to an increasingly common problem in simulation programming: multi-rate computation. The complexity of certain physical systems does not permit sufficient time for the average digital computer to perform all the necessary calculations required to simulate that system. However, since all calculations need not occur at the same frequency, the simulation can be divided into several sections (processes) which execute at different frequencies. This multi-rate computation, like multi-programming, introduces difficult scheduling problems.

MOTHER evolved to provide much more than just scheduling processes in multi-rate simulations (Figure 3). MOTHER provides all of the following:

1. Synchronization between the various hardware sections of a hybrid simulation, even for multi-rate computations.
2. Hybrid computer mode control (IC, Hold and Operate).
3. General program control.
4. Re-entrancy.

The remainder of this section will describe the methods by which MOTHER provides the above functions.

Design criteria

In designing a real-time scheduler, several im-

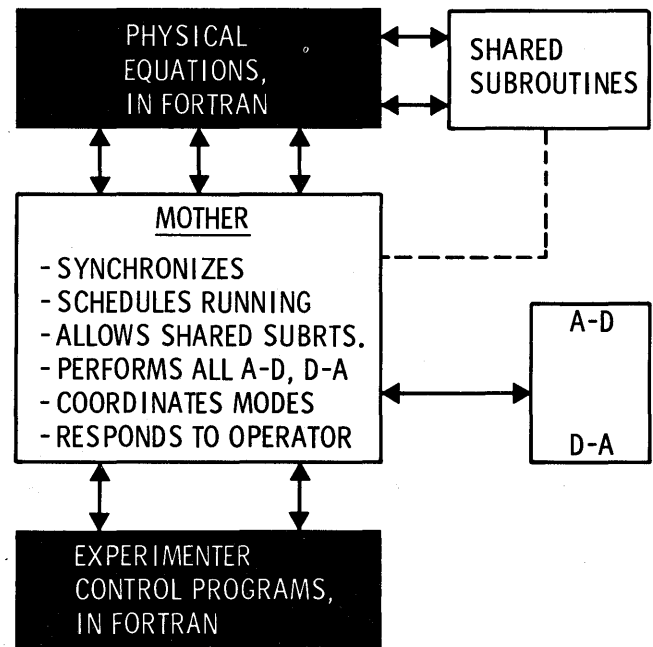


FIGURE 3—MOTHER provides execution scheduling, program control and real time I/O

portant design criteria must be considered.

First is the need to schedule two types of real-time processes. In one type, computations must be completed within repetitive time spans; for example, once every 20 milliseconds. These processes are said to be *constrained* with respect to time, and can be interrupted provided that they are completed before the end of their allotted time. In the second type of real-time process, the processes must be scheduled to occur at precise instants of time for synchronization. Analog-to-digital and digital-to-analog transfers are examples. These processes are said to be *synchronized* with respect to time, and cannot be interrupted. The constrained processes perform computations, the results of which are used for output to the analog world by the synchronized processes.

A second criteria of a real-time scheduler is that it uses an absolute minimum of time to perform its tasks. This is not a new concept and need not be belabored.

A third criteria is the need to find a solution to the re-entrancy problem. A multi-rate simulation is a special case of multi-programming in that the scheduler may suspend (interrupt) one set of calculations and begin execution of another set in order to maintain the schedule. Because it is quite possible that both sets of calculations share library subroutines, we are faced with the re-entrancy problem.

The fourth criteria is to make the hybrid executive as easy to use as possible. As will be seen, this criteria was responsible for the evolution of the real-time scheduler to a complete simulation executive system.

Scheduling

When discussing the timing aspects of scheduling, the terms *frame time* and *period* must be defined. A process that must execute every n milliseconds has a frame time of n milliseconds. A simulation program running in real time has a period which is defined as the least number which all frame times will integrally divide. For example, a simulation with processes having frame times of 20 and 30 milliseconds has a period of 60 milliseconds.

Priority

The MOTHER scheduler causes a process to be executed when it is *ready* and has *top priority*. A process is ready if it meets all of its "wake up criteria": for example, the process must be in memory, it must be time to execute it, and any external conditions pertaining to the process must be satisfied.

Because there are two types of processes, constrained and synchronized, there are two levels of priority. Processes at the synchronized level have priority over processes at the constrained level. The priority of synchronized processes, if more than one is ready, is determined by the order in which they were defined to the scheduler. The priority of constrained processes, if more than one is ready, is determined by completion time constraints; the top-priority constrained process is defined as the one which must finish first.

Synchronized process priority

The rigid synchronizing requirements of AD-DA input-output, and a desire to overlap this input-output with the housekeeping functions of the executive, led to the conclusion that MOTHER should generate linkage input-output routines. Having linkage I/O custom-made to the user's requirements and internal to MOTHER permits time efficiencies in executing and scheduling these routines. The programmer describes his linkage requirements during MOTHER setup, giving starting channels, number of channels, data buffers, and frame times. (See QMSAD, QMSDA and QMSCOM calls of Figure 4.)

```

C .....
C TYPICAL FORTRAN IV SETUP PROGRAM FOR MOTHER
C .....
C     EXTERNAL  ROT, TRANS, INTS1, INTS2, ICROU, HLDROU, OPROU
C     EXTERNAL  ENGOUT, SETUP
C .....
C     COMMON/SIM/ROTBUFF(16), TRABUFF(16), ROTDAB(16), TRDBUFF(16), EFRTBL
C .....
C     FRAME TIMES, START CHANNELS, NUMBER OF CHANNELS
C .....
C     ROTTIM = 20
C     TRTIM = 30
C .....
C     RADCH = 4
C     RADNUM = 6
C     TRADCH = 10
C     TADNUM = 4
C .....
C     RDACH = 8
C     RDANUM = 4
C     TRDACH = 4
C     TDANUM = 3
C .....
C     AD AND DA PROCESS LISTS
C .....
C     CALL QMSADI2, ROTAD, ROTTIM, RADCH, RADNUM, ROTBUF, TRNAD,
C     ITRTIM, TRADCH, TADNUM, TRABUFF
C .....
C     CALL QMSDAI2, ROTDA, ROTTIM, RDACH, RDANUM, ROTDAB, TRNDA,
C     ITRTIM, TRADCH, TDANUM, TRDBUFF
C .....
C     DEFINE COMPUTATIONAL PROCESSES
C .....
C     CALL QMSCOMI2, ROT, ROTTIM, TRANS, TRTIM)
C .....
C     DEFINE OPERATE, HOLD, AND IC PROCESSES
C .....
C     CALL QMSOP(6, ROTAD, ROTDA, TRNAD, TRNDA, ROT, TRANS)
C .....
C     CALL QMSHD(0)
C .....
C     CALL QMSIC(4, ROTAD, ROT, TRANS, ROTDA)
C .....
C     DEFINE INTERRUPT PROCESSES, FRAME TIMES AND INTERRUPT NUMBERS
C .....
C     CALL QMSINTI2, INTS1, ROTTIM, 14, INTS2, TRTIM, 11)
C .....
C     DEFINE MODE CONTROL AND ASSOCIATE CONTROL ROUTINES WITH SWITCHES
C .....
C     CALL QMSXMO(ICROU, 7, HLDROU, 4, OPROU, 5, EFRTBL)
C     CALL QMSEFRIBIT1, 0, ENGOUT, 0, 0)
C     CALL QMSEFRIBITS, 1, SETUP, 1, 1)
C .....
C     END SETUP PROGRAM, AND ENTER MOTHER TO START SIMULATION
C .....
C     CALL QMSEND
C     END

```

FIGURE 4—Typical FORTRAN IV setup program for MOTHER.

The priority of synchronized processes is best described through an examination of Figure 5. The linkage I/O processes, ROT I/O and TRANS I/O, are synchronized processes and must execute at, or as near as possible to, precise instants of time. Note that in every scheduling case these synchronized processes have priority over the constrained processes, ROT and TRANS. In fact, at times 20, 30, and 40, constrained processes are interrupted to perform a synchronized process. It so happens that in Figure 5 both types of processes have the same frame times. However, this is not a scheduling requirement. For example, if the synchronized processes had 15 millisecond frame times, the schedule would be: only constrained processes ROT and TRANS at time 20 and 40; only synchronized processes ROT I/O and

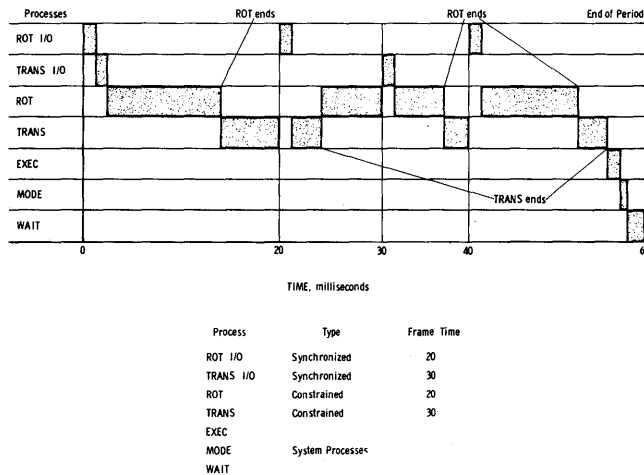


FIGURE 5—MOTHER multi-rate scheduling of typical processes

TRANS I/O at time 15 and 45; and both at times 0 and 30.

Constrained process priority

The top-priority constrained process is the one that must finish first. To understand the definition, consider the situation of Figure 5 at time 20. Suppose that the highest priority process was the one with the shortest frame time; then the process ROT would begin execution at time 20. Using this definition, the schedule cannot be maintained since ROT will execute until about time 32 and TRANS will not be able to complete by time 30. However, if TRANS is allowed to continue at time 20, as in Figure 5, it completes before time 30 and the schedule is kept. Therefore, defining the top-priority constrained process to be the one that must finish first permits more effective use of time, and allows more flexibility in choosing frame times in the simulation.

A clock frame, the time interval between timer interrupts, is varied during the period to cause interrupts for two purposes. The first is to execute synchronized processes (linkage input-output). The second is to act as scheduling milestones for constrained processes. In Figure 5, milestone interrupts associated with the ROT process occur at time 0, 20, and 40. From a timing standpoint, these mean that ROT is now ready to execute. Of equal importance, each means the previous execution of ROT must by then be completed, or else the schedule has failed.

Fixed time lists

The occurrence of the timer interrupt may mean that the priority of processes has changed, since processes may become ready which have to complete execution prior to the process which is presently executing.

The priorities of processes for a given clock frame can be represented by a fixed list. For example, the list derived from Figure 5 for time 0 to 20 contains, in order of priority, the processes ROT I/O; TRANS I/O, ROT, TRANS, WAIT.* For time 20 to 30, the list contains ROT I/O, TRANS, ROT, WAIT.

These lists give the order of execution of processes for the clock frame. If some process on the list is not ready when its turn comes, it is passed over. For example in Figure 5, had TRANS already completed execution by time 20, then ROT would execute first because TRANS is not ready again until time 30. Another example would be an interrupt process that becomes ready only when an external signal is received. In this manner asynchronous events are scheduled in a routine manner; they are placed on lists, but are passed over until short interrupt-service routines, activated by interrupts, mark the processes ready. These interrupt processes are defined to MOTHER during setup. (See QMSINT call of Figure 4.)

These considerations yield the conclusion that real-time scheduling can be accomplished from a set of fixed lists formed during setup when time is not critical. (See QMSOP call of Figure 4.) Each list contains, in order of priority, all processes that are candidates for execution during a given clock frame. The number of lists depends on the number of clock frames needed during a period, which is related to the choice of frame times. There would be four such lists (see Figure 6) associated with the example represented by Figure 5. The same lists are repeated for each successive period.

Whenever the timer interrupts, the scheduler moves to the list corresponding to the next clock frame. The scheduler begins to execute, in order, all processes on the list that are ready. If all processes on the list are finished before the clock frame ends, the system routine, WAIT, is scheduled.

Some processes on a list may not get a chance to execute prior to the next timer interrupt. Thus,

*EXEC and MODE are system processes scheduled at the end of a period. WAIT is a delay process representing unused time that could be given to a background program in a multi-programming environment.

Clock Frame	Fixed Time List
0 - 20	ROT I/O TRANS I/O ROT TRANS WAIT
20 - 30	ROT I/O TRANS ROT WAIT
30 - 40	TRANS I/O ROT TRANS WAIT
40 - 60	ROT I/O ROT TRANS EXEC MODE WAIT

FIGURE 6—Fixed time lists for scheduling example

associated with each timer interrupt is a check to see if the schedule is being maintained. If there is not enough time to execute all of the processes that must finish before the next interrupt, the schedule cannot be met; this means that frame times must be changed. The system's responsibility is to maintain the schedule given and if this is not possible, inform the user. For example, in Figure 6, ROT must complete during the clock frame 30—40, but TRANS does not have the same restriction.

One criteria of a real-time scheduler is that the scheduler itself should consume a minimum of time. Alternatives to a fixed time list scheme imply computing dynamically the next process to be scheduled. This technique would involve additional overhead, which might or might not be acceptable, depending both on the particular simulation programmed on the computer and the particular computer.

The advantage of the fixed time list technique is that they are built as part of the program setup. Since time is not critical at this point, the routine that builds lists can take as long as necessary. When the program eventually runs in real time, all scheduling decisions, except whether a process is ready, have been made outside of real time and are embodied in lists. The disadvantage of the list technique lies in the amount of memory space needed to store the lists, which can be considerable if the choice of frame times forces a long period, or if a large number of processes appears on each list.

A functional flow chart of the MOTHER scheduler is shown in Figure 7.

Re-entrancy

The next problem attacked was that of shared subroutines. As we saw it, there were three possibilities.

The first was simply not to permit shared subroutines. If more than one computational process requires the same subroutine, then each process is given its private copy. This solution causes as many problems as it solves. In addition to wasting

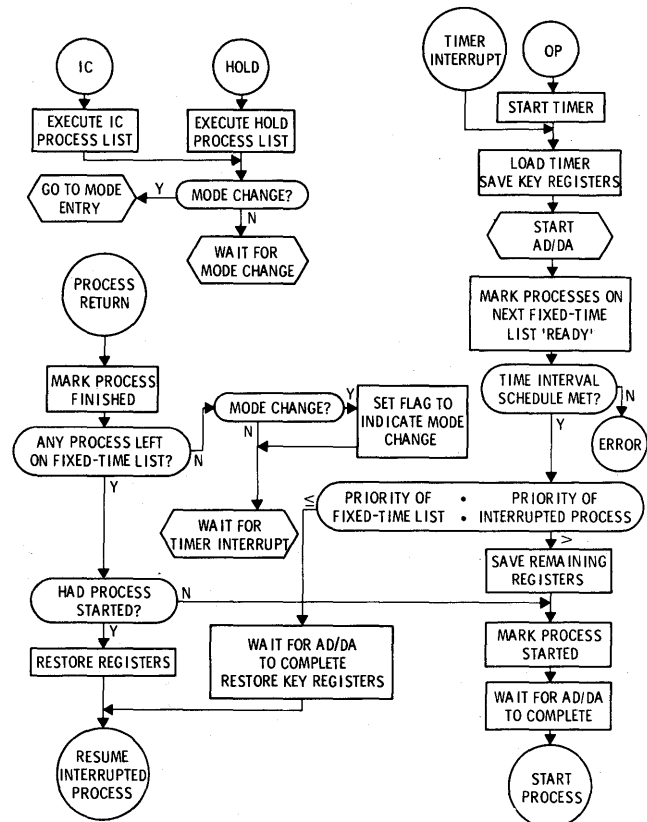


FIGURE 7—MOTHER scheduling

memory, there is the problem of forcing the loader to load the same subroutine more than once.

The second possibility was to supply re-entrant subroutines. The classic difficulty is the high overhead normally associated with a re-entrant system. Also, we would have had to rewrite all subroutines to make them reentrant.

The third possibility was to declare shared subroutines non-interruptable so that a subroutine cannot be entered by one process before the previous process completes the subroutine. The problem here is that the execution time of the shared subroutine must be kept short compared with the clock frame, or else the schedule may fail.

We decided that the third possibility was the most desirable, in terms of keeping the overhead of the system at a minimum, and implemented the following scheme:

Timer interrupts are permitted to occur as scheduled. Prior to executing the first scheduled constrained process (i.e., after executing synchronized processes), a test is made to see if the timer interrupt occurred while running a shared subroutine. If so, it is permitted to complete, at which time a forced interrupt is triggered,* and MOTHER resumes the Fixed Time List schedule.

It proved necessary in practice to provide re-entrant routines for functions which require a fairly large amount of computational time, such as numerical integration of system equations. For these cases, the programmer must provide the address of a stack, for temporary storage, through his calling sequence.

MOTHER was eventually extended to include a system to provide re-entrancy for library subroutines. A different stack pointer is included as a part of each Fixed Time List. The size of each stack is equal to the largest number of temporary locations which any of the standard library re-entrant subroutines might use; presently 16 locations are adequate. Since each timer interrupt moves the scheduler to a new Fixed Time List, there are no stack conflicts. Whenever a re-entrant routine is resumed after it has been interrupted, the stack pointer is automatically restored when the computer status is restored. This system has very low overhead, thus overcoming one major objection to reentrancy in simulation programs.

*One bit in each word of the EAI 8400 computer can be set, checked and made to force an interrupt, upon instruction execution, which made the implementation of this technique easy and the execution time minimal.

Mode control

In order to schedule and control mode changes properly, set mode flags, send synchronizing signals throughout the hybrid system, and make MOTHER-generated A-D/D-A routines available in IC mode, it was decided that mode control must be part of MOTHER.

The processes scheduled to run in IC or Hold modes are put on IC or Hold lists similar to the Fixed Time Lists. (See QMSHD and QMSIC calls of Figure 4.) Processes on these lists are executed repeatedly, on a round-robin basis. Mode changes are effected by scheduling a system process, MODE, during the last list of a period.**

The user indicates a mode change by generating a discrete signal via a pushbutton, an interrupt, or a program flag setting. When MODE executes it detects the request and takes the actions necessary to make the mode change effective at the start of the next period.

The use of MODE required conventions by which the programmer would tell MOTHER (1) which interrupt lines, if any, he wishes to employ to effect mode changes, (2) which memory flags would be set to indicate a mode change, and (3) which process must be executed when mode is changed. (See QMSXMO call of Figure 4.) User processes accomplish any special action associated with changing modes, such as simultaneously putting the entire hybrid system in the new mode.

Utility functions written by experimenters

All of the user's simulation control problems are still not solved. There is a class of utility functions that the programmer may want to execute as needed, such as special printouts.

A considerable amount of control logic is associated with executing utility routines. Some routines should not be executed if the simulation is in Operate mode. Other routines require putting the simulation into one mode prior to execution, and into some other mode after execution. There is also the problem of how to schedule these utility functions.

Hence a system executive control process, called EXEC, was incorporated into MOTHER. EXEC is scheduled just prior to the system MODE process. Its function is to detect requests for utility functions from bits in an Executive Flag Register

**In IC and Hold modes, there is no period, and MODE is scheduled at the end of the round-robin.

(EFR). The EFR is the register of communication between the user and EXEC. Each bit in this register is associated with a particular utility function through FORTRAN calls during MOTHER setup. (See QMSFR calls of Figure 4.) When a bit goes high, the associated function is executed by EXEC. Bits are set in the EFR either under program control or by a pushbutton register on the user's console.

Working in conjunction with MODE, EXEC is able to change mode, execute the utility function, and reestablish a desired mode, in the following manner:

The scheduler executes EXEC, which requests the specified mode change by setting a flag, and returns to the scheduler. MODE, scheduled next, performs the desired mode change and returns to the scheduler. Sometime later, after working its way through its lists, the scheduler executes EXEC again. EXEC, noting that it is processing a previous request, executes the required utility function and, following that, may request another mode change, according to the programmer's wishes. MODE, scheduled next, makes the mode change, thus completing the utility function by entering the mode required by the user.

MOTHER—Summary

MOTHER is a complete hybrid executive system. It solves the scheduling and control problems associated with a complex hybrid simulation program and, of equal importance, helps the user write his program by requiring him to provide, step-by-step, the information needed by MOTHER. (See Figure 4 for a complete MOTHER setup example.) This invariably leads to a complete simulation program, with no considerations omitted. In summary, the planning and drudgery of hybrid programming are placed where they belong, on the software system.

HYBRID CASPRE

In hybrid simulation there can be a wide time span between program preparation and a running program. Excessive programming and checkout time are normal; a simulation in its developmental stages requires extensive debugging and due to its experimental nature, is never really finished. Equations are changed and data modified. New control features are added. Daily setup is required. HYBRID CASPRE provides convenient interactive control for these functions.

Analog setup

Analog computer setup is considerably aided and the setup time decreased through the use of the analog module of HYBRID CASPRE. It permits the programmer to set, static test, and readout any component on the analog computer through control directives from the digital computer's input devices. For example, to set potentiometer 001 to 0.3452, we would keypunch on a card, (or type)

—C001 = .3452

or to set DAC 16 to —90.45 volts, we would keypunch a card

—0016 = —90.45

A typical analog setup program consists of a deck of punched cards giving potentiometer settings and amplifier readings. HYBRID CASPRE allows the programmer to use the same punched card deck to perform set, static test, and readout operations. For example, when requested to set analog components to given values, HYBRID CASPRE ignores all references in the punched card deck to components that cannot be set, such as amplifiers. On the other hand, when requested to do a static test operation, HYBRID CASPRE uses values on the cards that apply to comparisons rather than to settings. And, when requested to read-out analog components, HYBRID CASPRE ignores all values on cards and concerns itself only with the component names punched in the deck, reading-out the current value of each, and recording it for the next day's use.

As setup proceeds, all discrepancies, such as pots that set incorrectly and static tests that fail, are identified, by component, on the line printer. When setting pots and static testing analog components, HYBRID CASPRE accepts readings that fall within tolerance standards which the user may change if he desires.

Digital setup

The basic HYBRID CASPRE digital features are to readout or modify the contents of a digital memory word. A memory word can contain, among other things, an instruction, a number, or a string of alphanumeric characters. On the EAI 8400, binary numbers can represent octal, 16 or 32-bit fixed-point decimal, or single or double-precision floating-point decimal, numbers.

Thus, within HYBRID CASPRE, there are various format modes which allow the user to read or modify a memory cell as an instruction, a number, or a string of characters. There is mnemonic mode, which is an on-line assembler/disassembler of machine language instructions; there is octal mode; there is fixed-point mode; there is single-precision floating-point mode; and there is BCI input-output mode for character strings. For example, if the user selects floating-point mode, he then reads and modifies memory cells as if they were decimal numbers. These various modes free the user from having to learn the internal binary formats of numbers and instructions.

HYBRID CASPRE digital features are used: to *setup* the digital computer by making known corrections and reading initial data, to *debug* the program on-line, and to *modify* the value of program variables as a part of the trial-and-error process of working with the simulation.

Symbolic addressing

Every memory cell in a program has a symbolic address. This symbolic address is either a statement number, a variable name, or a relative address consisting of the program name plus an increment. In the FAMILY I system these symbols are generated by the FORTRAN compiler and loaded into memory with the program, where they are available to HYBRID CASPRE.

Symbolic addressing is an extremely useful feature to experimenters. Without it, the user must determine the octal address of each memory word he wishes to read or modify. This requires finding the relocatable program address on the load map, looking up the relative memory word address in his listings, and adding the two numbers together, in octal. The process is time consuming, subject to error, and not the type of task an experimenter should need to do. In contrast, HYBRID CASPRE takes over all the work, allowing the user simply to type a variable name, or at most, a program name plus a relative address. For example, suppose we want to change the value of GAMMA from 0.5254 to 0.05254. We would type: (CASPRE response is underlined)

```
+GAMMA = .5254 E 00 .05254$ 0.5254 E 01
```

Also of particular importance is the HYBRID CASPRE ability to readout or modify from punched cards. Thus in the leisure of an off-line situation, the experimenter can prepare new sets

of experimental data, new equations, and program corrections, all in symbolic form. When he returns to run his program, he simply reads his symbolic correction deck with HYBRID CASPRE, and his program is updated.

In addition to reading-out memory cells one at a time on the typewriter, HYBRID CASPRE permits dumping blocks of memory on the line printer. All of the format modes and addressing options mentioned above are available with the dump. Accompanying each dump are the values of all hardware registers as they existed when HYBRID CASPRE took control.

A debug function of HYBRID CASPRE is the breakpoint. It provides the ability to interrupt the flow of a program at a predetermined point, examine data, make corrections, and then resume at the point of interruption. The user can set more than one breakpoint at a time, and he can set a breakpoint in such a way that it will not stop the program until a specified number of repetitions are completed.

Other CASPRE functions allow the user to search a block of memory for a given bit pattern, or to set a block of memory to any desired value.

Summary of FAMILY I

FAMILY I is a complete operating system for hybrid computers, designed specifically for interactive use by research experimenters. MOTHER is an extension to the monitor for real-time simulation. It performs certain monitor functions, such as scheduling real-time processes and performing linkage input-output, and provides executive functions, such as computer mode changes. HYBRID CASPRE is an interactive operating and debugging system. It permits symbolic setting and readout of any analog or digital component of the hybrid computer as well as symbolic patching of the digital program.

The other members of FAMILY are also necessary for successful hybrid simulation. DAD, the digital monitor, controls batch language processing, loading and digital IO. It was developed to open hybrid simulation to experimenters who have little digital computer experience. It does this by allowing an experimenter to compile and load his program easily, *save* and later *input* his personal programs on his own magnetic tape, for his own use at any time, and keep his Fortran programs on magnetic tape and change them during compilation. RNTNTN, a reentrant real-time integration routine, was developed to provide sev-

eral numerical integration algorithms for simulation, on-line selection of integration algorithms, and integration of *all* state variables using only one Fortran call. RUNDUM, a data acquisition routine, was developed to record "movies" of a hybrid run on magnetic tape suitable either for display after the run, on a CRT or a strip chart recorder, or for obtaining a listing on the printer.

Each member of FAMILY makes its own contribution to solving the problems of hybrid simulation as can be seen in the Family Portrait, Figure 8.

CONCLUSIONS

FAMILY I is a convenient system for the research experimenter to use in operating his simulation. It provides on-line interaction with, and control of, hybrid simulations. However, this serves to aggravate the old problem of efficient utilization of computer time. The only way to improve utilization is to provide multi-programming of the digital computer, which introduces other problems. For example, the analog system of User A must be protected against an inadvertent transmission from User B. Also, User A must not be permitted to use excessive time during any clock frame or User B will not have sufficient time to complete his computations within that clock frame. It is imperative that the software system

protect one user against the consequences of the actions of other users.

The next step in the evolution of hybrid software at Ames will be the development of FAMILY II, a dual-programming version of FAMILY I. The Fixed Time List technique employed by MOTHER will be applicable for dual-programming scheduling of real-time processes, since there is no correlation of a process on a fixed time list with any particular user. However, dual simulations probably must be adjusted in frame times to run together, and dual mode control may present some challenge.

The first version of FAMILY II will permit only one user to be in operate mode at a time. Later we hope to be able to lift this restriction. FAMILY II will emphasize interactive operation by research experimenters, so that each experimenter can develop his simulation, run and control it, make changes and experiment with it, all at a problem-oriented language level, regardless of the sophistication of the environment being controlled by the computer system.

REFERENCES

- 1 M P BURWEN et al
Preliminary design plan for flight
NASO 67-9 Ames Research Center NASA internal document
December 15 1967
- 2 M S FINEBERG O SERLIN
Multiprogramming for hybrid computation
AFIPS Conference Proceedings Vol 31 1967
- 3 B W LAMPSON
Scheduling and protection in interactive multi-processor systems
Document No 40 10 150 Advanced Research Projects Agency
January 20 1967
- 4 User's Guide for EAI 8400/8900 Basic Monitor Operating System PPN 07 825 0060 2 Version 2
- 5 E A JACOBY D E ROBINSON
A hybrid debugging operation system
Ames Research Center NASA internal document November 30 1967
- 6 F J CORBATO V A VYSSOTSKY
Introduction and overview of the multics system
AFIPS Conference Proceedings Vol 27 1965
- 7 E A JACOBY J S RABY D E ROBINSON
Monitor time handling and executive routine
Ames Research Center NASA internal document May 1968

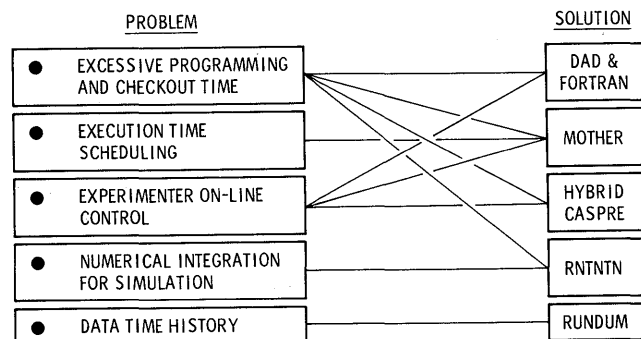


FIGURE 8—A family portrait—The solution to the problem of hybrid programming

Stand-alone /remote graphic system

by MICHAEL D. RAPKIN and OTHMAN M. ABU-GHEIDA

IBM Corporation
Kingston, New York

The big *revolution* in computer usage is, by now, an old story. No longer do users, programmers and computer operators always work independently and separately, communicating only through voluminous printouts. Engineers and designers can now participate continuously in the execution of their problems through displays. By accelerating design iterations, they can solve their problems more quickly and, at the same time, improve the quality of their solutions.

The *evolution* in computer usage—the expansion of graphic and other display capabilities through an ever-broadening spectrum of devices—is less well known, however, and is continuing. These devices are becoming both more and less sophisticated. The large user with a large system at his disposal can now do more things. But probably even more important, the small user has not been forgotten.

This paper describes a system* developed to meet the needs of two particular classes of users—the “stand alone user and the remote user.” (Figure 1)

In the remote display configuration, the effect of the substantially lower data transmission rates available with telecommunication facilities, as compared with those provided by a CPU channel, had to be compensated for in order to minimize any degradation in system response to operator actions. For this reason, it was necessary that specific functions, preferably those characterized as high-usage or conversational, be capable of being performed within the display subsystem. This defined a need for local data storage, I/O capabilities, decision making, and data processing capabilities within the subsystem. These capabilities were also required of a low-cost, stand-

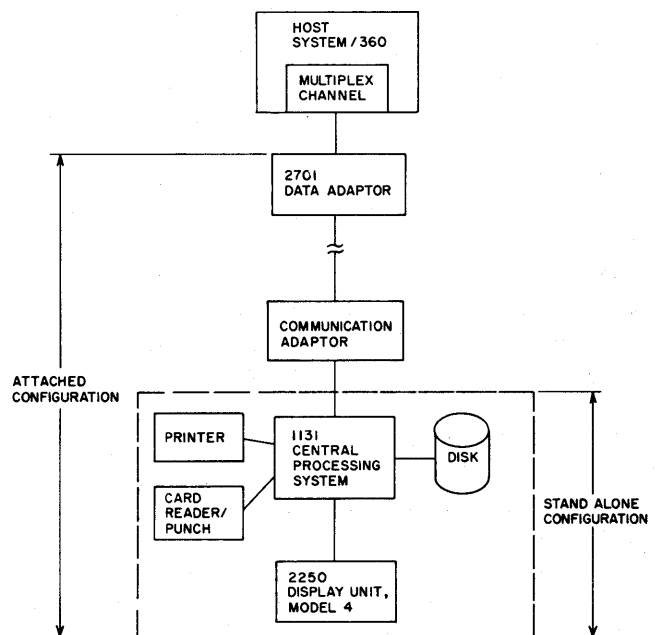


FIGURE 1—Stand-alone and remote IBM 1130/2250-4 configurations

alone system. Further study indicated that the key system design criteria of the two configurations were also similar—particularly the following:

- Regeneration of the display from the CPU core, for ease of programming and rapid display unit control.
- Direct attachment of the display to the CPU without the use of an intermediary channel for efficient interrupt-handling.
- Facilities within the display-CPU interface to make efficient use of core and to minimize the number of interrupts generated by the display.

*IBM 1130/2250 Graphic Data Processing System

- Fast, convenient, auxiliary-data storage.
- Programming support for graphic functions—i.e., all high-activity conversational operations.
- FORTRAN language support.
- Non-programmer interface for display user.

The result was a low-cost, dual-purpose configuration for both types of users.

General description

The display is regenerated from computer core storage by cycle-stealing; i.e., once the display has been started by an I/O command, it operates asynchronously with the CPU program and other I/O devices. The display is designed not to interfere with the operations of other I/O devices attached to the computer system. In general, these devices will have very little effect on the performance of the display. The computer system is not changed in any way to accommodate the display. Thus, when the system is not being used for a graphic application, the computer can be used for other data processing functions. The remote configuration can be operated at two speeds: from 1200 baud to 2400 baud, and from 19.2K baud to 230.4K baud (2.5 to 30K characters per second).

The 1130/2250 is a self-sufficient graphic system. Its basic programming support, the support for the stand-alone configuration, is also the basic support for the remote configuration. In remote configurations, the system will handle the high-usage conversational and image-generation functions in an application, using the central System/360 for computing functions and access to large, central data bases.

The central processing unit of the satellite computer is a compact, desk-size binary computer. The system features a core storage capacity of up to 32K 16-bit words, core speeds of 3.6 or 2.2 μ sec per word, a built-in disk file of 512,000 words, up to four additional disk files, card and paper tape I/O, a plotter, and line printers.

The display is composed of the display CRT and an interface for attachment to the controlling CPU. The display incorporates a 21-inch CRT (having a 12-inch by 12-inch usable area) and a program-controllable, fibre-optic light-pen with a pressure-operated tip switch. Optional features include an alphanumeric keyboard and a 32-key, programmable function keyboard. (Figure 2)



FIGURE 2—1130/2250-4 system

Some of the important performance characteristics of the display are:

- A total of 3,848 character positions: 52 lines with 74 characters per line.
- 1024×1024 addressable positions
- Up to 2000 characters or 2800 incremental vectors generated at a 40-cps regeneration rate; up to 2600 characters or 3700 incremental vectors generated at 30 cps.
- Absolute addressing—the capability to generate straight lines between any pair of the 1024×1024 positions on the display screen.
- Incremental addressing—the capability to position draw increments ranging between -64 and $+64$ raster units in X and Y from the current position.

Character generation is a programmable function allowing the user flexibility in the generation and use of character sets. Upper- and lower-case alphabetic characters can be generated on the display screen through the use of the alphanumeric keyboard, or directly as part of a display program. Other character generation facilities provide extensive editing capabilities—the operator can overwrite, subscript or superscript a character, or overwrite a whole line of characters.

The display (Figure 3) attaches to the CPU via a storage access channel. Core storage is both space- and time-shared. Display commands and orders are stored in core storage, and are decoded and executed in the display interface. The dis-

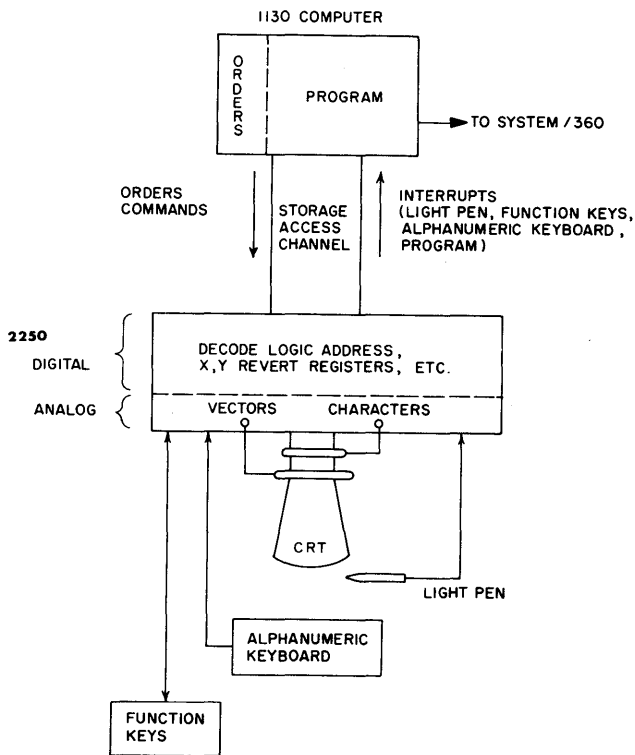


FIGURE 3—1130/2250-4 system organization

play program (the set of display orders) therefore shares core storage with the CPU program, resulting in direct control over the buffer program, fast up-dating of the display image, and efficient graphic-system programming. Once the display has been started by an I/O command, display orders are accessed from core by stealing core memory cycles; i.e., the display operates independently of the CPU program, and both can be running simultaneously.

The display interface consists of a Memory Address Register, a Data Register for temporary data buffering, a Revert Register used in display image subrouting, and decoding logic. The display CRT has two deflection systems: the main deflection system for generating vectors, and a character deflection system for generating characters. CPU interrupts are caused by light-pen detects, depression of keys on either the alphanumeric or program-function keyboards, or by the display program orders. When an interrupt occurs, all interrupt data are read into CPU core by the execution of a single display command.

Cycle Stealing and Interference

Attachment of the 2250 Model 4 to 1130 system via the storage access channel permits the op-

eration of the display asynchronously with the CPU. That is, once the 2250 has been started, it continues to execute display orders, similar to the operation of a channel, by stealing core storage cycles without CPU program intervention. The portion of the core having the display orders becomes a buffer for the display. These orders are accessed through the storage access channel and sent to the display up to 40 times per second. Since the display, I/O devices, and CPU are requesting core-storage cycles from a single source, some delay must occur. However, the design of the storage-access channel and the display interface prevents any significant interference with other I/O devices.

The following are some characteristics of cycle-stealing:

- The lowest-priority for cycle-stealing is the CPU. Some of the I/O devices require interrupt service within a specified length of time. To ensure that they obtain this service, the display is inhibited from cycle-stealing when interrupt service is required for these devices.
- When the display is drawing vectors, it steals one or two cycles to access the data related to the position of the vectors. The display will not cycle-steal during the time that a vector is being drawn.
- The maximum interference from cycle stealing occurs when the display is generating characters.
- The maximum interference with display operation occurs when the CPU has to continuously process interrupts for time-dependent devices. This interference will not normally be observed on the display.

The display interface

Because it was essential that the CPU be an effective processor of data in both system configurations, a display interface was included which reduces core requirements for the display, and handles interrupts quickly and efficiently.

A display program executed by this interface consists of orders and data. Orders either define the display operation or establish its "Mode." Order-defined operations include vector and point-plotting, branching, and CPU-interrupt generation. Three orders establish modes: Set Graphic, Set Character, and Set Light-Pen. The display is always in either Graphic or Character Mode, and in one of four pen modes.

Display Orders

Regeneration of the display program (Figure 4)

A combination of a Start Timer order at the beginning of the display program and a BRANCH order at the end of the program provides the regeneration cycle and ensures that the regeneration rate is no greater than 40 cycles per second.

There is no lower limit placed on the regeneration rate. However, display images regenerated at rates below 30 cycles per second will tend to "flicker."

Graphic mode

Either vector or point operations can be performed in Graphic Mode. In this mode, the display can receive, from the display program, either beam positioning or drawing orders, or an order to change mode. There are three basic beam-positioning orders which can be executed in Graphic Mode:

- Absolute positioning to any point (X, Y) on the 1024×1024 grid, with beam either on (to draw a vector or point) or off (to position the beam).
- Absolute positioning to any point, X or Y, with beam on or off. This order moves the beam vertically or horizontally, and minimizes the use of CPU core in display images with a predominance of vertical and horizontal lines (Figure 5).

In generating the 10×7 grid, only 31 words of core are required, whereas 62 words would have been required using the normal

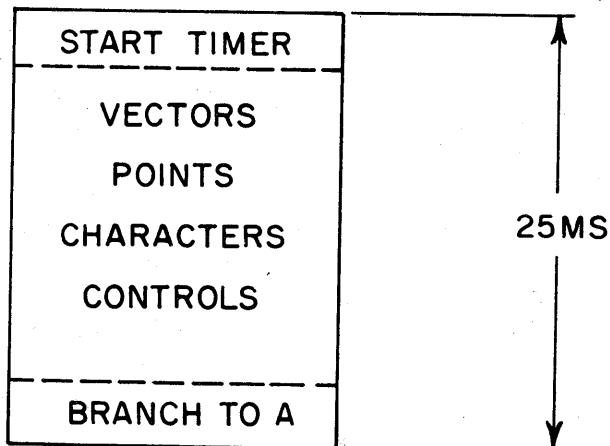


FIGURE 4—Display regeneration

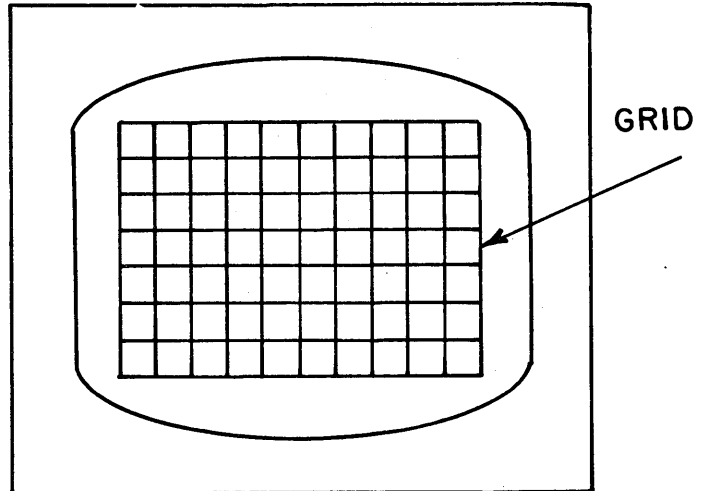


FIGURE 5—Grid drawn with axial vectors

two-word, absolute-vector format. This two-to-one savings in core storage will also apply in the generation of bar charts, wiring diagrams, and integrated circuit layouts.

- Relative positioning with increments Δx , Δy , up to +63 raster units or -64 raster units. The use of this order reduces core utilization for images with a large number of lines of $\frac{3}{4}$ inch or less in length, and is necessary for image subroutines.

When the relative positioning of the beam causes it to exceed the bounds of the screen area and a total displacement of 1024 raster units beyond the perimeter is not exceeded, the vectors, points, or character strokes displaced will be blanked. Unless the overflow limit of 1024 raster units is exceeded, the displaced beam can be returned to the normal display area. The virtual image size is four times the actual screen size, and can be positioned anywhere within a region equivalent to nine screen areas. (Figure 7)

Figure 6 illustrates the importance of relative vectors. In the illustration, the resistor would be represented by a series of incremental vectors and stored in 1130 core storage as a graphic subroutine. Thus, even though the resistor appears in several places on the screen, the order appears only once in the display list.

Graphic subroutines

A graphic subroutine is a sequence of display

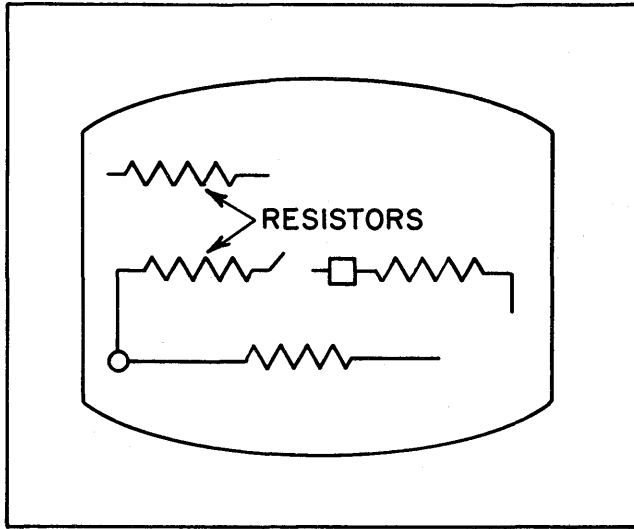


FIGURE 6—Circuit diagram showing resistor drawn by a subroutine

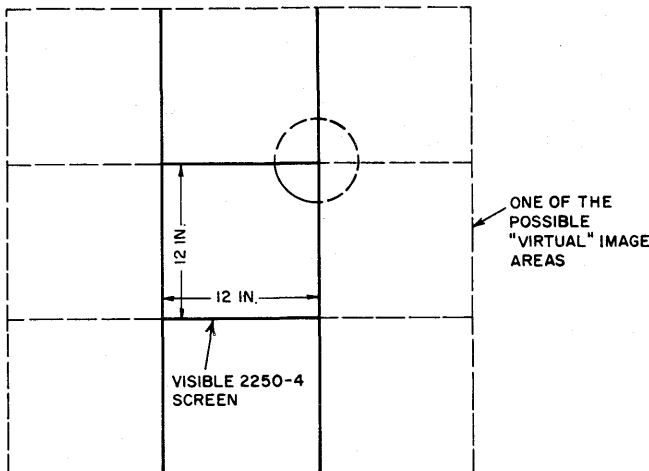


FIGURE 7—Automatic scissoring outside 2250 visible area

orders which displays a logical element or entity (such as a logic block, a resistor, a bolt, etc.). Graphic subroutine capability significantly reduces storage requirements for display images. Instead of requiring a copy of an image entity wherever it appears in the display image, the entity can be represented once by a graphic subroutine and can be generated as often as required by the execution of a "Branch" order. The display uses three orders to provide basic and multiple-level subroutine capability. (Figure 8)

Character generation

Character generation is a programmable func-

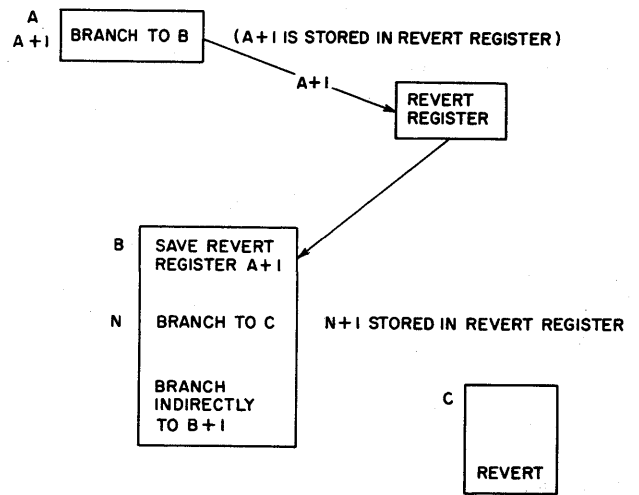


FIGURE 8—Multi-level subroutines

tion. Two character sizes can be displayed: .16 of an inch high and .24 of an inch high. Characters represented by their component strokes are organized into graphic subroutines and stored in 1130 core storage. Character generation is initiated by a "Set Character Mode" order. This order is followed by a series of "Branches" to character stroke subroutines.

The first branch order transfers program execution to the character-stroke subroutines. Up to two character strokes are contained with the 16-bit computer word. The last character stroke word of the character contains a revert bit, R, which performs the same function as the "Revert" order; i.e., it causes an automatic return to the display program. In addition, automatic character spacing results from the detection of the Revert bit.

Spacing to a new line is also automatic if the characters have been initially positioned by an absolute movement of the CRT beam. New line spacing is suppressed in the case of relative positioning. Special control codes within the character-stroke word are used to suppress spacing, position to a new line, insert a superscript or subscript, and reserve a location in CPU storage for later character placement.

Logical control orders

The logical control orders are used in the display program to reduce CPU program intervention, especially with respect to light-pen-detect interrupts and light-pen tracking. The control orders fall into three major categories: light pen control orders, conditional branch orders which

provide capability for logical decision making, and conditional interrupt orders which also supply logical decision-making capability and allow the CPU to be used effectively in support of display operation.

Stand-alone programming support

The Stand-Alone support, in addition to directly supporting the stand-alone configuration, is also the basis for the remote configuration. It has three components:

- Modifications to the 1130 Disk monitor system to allow the loading and execution of graphic programs.
- Extensions to the assembler to permit the symbolic coding of graphic programs.
- The Graphic Subroutine Package, a set of assembler language subroutines callable from both FORTRAN and Assembler Language, which perform image generation, image management, attention handling, support of the alphanumeric keyboard, and light-pen support.

The Graphic Subroutine Package is a set of assembler language subroutines which allow the FORTRAN or Assembler or Language programmer to create graphic images on the display. The displays can be constructed of lines, points, and characters. This package also furnishes the communication between the user and the program through routines related to the use of the light pen, function, keys, and the alphanumeric keyboard. The graphic subroutine package design is closely aligned to the design of the display interface, and its facilities make optimum use of the interface's functional capabilities. The structure of the graphic subroutine package can be best understood by looking first at the set of functions which are common to most graphic applications. These functions are shown in Figure 9.

The user at the display generates an attention through the alphanumeric keyboard, program function keyboard, or light pen. The attention is processed by an attention-control function which consists of:

- System attention handling which recognizes attention and indicates to the program that an attention of a certain type has occurred.
- User attention control, which controls the program flow and determines if any further processing is needed.

The attention controller may require access to the problem model or data base in the system. Control from the attention controller may be passed to the application program which performs the arithmetic and logic processing on the data base. When this is completed, control is passed to a group of routines which generate and organize new display data. This new display data is placed in the buffer, thus modifying the display content and completing the cycle back to the user.

The Graphic Subroutine Package provides the FORTRAN programmer with all the necessary routines required for the image generation, image control, and attention handling functions of his application.

Image management and control routines allow the logical grouping and structuring of the basic display elements (lines, points, and characters). Any group of these elements becomes an entity. Thus the four lines forming a box become one box entity that the user can create, delete, modify, or group with another entity.

Each created entity or elements within an entity can be given a unique identification value which is returned to the program (by the at-

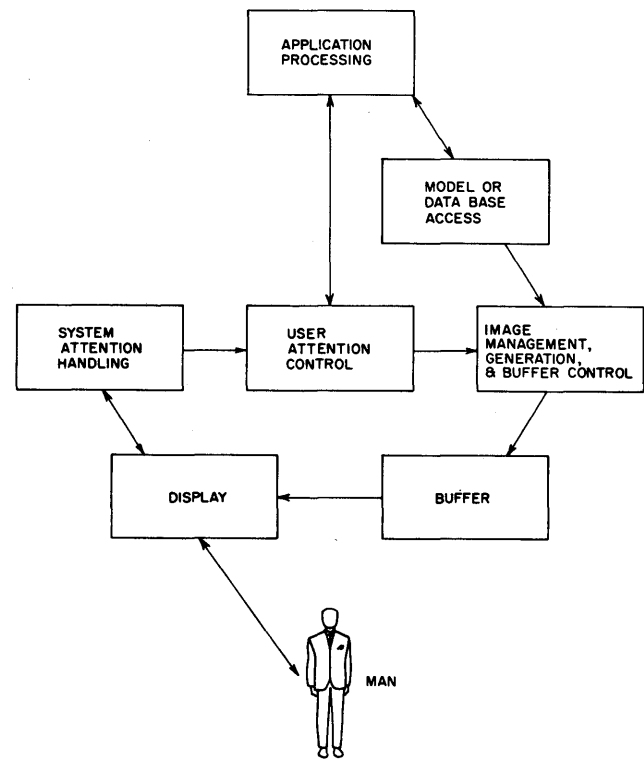


FIGURE 9—Functions and data flow within a graphic application

tention handling routines) when the entity is detected by the light pen.

The attributes of an entity can be dynamically controlled. A grid, for example, can be turned on or off. When the grid is displayed, it could be made detectable by the light pen (for generating a drawing) or undetectable when it is used as a background reference.

The collection of all entities is called an image entity, whose structure is defined by the series of calls to the image management subroutines. The elements within an entity are defined by a series of calls to the image generation subroutines. Some functions performed by image management routines are:

- . Initialize the image construction area.
- . Begin an entity, which defines the name, beginning, and attribute of the entity. This routine will usually be followed by calls to the image generation routines.
- . End an entity.
- . Delete an entity.
- . Update an entity: this routine is used to add, change, or remove elements from an entity.
- . Display an image entity.
- . Stop the display.

The image generation subroutines are used to define the contents of an entity by converting the program input data into display format. Thus, an array of user X, Y, floating point data representing a graph is scaled, translated, scissored (eliminating any portions of the graph which are outside the display area of the display), and converted into vectors for display. By changing the scale factor, the same graph can be enlarged or made smaller. The graph can also be moved by changing the translation factor.

Some of the functions performed by the image generation routines are:

- . Set the image generation control parameters of scaling, translation, scissoring, type and format of user data, and type and format of output data.
- . Plot Line(s)
- . Plot Points
- . Plot Text
- . Plot Grid
- . Copy and Entity

Attention handling routines allow the program to specify the types of acceptable attentions (user

actions), process the attentions when they occur, and provide the attention information to the user program.

The graphic subroutine package also includes the following auxiliary routines:

- . Tracking subroutines which allow the user to draw lines, through the use of the light pen and a tracking symbol.
- . Alphanumeric key support routine which allows the user to input and edit messages from the alphanumeric keyboard.

Remote configuration support

The programming support package for the remote configuration is structured such that the 1130 can handle all the graphic functions in an application—image management, image generation, attention-handling, and communication with the application program—and call on the central system for computational assistance and/or access to a large central data base. It is important to note how this structure relates to that illustrated in Figure 9. It can be seen that the functions of the graphic subroutine package are common to both configurations, and that the functions labeled “Model or Data Base Access” and “Application Processing” reside in the 1130 or in a central System/360, depending on whether we are dealing with a Stand-Alone or a Remote 1130/2250 System configuration. Thus, the Stand-Alone support is fundamental to supporting the Remote configuration.

The Remote Configuration support consists of two elements:

1. Data transmission and conversion subroutines which facilitate communication and interchange of data between an IBM System/360 and one or more 1130 Computing Systems.
2. The Satellite Graphic Job Processor (SGJP), which allows a remote display user to define, initiate and control a job which is either exclusively processed in System/360 or concurrently in System/360 and the 1130/2250 System.

Data transmission and conversion subroutines

The data transmission and conversion subroutines make up what is called the processor-to-processor (PTOP) program. They are in-

voked via calls in the FORTRAN IV or the ASSEMBLER languages.

Separate sets of transmission subroutines are available in System/360 and in the 1130. They perform transmission control functions and insert the proper transmission-line control characters, enabling the programmer to perform data transmission without having detailed knowledge of telecommunications programming.

The transmission subroutines perform the following in either system:

- Initialize the communication line(s).
- Transmit and receive data via the communication lines.
- Test the status of a previously requested transmit or receive operation.
- Activate a user-written asynchronous routine in the other system.
- Terminate the communication linkage between the System/360 and the 1130 PTOP programs.

The capability of terminating the communication link in one system at a time makes it possible, for example, for a new core load (in the 1130) to reinitialize communication with the transmission program existing in System/360. This would allow a user to monitor the progress of lengthy computation by receiving intermediate results, terminating communication, analyzing these results, and then reinitializing communication with perhaps a new set of input parameters.

In addition, the System/360 transmission subroutines enable the programmer to terminate the 1130 mainline program that is currently being executed.

Conversion subroutines are provided to resolve differences in the FORTRAN data formats of System/360 and the 1130. These subroutines can be invoked only from the System/360 program.

Figure 10 illustrates the sequence of operations and data flow for transmission from the 1130 to System/360. Figure 11 illustrates the same for transmission from System/360 to the 1130.

The Satellite Graphic Job Processor

The Satellite Graphic Job Processor (SGJP) is a program that elicits job control information from a user at the display unit, enabling him to process a job exclusively in the System/360 or concurrently in the System/360 and the subsystem. SGJP interprets the job control information

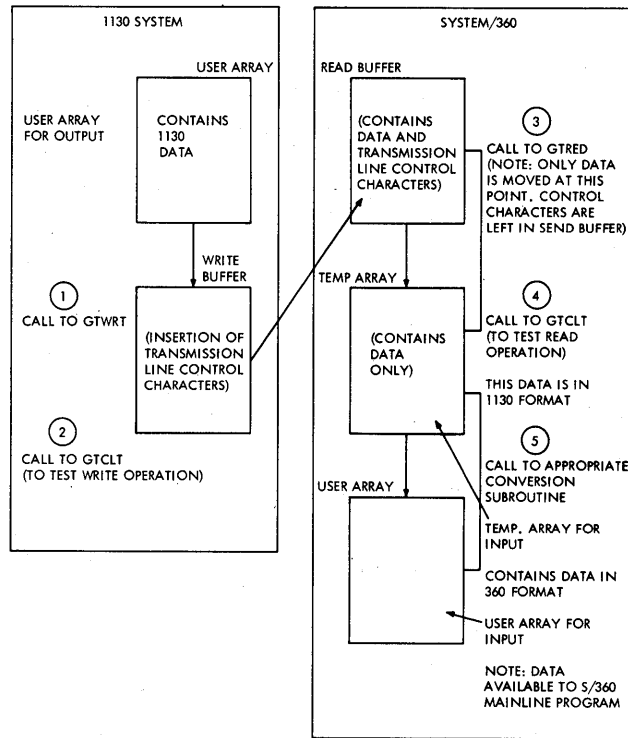


FIGURE 10—Sequence of operations and data flow: Transmission to system/360

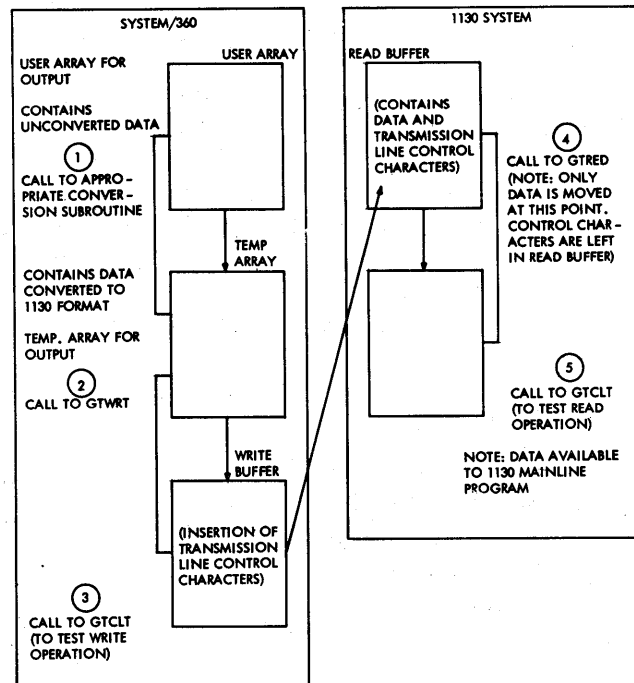


FIGURE 11—Sequence of operations and data flow: Transmissions to 1130

entered through the display unit and converts it into a language (Job Control Language) meaningful to the System/360 Operating System. The Job Control Language is then passed to the operating system to actually initiate the desired program. These services allow the non-programmer user to conveniently, rapidly define and start his job.

A job is defined as the fundamental unit of work for a computing system as seen by the user. A job may consist of one or more job steps, each

of which requests the processing of a program or procedure.

Job control operations provide the job control information necessary to define jobs step-by-step, to describe data characteristics and device requirements related to job steps, and to start the processing of jobs. Job control information is presented to the operating system from the subsystem by means of SGJP.

The job control operations available with SGJP and their functions are listed below:

<i>OPERATION</i>	<i>FUNCTION</i>
LOG ON	Identifies the user to the operating system.
SPECIFY JOB STEP	Names a program or procedure to be executed in the System/360.
BEGIN PROCEDURE	Causes a named procedure to be processed as a foreground job in the System/360.
DESCRIBE DATA	Identifies data to be used in the specified System/360 job step.
BEGIN JOB	Starts the processing of the defined System/360 job.
SPECIFY 1130 PROGRAM	Names an 1130 program that is to run in conjunction with a program in the System/360.
WRITE MESSAGE	Sends a message to the System/360 operator and handles a reply to the 2250 user.
ENTER DATA	Allows 80-character records to be entered from the display unit or a card reader for use by the System/360 program.
CANCEL JOB	Deletes the job that is currently being defined.
RECALL	Allows the user to re-examine and modify previously completed job control operations.
LOG OFF	Completes user interaction with the display unit and frees it for the next user.

The above operations can be used to describe and start the processing of all types of application jobs (for example: a graphic program, an assembly, a service program, etc.) directly from the display unit. A job initiated at the subsystem can be designated to process as either a foreground or background job.

System-user communication

To enable the selection and performance of job control operations, SGJP establishes communication between the user at the subsystem and the operating system by means of displayed frames.

The frames are displays that request the entry

of job control information from the user. Each information request is indicated by a word or phrase. The entry area related to the individual request is indicated by a short underscore or a rectangular box where an entry is to be made.

The frames are displayed in a logical sequence (that is, only as they are applicable to the user's job). Through interaction with SGJP, the user's responses to the frames convey the information necessary to process his jobs.

The particular information to be supplied in response to the frames (such as accounting code, procedure name, etc.) depends upon installation and user requirements.

There are two types of frames used for com-

munication between the user and SGJP. These are the *select frame* and the *parameter frame*.

The select frame

The select frame presents the job control operations available to a user at each point during definition of his job. It is from this frame that the user selects the next operation he wants to perform. The select frame is divided into three areas: the selection area, the history area, and the message area (see Figure 12).

The selection area contains the list of the job control operations currently available to the user and then entry areas for their selection. This list varies as operations are selected and processed, and guides the user by presenting only the operations that are applicable to his job. An entry area for selecting an operation is indicated by a short underscore preceding the name of the operation. For an operation for which the user must provide information, the area in which the user is to enter information is denoted by a short underscore following the name of the operation.

The history area contains a sequential list of the 14 most recent job control operations selected by the user. As the user completes each operation, an entry for that operation is added to the list. If there already are 14 operations in the list, the oldest operation is removed from the history area as each new operation is added to the list. Each operation is preceded by a 3-digit history number to indicate the sequence in which the operation was performed during the current session.

The message area is used to display status information and diagnostic messages.

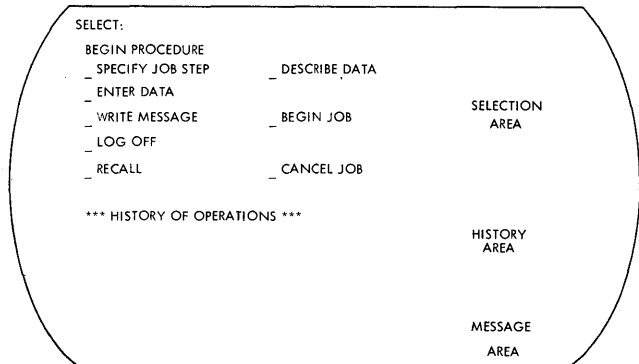


FIGURE 12—Sample select frame

The parameter frame

The parameter frame asks the user to supply values, called parameters, that are necessary to complete a selected operation. It is divided into three areas: the description area, the message area, and the key area (see Figure 13).

The description area contains the name of the selected operation, all parameter requests, and the entry areas associated with the requests.

If the parameter request is one for which information can be entered from the keyboard, the request is followed by a short underscore. In addition, if the user must provide the information, the entry area following the request is enclosed in a box.

If the parameter request may be selected with the keyboard or with the light pen, the request is preceded by a short underscore. In some cases, the user is given the choice of two or more options that will satisfy a parameter request. Normally, one of the options is completely underscored. The underscored option is called the default option, and this option will take effect if no other option is selected for the request. The message area is used to display status information, diagnostic messages, or replies from the system operator.

The key area displays the words END and CANCEL which may be used to perform the END and CANCEL functions with the light pen. Because they perform the same functions as the END and CANCEL keys on the alphanumeric keyboard, they are referred to as the END key and CANCEL key.

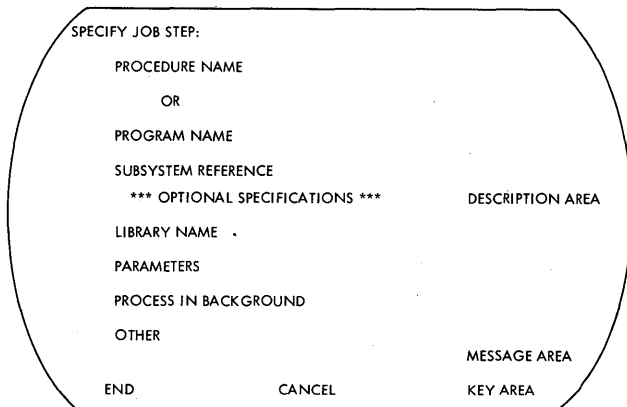


FIGURE 13—Sample parameter frame—Specify job step

Application example

To illustrate the application environment and to demonstrate the sequence of operations required to execute a graphics application from a remote site, the following application example of a sample job that is initiated and processed by a user at the subsystem is presented. Illustrations of the frames that accompany the job control operations at the display unit are included.

John Doe, a user, is sitting at a display unit that is not in use. He wishes to process a job to design a lens at the display unit. This job consists of one System/360 job step that is executed in conjunction with a related 1130 program.

The System/360 job step calls for execution of a program named LENSDESN which will perform the calculations for a lens display. LENSDESN uses a previously created data set named LENS SAVE. The subsystem is referenced as DEVICE by the program. The data set is referenced as OUTPUT and is to be retained at the end of the job step.

The 1130 program, named LPGM, operates in the subsystem in conjunction with the System/360 program. LPGM contains the specifications for a "thin lens" design, except for two user-supplied parameters that specify the aperture and focal lengths of the lens. During processing, LPGM accepts data entered from the 2250 Display Unit and transmits it to the System/360 program where computations for the lens display are performed. When the System/360 program has completed the calculations, it transfers the results to LPGM. LPGM then displays the results on the 2250 screen.

Note that, during definition of the job, the user recognizes an error in a job control operation he has already completed and uses the RECALL operation to correct the error. He then completes the definition of the job and starts processing.

The first thing the user must do is identify himself to the operating system and provide his account number of KG505301. By performing the CANCEL function from the keyboard he obtains the LOG ON frame. First, he enters his name in the frame from the alphanumeric keyboard. Then, he positions the cursor on the screen to ACCOUNT and enters his account identification from the keyboard. To obtain a list of the job

control operations that he performs, he also selects the PRINTED RECORD option.

LOG ON:

```

USER'S NAME      JOHN DOE
ACCOUNT          KG505301
OTHER
X PRINTED RECORD
___ DISCONNECT THE SUBSYSTEM

```

OPERATIONAL HINTS:

1. USE THE KEYBOARD TO ENTER ALPHAMERIC INFORMATION
2. ENTRY AREAS ARE INDICATED BY A SHORT UNDERScore OR A BOX. BOXED ENTRY AREAS DENOTE REQUIRED INFORMATION.
3. BEFORE ENTERING ALPHAMERIC INFORMATION, POSITION THE CURSOR TO THE ENTRY AREA WITH THE JUMP KEY OR THE LIGHT PEN.
4. DESIGNATE A SELECTION WITH THE LIGHT PEN OR THE KEYBOARD. DEFAULT SELECTIONS ARE UNDERLINED.
5. THE END AND CANCEL KEYS ON THE SCREEN ARE EQUIVALENT TO THE KEYBOARD END AND CANCEL KEYS.
6. USE THE END KEY TO INDICATE FRAME COMPLETION. USE THE CANCEL KEY TO NEGATE A FRAME.

```

END              CANCEL

```

At this point, the user has completed the LOG ON operation. He performs the END function to indicate that he has finished entering information on the LOG ON frame.

A select frame now appears on the screen. Displayed in this frame are the various job control operations the user can perform at this time. The first entry in the history area of the frame reflects the LOG ON operation he has just completed. The user now selects SPECIFY JOB STEP in order to define the System/360 program.

SELECT:

```

BEGIN PROCEDURE
X SPECIFY JOB STEP
___ ENTER DATA
___ WRITE MESSAGE
___ LOG OFF

```

HISTORY OF OPERATIONS

```

000 LOGON      JOHN DOE

```

The SPECIFY JOB STEP frame now appears on the screen. This frame requests certain information about a job step, such as the name of the procedure or program, subsystem reference, and other optional specifications.

The user begins SPECIFY JOB STEP by entering the name of the program (LENSDESN) from the alphanumeric keyboard. To indicate that his job step is a program, he enters the name after the PROGRAM NAME option. Then, since this System/360 program will be processed in conjunction with an 1130 program (LPGM), he enters the subsystem reference, FT49F001, from the alphanumeric keyboard. The subsystem reference is a symbolic name by which the user's System/360 program refers to the subsystem. The user then enters from the keyboard the parameters (aperture of 5.0 and focal length of +4.2) necessary for his program. The frame now appears as follows:

```

SPECIFY JOB STEP:

  PROCEDURE NAME
    OR
  PROGRAM NAME      LENSDESN
  SUBSYSTEM REFERENCE FT49F001
  ***OPTIONAL SPECIFICATIONS***
  LIBRARY NAME ____

  PARAMETERS APERT=05.00, FOCLEN=+4.20

  ____ PROCESS IN BACKGROUND
  OTHER ____

  END              CANCEL

```

Since all information necessary for his job has been entered in the frame, the user performs the END function to indicate that the SPECIFY JOB STEP operation is complete.

A second select frame appears on the screen displaying the job control operations now available to the user. The second entry in the history area reflects the SPECIFY JOB STEP operation. To describe his data set, the user selects DESCRIBE DATA.

```

SELECT:

  BEGIN PROCEDURE
  __ SPECIFY JOB STEP  X DESCRIBE DATA
  __ ENTER DATA
  __ WRITE MESSAGE    __ BEGIN JOB
  __ LOG OFF
  __ RECALL           __ CANCEL JOB

  ***HISTORY OF OPERATIONS***

  000 LOGON          JOHN DOE
  001 JOB STEP LENSDESN

```

The DESCRIBE DATA frame now appears on the screen. This frame requests the information necessary for the user to identify his data set, such as the data set name, the data reference by which the System/360 program refers to the data set, and other specifications.

The user begins the DESCRIBE DATA operation by entering the name of his data set, LENSSAVE, from the alphanumeric keyboard. Then, he positions the cursor to DATA REFERENCE and enters the name OUTPUT from the alphanumeric keyboard. Options for status and disposition can now be specified. The user knows that the data set is CATALOGED; i.e., it already exists and can be found automatically by the operating system. The user does not have to specify CATALOGED, however, because it is a default option (note underscore on frame). Furthermore, the user does not have to specify a disposition since he wishes to retain the data set and the operating system (if he specifies no option) will assume the disposition already assigned to the data set (KEEP)

```

DESCRIBE DATA:

  DATA NAME      LENSSAVE
  DATA REFERENCE OUTPUT
  INDICATE STATUS:  CATALOGED  __ OLD
                   __ MOD  __ SHARE  __ NEW
  ***ADDITIONAL INFORMATION WILL BE REQUESTED***
  FOR OTHER THAN CATALOGED STATUS
  OTHER ____
  CHOOSE DISPOSITION:  __ KEEP  __ PASS  __ DELETE
                       __ CATLG  __ PRINT  __ PUNCH

  END              CANCEL

```


The user now performs the END function to indicate that the DESCRIBE DATA frame has been completed.

A third select frame is now displayed on the screen. At this point, however, the user suddenly realizes that he meant to specify a focal length of 4.02 (instead of 4.20) as the parameter for his lens specification in the SPECIFY JOB STEP operation. The user decides to correct the operation in which he provided the focal length parameter. To re-examine the operations in his job, he selects RECALL on the select frame.

```

SELECT:
  BEGIN PROCEDURE
  __ SPECIFY JOB STEP      _ DESCRIBE DATA
  __ ENTER DATA
  __ WRITE MESSAGE        _ BEGIN JOB
  __ LOG OFF
  X RECALL                _ CANCEL JOB

  ***HISTORY OF OPERATIONS***

000 LOG ON      JOHN DOE
001 JOB STEP    LENSDESN
002 DESCRIBE    LENS SAVE
    
```

He performs the END function and the SPECIFY JOB STEP frame is now displayed as it appeared when the user had completed this operation earlier in his job.

The user positions the cursor to the entry area following PARAMETERS on the frame. He uses the ADVANCE key to position the cursor to the desired point of change and enters 02 in place of the previous 20. The frame now appears as follows:

```

SPECIFY JOB STEP:
  PROCEDURE NAME
  OR
  PROGRAM NAME LENSDESN
  SUBSYSTEM REFERENCE FT49F001

  ***OPTIONAL SPECIFICATIONS***
  LIBRARY NAME __
  PARAMETERS APERT=05.00, FOCLEN=+4.02
  __ PROCESS IN BACKGROUND
  OTHER __

  END          CANCEL
    
```

The RECALL frame appears on the screen with the first operation in the job (after LOG ON) indicated after "CURRENT ITEM." Because it was during this operation (SPECIFY JOB STEP) that the user specified the parameter that he now wishes to change, he selects the MODIFY option. The RECALL frame appears as follows:

```

RECALL:
  CURRENT ITEM - 001 JOB STEP LENSDESN
  __ ACCEPT THIS OPERATION UNCHANGED
  X REVIEW, MODIFY IF DESIRED
  __ INSERT NEW OPERATION BEFORE THIS ONE
  __ OMIT THIS OPERATION

  ***HISTORY OF OPERATIONS***

000 LOG ON      JOHN DOE
001 JOB STEP    LENSDESN
002 DESCRIBE    LENS SAVE
003 RECALL
    
```

Since this was the only change he wished to make, he performs the END function and the RECALL frame is displayed again.

The next operation (DESCRIBE DATA) is shown after "CURRENT ITEM" and the SPECIFY JOB STEP has been added to the history area of the frame. The frame appears as follows:

```

RECALL:
  CURRENT ITEM - 002 DESCRIBE LENS SAVE
  __ ACCEPT THIS OPERATION UNCHANGED
  __ REVIEW, MODIFY IF DESIRED
  __ INSERT NEW OPERATION BEFORE THIS ONE
  __ OMIT THIS OPERATION

  ***HISTORY OF OPERATIONS***

000 LOG ON      JOHN DOE
001 JOB STEP    LENSDESN
002 DESCRIBE    LENS SAVE
003 RECALL
004 JOB STEP    LENSDESN
    
```

Since the user has no changes to make in the DESCRIBE DATA operation he performs the END function. The system assumes he wishes to retain the operation since ACCEPT is the default option. Because this was the last operation in the history area that could be recalled, a select frame appears on the screen so that the user can continue his job.

At this point, the user has completed all specifications needed for his job. Therefore, he selects BEGIN JOB to indicate that his job is ready to start processing.

```

SELECT:
  BEGIN PROCEDURE
  _ SPECIFY JOB STEP   _ DESCRIBE DATA
  _ ENTER DATA
  _ WRITE MESSAGE      X BEGIN JOB
  _ LOG OFF
  _ RECALL             _ CANCEL JOB

***HISTORY OF OPERATIONS***

000 LOGON      JOHN DOE      003 RECALL
001 JOB STEP   LENSDES      004 JOB STEP   LENSDES
002 DESCRIBE   LENS SAVE    005 DESCRIBE   LENS SAVE
    
```

A message containing an identification number given to the job by the system is displayed in the message area of the above select frame. In this sample the message returned is:

"JOB SCHEDULED AS J0240001"

The user performs an END function to acknowledge the message.

The SPECIFY 1130 PROGRAM parameter frame now appears on the screen and the user enters the name of his 1130 program (LPGM) from the alphanumeric keyboard.

```

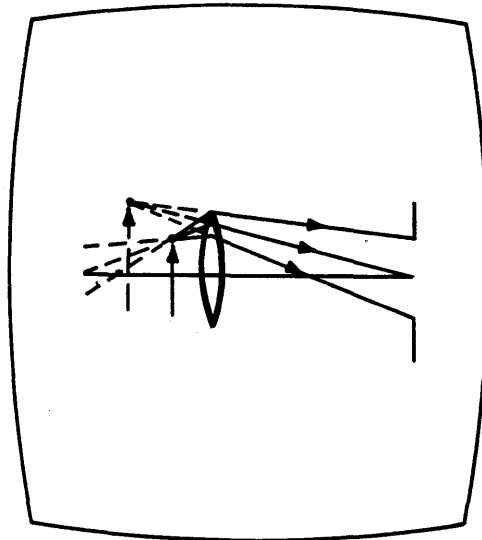
SPECIFY 1130 PROGRAM:

NAME  LPGM

END          CANCEL
    
```

The user now performs the END function to indicate the frame is complete and his job begins processing.

Using the aperture and focal length parameters provided as part of the SPECIFY JOB STEP operation, the System/360 program performs calculations and transmits data to the 1130 necessary to display an image, such as the one that follows:



The 1130 program is designed to accept additional information entered from the alphanumeric keyboard or with the light pen. The data are transmitted to the System/360 where new calculations are performed and information needed to modify the lens display is returned to the 1130. By providing new data and manipulating his display, the user designs a lens that meets his requirements. When the user has completed designing the lens, he terminates his program in a manner specified by the installation.

When the job has completed processing, a select frame automatically appears on the screen with a message indicating that the job has been completed. Since the user had only this one job to process, he selects LOG OFF. (If the user had wanted to define additional jobs, he could have continued by selecting another job control operation.)

```

SELECT:
  BEGIN PROCEDURE
  ___ SPECIFY JOB STEP
  ___ ENTER DATA
  ___ WRITE MESSAGE
  X LOG OFF
  ___ RECALL
  ***HISTORY OF OPERATIONS***
000 LOGON      JOHN DOE
001 JOB STEP   LENSDESN
002 DESCRIBE   LENS SAVE
003 RECALL
004 JOB STEP   LENSDESN
005 DESCRIBE   LENS SAVE
006 BEGIN      J0240001
007 SPECIFY 1130 LPGM

Job J0240001 completed
    
```

The LOG OFF frame now appears on the screen. The frame contains a message from the accounting routine asking the user to supply the number of jobs he has executed since he logged on. Since the user has executed only one job, he positions the cursor to the entry area after TEXT and types the number "1." The user wants to leave the 2250 Unit available for SGJP operation by a different person, so he does not designate the DISCONNECT THE SUBSYSTEM option. The frame appears as follows:

```

LOG OFF
  ***UP TO 72 CHARACTERS MAY BE ENTERED FOR ACCOUNTING***
  TEXT 1
  ___ DISCONNECT THE SUBSYSTEM

  HOW MANY JOBS HAVE YOU RUN SINCE LOG ON?
  ENTER NUMBER AFTER TEXT AND PERFORM END FUNCTION.

  END          CANCEL
    
```

Having entered the required information, the user performs the END function. The screen now goes blank. The LOG ON frame is made available to another user when the CANCEL key is depressed at the keyboard.

Communication requirements

The important question in the design of a remote graphic system is whether the 300-cps or higher-speed (5000 to 30,000 cps) transmission between the 1130 and System/360 is sufficient for

graphic applications. To answer this question, it is important, first, to recognize that the satellite system is not always communicating with the System/360. During the input phase of the application, the system is in a stand-alone mode for some duration of time within an application. During this mode, there is no transmission to System/360 and the response time due to operator actions will be very fast.

The effect of the data rate on response time should, therefore, be considered only when the system is communicating to System/360, and the relationship of this time to the total application cycle time is important. Figure 14 illustrates this principle. Specifically, the turnaround time for transmission to the System/360 computation or access of data in System/360, and then transmission from System/360 to the satellite computer must be considered. Depending on the type of application, the justification of the 300 cps or higher speed is based on any or all of the following:

- The amount of transmitted data
- The ratio of System/360 computation or data access time to transmission time
- The ease with which the data display can be overlapped with transmission. For example, a quick response time can be achieved by starting to display portions of the data as soon as the data is received in the subsystem.

Last and most important, it must be recognized that the 1130 computing capability reduces the frequency of communication between the dis-

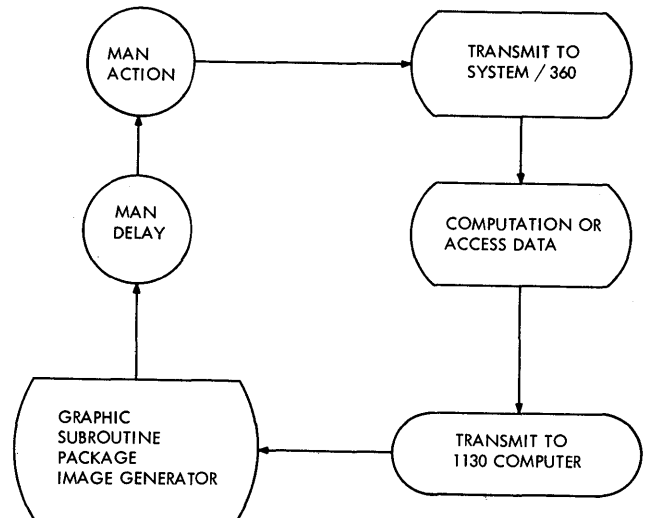


FIGURE 14—Response-time cycle in a remote configuration

play and System/360. If some operator action causes slower response than is normally acceptable to the operator, it will be tolerated as long as it does not occur frequently.

In general, 300 cps should be satisfactory for analysis-type applications in which small amounts of data (messages, control parameters, single graphs, etc.) are transmitted between the subsystem and the central processor; higher transmission rates may be needed for applications re-

quiring frequent transmission of large amounts of data.

ACKNOWLEDGMENTS

The following individuals made important contributions to this paper: Mr. D. B. Arnold, Product Publications, Mr. R. L. Bean, Programming Publications, and Mr. J. M. Ettinger and his group, Remote Graphic System Support.

The adage graphics terminal

by THOMAS G. HAGAN, RICHARD J. NIXON
and LUIS J. SCHAEFER

Adage, Inc.
Boston, Massachusetts

INTRODUCTION

Interactive computer graphics applications require that the system present visual displays to the user for his interpretation and that these displays change in response to actions taken by the user as he goes about solving the problem for which he is using the system. Effectiveness of a graphics system is very dependent upon the complexity of the displays which it is able to present and the speed with which it can produce changes in the displayed image. A CRT displaying dynamic images of sufficient complexity for meaningful visual assimilation imposes a data transfer and arithmetic burden very much greater than that imposed by such conventional output media as typewriters, line printers and point plotters.

Terminals intended to provide interactive graphics capability have, therefore, quickly evolved to the point where they typically include a small computer, as shown in Figure 1, for purposes of refreshing the CRT and for some local management of the image manipulation workload imposed by the terminal.¹ For images composed of more than a few dozen vectors, however, the coordinate transformation task necessary for a truly dynamic display is too much for the limited arithmetic capability of a small digital computer. Thus, lacking special coordinate transformation capability, such systems are restricted to handling only the quasi-static display of reasonably complex images—those comprising 1,000 or more vectors. It has been suggested that the rotation and scaling tasks be thrown back upon the central computer to which the graphics terminal is connected.² Generalized scaling and six-degree-of-freedom geometrical coordinate transformations require about 25 arithmetic operations per vector. If the display is to be truly dynamic, these operations must be performed once per frame for each vector. At 40 frames per second, such operations upon an image composed of 1,000 vectors constitute a computational

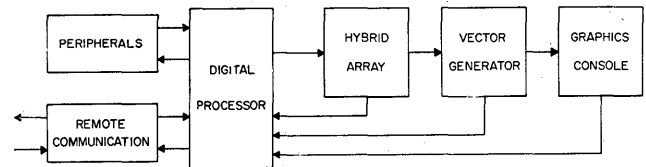


FIGURE 1—Overall organization of a graphics terminal incorporating a local digital processor

load of one million arithmetic operations per second, most of them multiplications. Central computers do exist which are capable of such performance, interleaved with the necessary high-data-rate output communication to a display device, but operational costs are prohibitive for routine computer graphics applications, and therefore this method of achieving a dynamic output display is suitable only for feasibility studies and programming research. In practice, users of systems which depend upon the central computer for coordinate transformation, or upon the digital arithmetic capabilities of a small local computer, are restricted to working in applications areas for which quasi-static displays are satisfactory, and the powerful pattern recognition abilities of the eye and brain which depend upon *motion* of a perceived image can be exercised only for very simple images composed of a few dozen vectors at most.

Organization of the AGT

An alternative approach was taken in the design of the AGT. Coordinate transformation hardware is included in the graphics terminal itself so that the transformations necessary for dynamic displays can be included among the image manipulations accomplished locally, with minimum recourse to the central computer. As a result, images composed of as many as 5,000 line segments can be displayed dynamically with arbitrary

changes in scale factor, position and rotation between successive frames.

Overall system organization conforms to the block diagram of Figure 1. The digital processor is a 30-bit word length, 2-microsecond cycle time machine with core memory sizes ranging from 4k to 32k words. Digital peripherals include card reader, line printer, high and low speed magnetic tape units, and a disk subsystem with storage capacity ranging from 3.1 million characters (1 disk drive unit attached) to 12.5 million characters (4 disk drive units attached).

Remote communications interfaces are available for telephone line communication to a central computer at speeds ranging from 1,200 baud (150 ASCII characters per second) to 40.8 Kilobaud (5.1K ASCII characters per second). Operation with the lower speed communications to the central computer is feasible because of the comparatively high degree of autonomy of the terminal.

The hybrid array shown as part of the system in Figure 1 is used for scaling and coordinate transformation of values as they are passed from core memory in the digital processor to the vector generator. The vector generator output drives horizontal, vertical and intensity inputs of the CRT contained in the graphics console. The console also contains function switches, light pen, control dials, data tablet and joystick, whose outputs are monitored by the digital processor. A photograph of a complete AGT is shown in Figure 2.

Line drawing CRT display

Most graphics terminals which incorporate a local

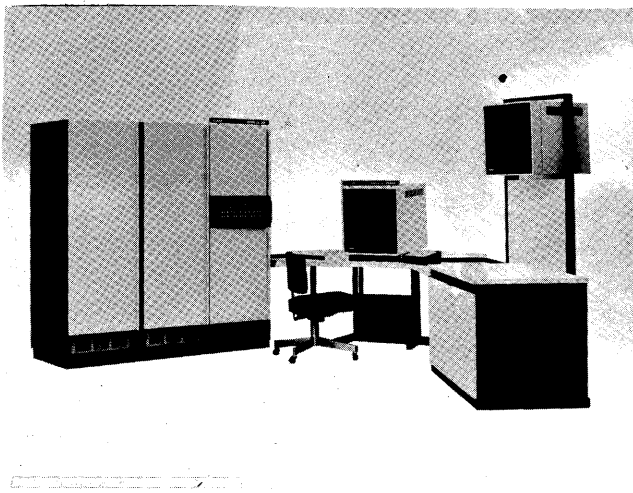


FIGURE 2—The Adage Graphics Terminal, with auxiliary high-mount viewing scope

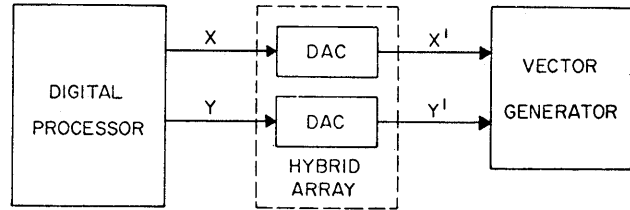


FIGURE 3—A graphics terminal in which coordinate values are passed from the digital processor to the vector generator through a pair of DAC's

computer use a pair of DAC's between the digital processor and the vector generator, as shown in Figure 3. Digital values extracted sequentially from a display list contained in memory in the digital processor are fed through the DAC's to present analog x' and y' values to an analog vector generator. The vector generator, in turn, develops straight-line segments on the face of the CRT.

Systems of this type develop pictures on the CRT by drawing a series of connected vectors, in contrast with systems which modulate intensity during a TV-like raster scan. Each vector in the series drawn on the CRT can be specified as either visible or invisible. A visible vector corresponds to a "draw" operation; an invisible vector to a "move" operation. Use of such a scheme for generating a figure such as the block letter "A" is shown in Figure 4. In this case, a total of 13 vectors is used, 11 visible "draw" vectors and 2 invisible "move" vectors.

Image manipulation

Graphics terminals using the end-to-end vector drawing technique for generating pictures have usually required that the complete picture displayed on the CRT

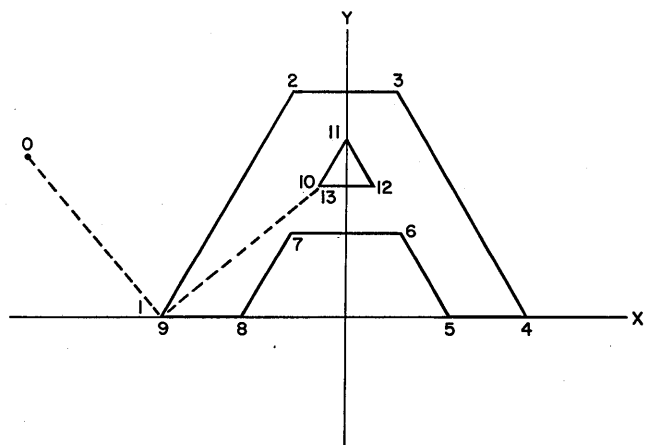


FIGURE 4—Creating a picture (here the block letter "A") with a series of connected visible and invisible vectors

be represented by a display list in memory specifying a coordinate pair for each and every vector end point contained in the picture. Any change in the picture is accomplished by computing new values for coordinates stored in the display list. Manipulation of images in such systems, therefore, consists of performing arithmetic procedures to achieve desired results. For example, to permit translation of an image element in x and y , it is necessary to implement a procedure whereby each coordinate describing the image element has added to it values that represent the increments of motion desired in x and y .

Scaling of an image element, i.e., changing its size, can be accomplished by implementing a procedure whereby two multiplications are accomplished for each coordinate describing the image element.

Structured images

A simple image is one whose description consists of a string of values specifying the end points of vectors or straight lines composing the image. A structured image is built up of sub-images by performing operations upon one or more simple images. The operations can consist of translation and scaling transformations of the type we have already described. A complete description of a complex image, therefore, includes one or more coordinate lists and also the specification of one or more transformation operations.

The picture shown in Figure 5 can be composed in several different ways. If composed as a simple, unstructured image, 25 coordinate pairs would have to be specified. Twenty of these would be necessary for the 20 "draw" vectors, and five would be necessary for the invisible "move" operations necessary to get from square to square, including an initial move to get positioned to begin the first square. A total of fifty values—25 for x and 25 for y —stored in memory would therefore be necessary to describe this image.

On the other hand, if scaling and translation operations are available, the image can be built up of sub-images, using a smaller number of values to describe the same resultant image. For example, it is possible to compose the image by storing five coordinate pairs for one square and then specifying five different scale factors and five x - y positions in order to compose the total image comprising five squares. If described this way, a total of 25 values is sufficient to describe the image: 10 x - y values for a square, 5 scale factor values, and 10 x - y position values to specify the five positions at which the squares are to be located. With no scale factor operator available, 30 values could suffice to define the image: five coordinate pairs would specify the large square, five coordinate pairs would specify the

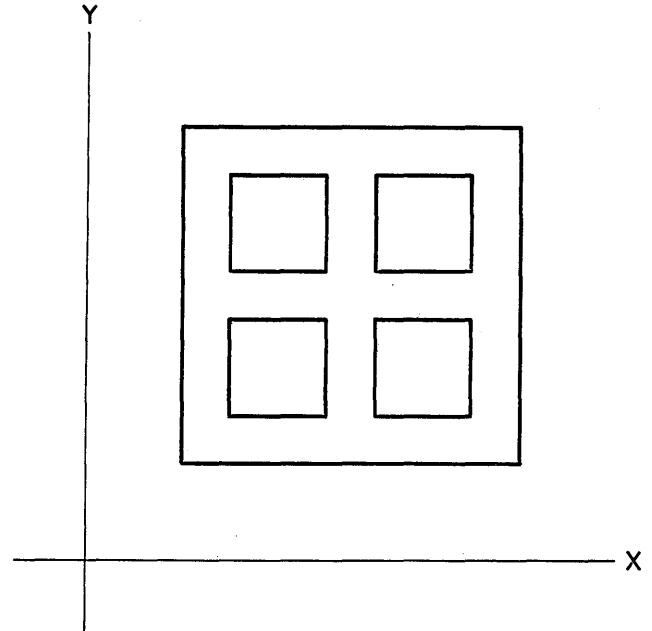


FIGURE 5—A picture resulting from an image which can be defined in terms of five squares

small square, and then five pairs would specify the positions for each of the five squares. This is a total of 15 coordinate pairs, or 30 values. The point of all this is that the availability of translation and scaling operations allows a complex image to be described with fewer words of memory than are necessary to describe a totally unstructured image.

Even more important is the ease of introducing changes in a structured image. A change need only be made once in a sub-image to have it properly reflected in all repeated instances of the sub-image. In the present case, by changing the square to a hexagon, for example.

The translation and scaling operations necessary to permit the kind of structured image composition described above can be accomplished by implementing the following equations:

$$x' = x_a + SC \cdot x \quad (1)$$

$$y' = y_a + SC \cdot y \quad (2)$$

Hardware for image manipulation

The hybrid array shown in Figure 6 is incorporated in the AGT10 for implementing translation and scaling. It accepts digital inputs as coefficients for specifying translation and scale parameters and then, subsequently, it accepts successive x and y input values to produce appropriately translated and scaled x' and y' output values. The key to performing the necessary

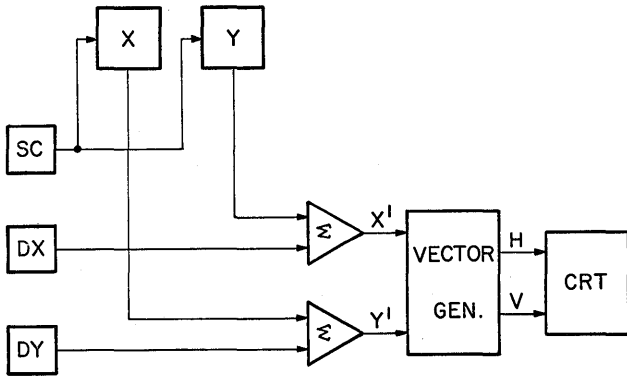


FIGURE 6—The hybrid array included in the AGT10 for scaling and positioning

multiplications is the multiplying DAC element included in the array. The operation of multiplying DAC's for image manipulation purposes has been described in a previous paper.³

A limited degree of structured image manipulation capability has been provided in other display systems by driving the x and y output DAC's from accumulator registers. This permits operation in an incremental vector mode, where each line segment in the string is specified in terms of incremental displacement in x and y from the end of the preceding vector. Display lists organized as strings of incremental coordinate pairs can be treated as sub-images and can be located at arbitrary positions on the screen by first loading the x and y accumulator registers with initial values corresponding to the desired displacements in x and y. Such systems are, therefore, capable of positioning sub-images whose display lists are organized in terms of incremental coordinates, but they cannot subject sub-images to operations such as scaling or rotation, and they cannot treat as a sub-image any display list composed of absolute coordinates. In the AGT, use of hybrid coordinate transformation hardware described above makes it possible to eliminate these restrictions for translation and scaling in the AGT10, intended primarily for 2-D work, and also for rotation in the AGT30 and AGT50, which are capable of handling 3-D images.

Rotation of 3-D images

Figure 7 illustrates the projection onto a two-dimensional plane of a three-dimensional figure, in this case a cube, whose axes are rotated with respect to the axis system of the viewing plane. These equations describe rotation of a three-dimensional object:

$$x' = R_{11}x + R_{12}y + R_{13}z \quad (3)$$

$$y' = R_{21}x + R_{22}y + R_{23}z \quad (4)$$

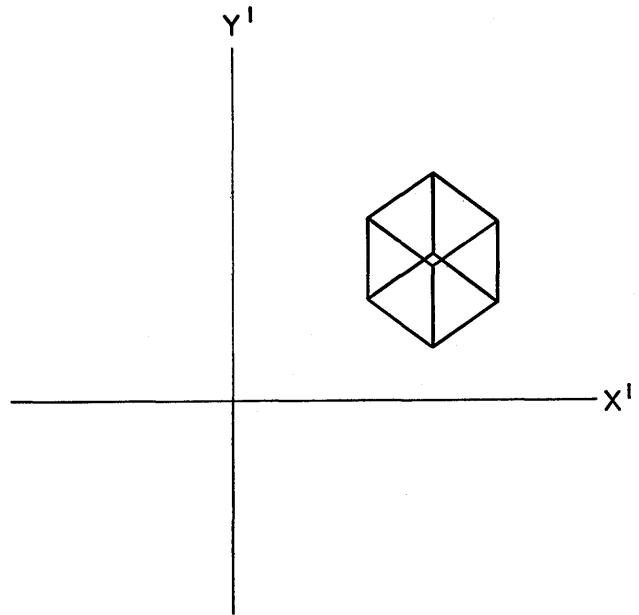


FIGURE 7—An orthogonal projection of a cube onto a two-dimensional viewing plane

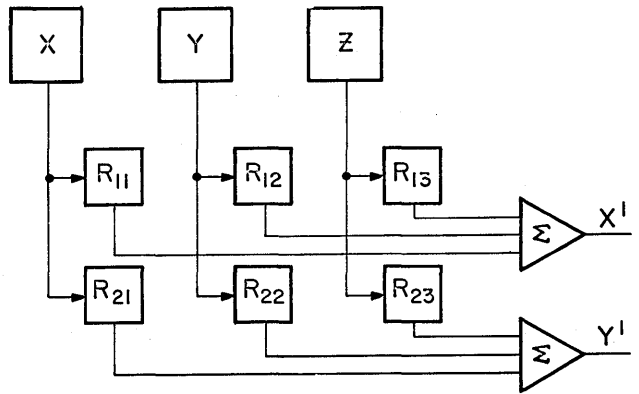


FIGURE 8—A hybrid array for rotation

A hybrid operator array appropriate for implementing these equations is shown in Figure 8. x, y and z are input coordinates describing the objects to be rotated. R11 through R23 are fixed coefficients specifying the rotation to which the input coordinates are to be subjected. x' and y' are the outputs representing the projection onto the x'y' plane of the xyz coordinates.

The hybrid array actually included in the AGT30 implements the following equation set:

$$x' = PS[x_a + SC(R_{11}x + R_{12}y + R_{13}z)] \quad (5)$$

$$y' = PS[y_a + SC(R_{21}x + R_{22}y + R_{23}z)] \quad (6)$$

$$z' = PS[z_a + SC(R_{31}x + R_{32}y + R_{33}z)] \quad (7)$$

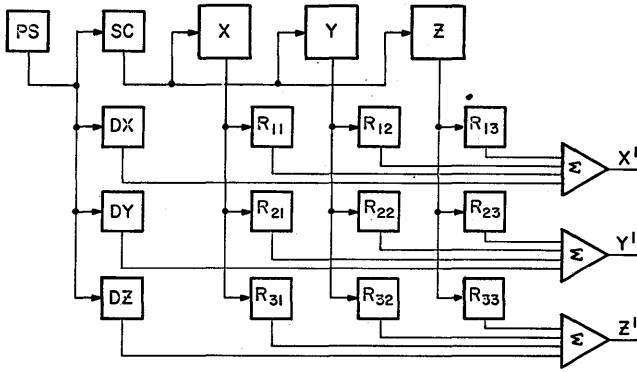


FIGURE 9—The hybrid array incorporated in the AGT30 and AGT50 for generalized scaling, translation and rotation

The block diagram of Figure 9 shows the organization of elements used to implement these equations. They are capable of providing nine multiplications for rotation, seven for scaling, and twelve summation operations, all within less than four microseconds. This is equivalent to sixteen multiples plus twelve adds in less than four microseconds, making it possible to display over 5,000 vectors subjected to arbitrary scaling and coordinate transformation at flicker-free rates.

Figure 10 shows some alphanumeric text which was generated programmatically (without the use of a character generator). Scaling and translation operations are performed by the array to place characters of appropriate size at appropriate locations on the screen.

The hybrid array is normally used to provide or-

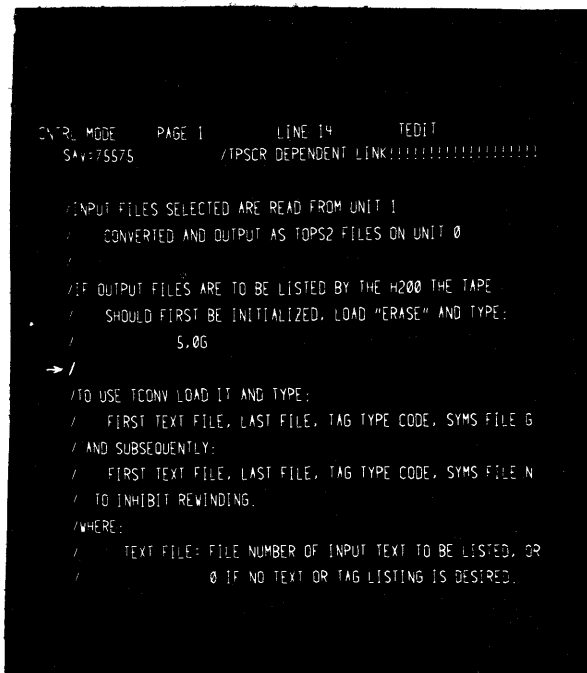


FIGURE 10—Programmatically generated text

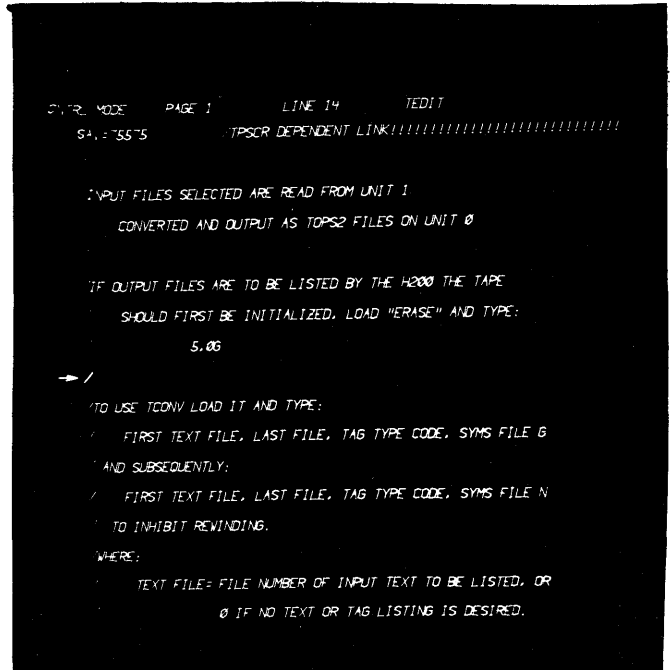


FIGURE 11—Programmatically generated text subjected to a skewed transformation

thogonal rotational transformations; however, it can also be used to perform various skewed transformations. The alphanumeric text shown in Figure 11 is derived from the same programmatic font as that of Figure 10, but in this case it has been passed through the array while the array contained the set of coefficients that implemented a skewed transformation to produce the italicized, sloping text.

Figure 12 shows 1,000 vectors of approximately ran-

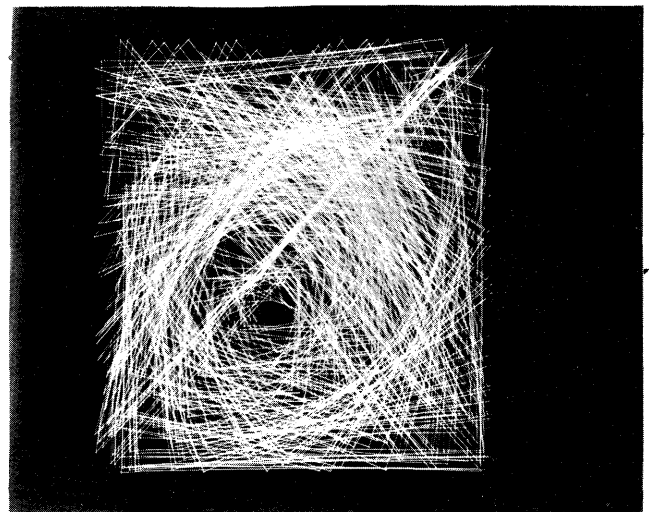


FIGURE 12—One thousand approximately random straight lines

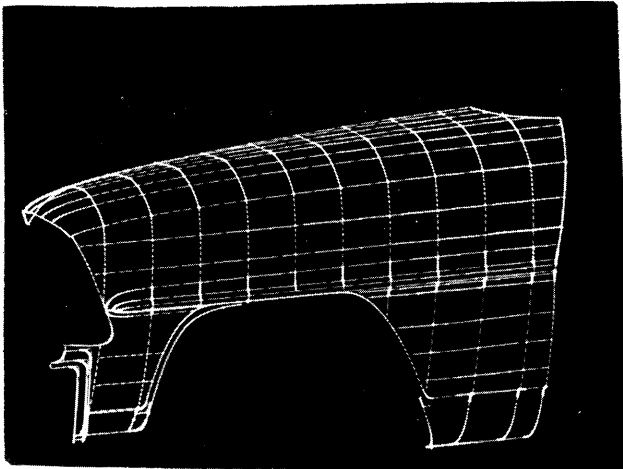


FIGURE 13—A CRT picture derived from an image described by approximately 2,500 short line segments

dom length and position. The system is capable of displaying pictures of this complexity with completely arbitrary scaling, translation and rotation.

The auto body part illustrated in Figure 13 consists of approximately 2,500 short vectors. This is about 50 percent of the limit of complexity that can be handled flicker-free by the system for images that are comprised of vectors of less than one-half inch in length.

Depth cues

The dynamic cues provided by rotating three dimensional objects and viewing their projection onto a two-dimensional plane do not completely resolve the question of what the shape looks like. Front-to-rear ambiguity occurs, as in the cube of Figure 7.

Variable intensity depth cueing has been incorporated in the AGT30 and AGT50 to help indicate picture depth by having the intensity of a line fall off as its z' coordinate goes into the viewing screen. This is a rather natural cue as it is a common experience that objects get dimmer and less clear and textures get finer as they get farther away; depth cueing is the application of this experience to a much smaller scale. The transfer function from z'/m to the intensity signal to the scope is shown in Figure 14. (Assume for now that operation takes place to the left of the $+10v$ cutoff.) I falls exponentially as z'/m decreases and is down 10:1 at $z'/m = 0$. The exponential falloff has been found to lead to equal perceived brightness ratios for equal distance ratios. This agrees with the well-known results that the eye's response to brightness stimuli is logarithmic over a wide range.

Some illustrations of the results obtained on a CRT are shown in Figures 15 and 16. When one stares at the

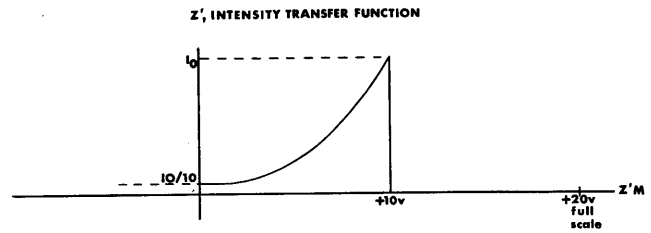


FIGURE 14—Beam intensity versus image depth for variable intensity depth cueing

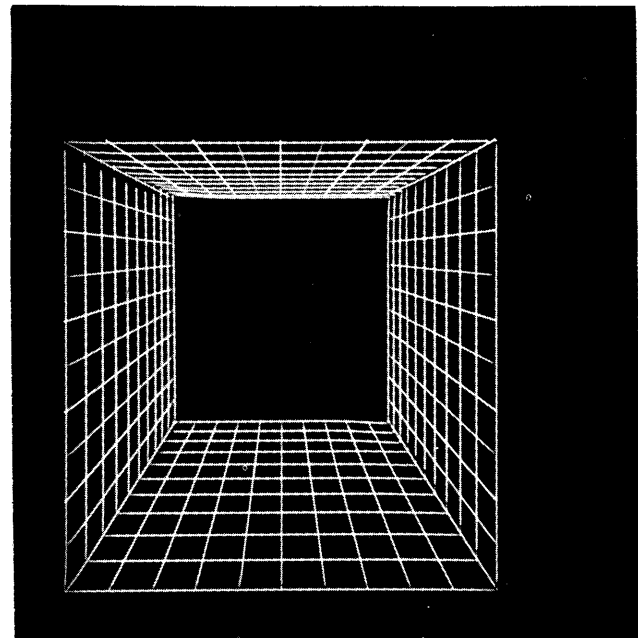


FIGURE 15—A picture without depth cueing

cube of Figure 7, it appears to flip from one orientation to another because the third dimension cue is missing. Without depth cueing, the object in Figure 15 appears to many people to be the top view of a pyramid. With depth cueing, as in Figure 16, the picture is much more clearly the perspective view of a rectangular "tunnel." Similar visual tricks occur in viewing orthogonal projections of rotating objects, and depth cueing is again found to reduce or eliminate the ambiguities which occur.

It has been suggested that equally important as a visual cue is the fact that the dimming of the line is accompanied by a narrowing of the line as well, thus giving a sort of linear perspective cue. While this perspective may provide some information about depth, the brightness depth cueing has been found to be valid when the observer is too far from the screen to distinguish differences in line width.

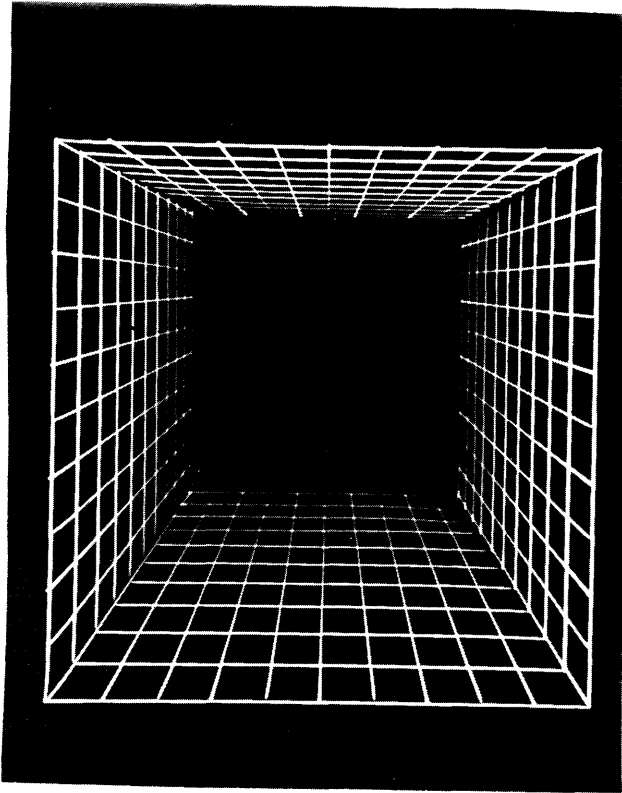


FIGURE 16—The picture of Figure 15, but with variable intensity depth cueing

Intensity windowing

The sharp cutoff at $z'/m = +10v$ in Figure 14 leads to another useful display phenomenon which we call z -windowing. An image can be moved back and forth across the z_0 boundary. Those portions of lines whose z -values are greater than z_0 are blanked while those portions behind z_0 are displayed. Thus one can examine thin sections of an image. This has been found useful as an alternative to the removal of hidden lines as a means for “de-cluttering” images composed of many line segments.

The ability to look at sections gives some interesting flexibility in viewing complicated objects. The z_0 cutoff plane can easily be moved by a manual, interactive device such as a joystick. Moving through an object section by section can give rapid insight into the shape and nature of the object. Sectioning accompanied by rotation of the object also gives some interesting and useful effects; it enhances depth perception by cutting the front portion as one quickly learns to associate the cut plane with the front part of the object.

The transformation from z' to z'/m allows further freedom in expanding or shrinking the picture to match not only the z_0 cutoff but the depth-cue slope as well. Thus the picture can be easily moved in front or in back

of z_0 without changing the data base; and the degree of depth-cue falloff can be varied at will from no perceptible change to extreme variations.

Removal of image portions in the “foreground” can also be used, with and without perspective transformations, to give the illusion of moving into the screen, i.e., objects move “behind” the observer and out of sight.

3-D windowing operator

A three dimensional windowing subsystem is available for the AGT in which upper and lower bounds can be placed (in digital registers) on x , y , and z . The vector generator then blanks whenever the beam goes beyond one of the bounds, and it also tells the program which bound was exceeded. This device finds use in a number of applications including uncluttering pictures, testing the dimensions and intersections of solids, and splitting the CRT screen up into rectangles allocated to different pictures, which can then move beyond the “edge” without encroaching upon its neighbor’s display space.

Software environment

A standard software package called AMOS has been developed to support the AGT. It includes a resident monitor available in two versions, their use depending on where the program library is stored. One version is suitable for use with magnetic tape storage, the other for use with disk. The monitor provides for on-line user control of all hardware and software. Control is exercised by control statements in which program entries and names of variables use the same symbols as those in the programmer’s source language. By typing the appropriate control statement, any subroutine in the program library can be loaded, linked and executed.

After initiating operation of a program, continued operator intervention is possible by use of a foreground/background mode, in which the monitor operates in the foreground on an interrupt basis while the user program operates in the background. The user can interact with the operating program in on-line fashion, interrogating memory contents and inspecting and changing parameter values. The user has access to all program entries and to all external variables and references by means of their symbolic names, as defined in the original source language programs.

A macro-assembler and a Fortran compiler permit preparation of source language programs either in symbolic machine language or ASA Basic Fortran. The macro-assembler can accept changes or additions to its own structure, so that the user who wishes to can alter it, departing from straight symbolic machine language

to devise his own problem-oriented language for particular graphics applications. Source language programs can be prepared for entry into the program library using an on-line text editor with which the user can scan selected files, display them on the CRT, and make additions, changes or deletions.

Graphics operators

Design of each of the software items mentioned briefly above is affected only slightly by the nature of the image manipulation hardware built into the AGT. A set of graphics software operators has been developed, however, whose design is much more intimately interwoven with the image manipulation capabilities of the system hardware. These graphics operators include a *display operator*, *save/retrieve operators*, a *build operator*, and a *freeze operator*. Some words about each of these is in order.

The display operator

The display operator is used to interpret data structures in core memory which represent simple or structured two- or three-dimensional images, and to display the resultant two-dimensional picture on the CRT. The data structure representing an image is *not* routinely translated to an unstructured display list for refreshing the CRT. Instead, it is left resident in core, and the display operator uses the coordinate transformation hardware to process it afresh with each successive frame to present the appropriate series of analog output values to the vector generator driving the CRT.

Each image is represented by sequences of variable length machine-word-lists called items. These may be logically linked, referenced, or looped into structures permitting hierarchical levels of processing with parametric and conditional control capabilities. The images are re-entrant and nestable for up to 16 levels of recursion.

Each item in an image describing data structure contains a command followed by an argument list. Each command has a field specifying the type of arguments contained in the argument list, and also an operation field which specifies the nature of the operation to be performed. Operations available for image description commands are of the following four types:

- Element generation
- Transform specification
- View definition
- Control operations

An Element Generating Operation specifies a particular kind of visual element in an image definition, such as a straight 3-D line or a string of packed 2-D strokes,

or a string of packed characters. The associated argument list references any parameters needed for generating the elements.

Transform Specifying Operations provide for scaling, translation and rotation operations which are applied to subsequent items of the image definition. Associated arguments are used to specify values of scale factor, displacement, and angular rotation. In general, Transform Specifying Operations affect all subsequent items in the image, and their effects are cumulative. There are some Transform Specifying Operations, however, whose function is to clear the effect of all previously established transforms.

View Defining Operations use their associated arguments to specify boundaries in the x-y plane and in depth outside of which the picture is blanked. This permits viewing only an arbitrarily specified portion of the image with provision for the detection of images that extend beyond the specified bounds.

Control Operations available for inclusion in image-defining data structures permit operations such as the following:

- Loop—Repetition of image items
- Sub-image—Reference to another image as a single item in the present definition (the image equivalent of subroutine jumps).
- End-image—Returns from current definition
- Save—Saves current transform and repeat count.
- Restore—Restores last-saved transform and repeat count.
- Jump—Branching in an image.
- Execute—Perform a computer subroutine (may be a necessary part of describing image portions which depend on values or states of real time inputs).
- Jump/Execute Conditional—Image branching or program execution conditional on pen, widow, parameter values, etc.

Most image commands require values for parameters which specify their action—number of degrees of rotation, scale factor values, etc. The referencing information necessary to access these values is contained in the arguments associated with each image item. Values can be accessed via several addressing modes, including immediate, direct, multi-level indirect and structured (where the address is an index to be added to the following address reference).

The build operator

Whereas the display operator outputs data to the CRT from image descriptions in core, it is the function of the build operator to facilitate the on-line creation and manipulation of such image descriptions. Its per-

mits composition of appropriately structured image descriptions from sub-images previously stored in an image library, or in response to user-generated inputs from light pen, joystick or data tablet. Changes in the image produced by the build operator are immediately (by the next frame) reflected in the CRT output display by the display operator, providing thereby a high degree of responsiveness to user inputs.

Images are built in a parametric form, where the user is free to set his own parameters. For example, in constructing a spoked wheel from straight line segments, the coordinate values of each line segment may be of no interest. Useful parameters might be wheel size and axle position. The wheel radius and center would be defined as formal parameters. The formal parameters used by the image being built may be set to a value or set to track a value, such as the setting of a variable control dial or the joystick. Thus, the wheel might be positioned with the joystick and its radius set with a variable control dial.

For the most part, the operator uses the build operator by sitting at the console and using the light pen and function switches. A menu of build operations is always displayed on the CRT. Three separate menus are used with the build operator: a menu of control operators, a menu of the current list of formal parameters, and a menu which is a portion of the current external symbol table, including all display image items.

Save/retrieve operators

These are standard programs used for filing and retrieving image structures using local mass storage.

The freeze operator

The structure of an image created by means of the build operator can easily be varied, and such an image will generally be economical for core memory, but it will usually take more time for processing by the display operator than the equivalent simple, unstructured image. The freeze operator permits the user, once he no longer requires the ease of varying a structured image, to transform it into a simple, unstructured image, perhaps for inclusion as a sub-image in a subsequent image. By saving a copy of the structured image prior to freezing, he can, of course, keep open the option to retrace his steps and retrieve the structured version of the image for further manipulation.

Interrupt hierarchy

Much of the operation of the AGT occurs in response to interrupts. The hierarchy of interrupt priorities is shown in Figure 17. The Display Clock, priority level 8, generates interrupts which are used to establish a frame rate of 60, 40, 30, 24 or 20 frames per second, as called for by the Display Operator.

The Vector Generator, priority level 11, generates interrupts in some of its operating modes, as a means of informing the digital processor that it is ready to accept the next vector.

CHANNEL	DEVICE
0	ARITHMETIC OVERFLOW
1	MAGNETIC TAPE
2	ALPHANUMERIC KEYBOARDS (UP TO 4)
3	MANUAL INTERRUPT FROM CONTROL PANELS
4	TELETYPE
5	COMMUNICATIONS DEVICES
6	DISK
7	CARD READER
8	DISPLAY CLOCK
9	WINDOWING OPERATOR
10	LIGHT PENS (UP TO 4)
11	VECTOR GENERATOR
12	CHARACTER GENERATOR
13	SPARE
14	PROGRAMMATICALLY INITIATED INTERRUPT
15 - 24	SPARE (LINE PRINTER, EVENT COUNTERS, ETC.)

FIGURE 17—Priority interrupt channel assignments

REFERENCES

- 1 M H LEWIN
An introduction to computer graphic terminals
Proceedings of the IEEE 1967 Vol 55 No 9 pp 1544-1552
- 2 J H MYER I E SUTHERLAND
On the design of display processors
Communications of the ACM June 1968 Vol II No 6 pp 410-414
- 3 T G HAGAN R TREIBER
Hybrid analog/digital techniques for signal processing applications
AFIPS Conference Proceedings 1966 Vol 28 p 379-388

A head-mounted three dimensional display*

by IVAN E. SUTHERLAND**

The University of Utah
Salt Lake City, Utah

INTRODUCTION

The fundamental idea behind the three-dimensional display is to present the user with a perspective image which changes as he moves. The retinal image of the real objects which we see is, after all, only two-dimensional. Thus if we can place suitable two-dimensional images on the observer's retinas, we can create the illusion that he is seeing a three-dimensional object. Although stereo presentation is important to the three-dimensional illusion, it is less important than the change that takes place in the image when the observer moves his head. The image presented by the three-dimensional display must change in exactly the way that the image of a real object would change for similar motions of the user's head. Psychologists have long known that moving perspective images appear strikingly three-dimensional even without stereo presentation; the three-dimensional display described in this paper depends heavily on this "kinetic depth effect."¹

In this project we are not making any effort to measure rotation of the eyeball. Because it is very difficult to measure eye rotation, we are fortunate that the perspective picture presented need not be changed as the user moves his eyes to concentrate on whatever part of the picture he chooses. The perspective picture presented need only be changed when he moves his head. In fact, we measure only the position and orientation of the optical system fastened to the user's head. Because the optical system determines the virtual screen position and

the user's point of view, the position and orientation of the optical system define which perspective view is appropriate.

Our objective in this project has been to surround the user with displayed three-dimensional information. Because we use a homogeneous coordinate representation,^{2,3} we can display objects which appear to be close to the user or which appear to be infinitely far away. We can display objects beside the user or behind him which will become visible to him if he turns around. The user is able to move his head three feet off axis in any direction to get a better view of nearby objects. He can turn completely around and can tilt his head up or down thirty or forty degrees. The objects displayed appear to hang in the space all around the user.

The desire to surround a user with information has forced us to solve the "windowing" problem. The "clipping divider" hardware we have built eliminates those portions of lines behind the observer or outside of his field of view. It also performs the division necessary to obtain a true perspective view. The clipping divider can perform the clipping computations for any line in about 10 microseconds, or about as fast as a modern high-performance display can paint lines on a CRT. The clipping divider is described in detail in a separate paper⁴ in this issue. Because the clipping divider permits dynamic perspective display of three-dimensional drawings and arbitrary magnification of two-dimensional drawings, we feel that it is the most significant result of this research to date.

In order to make truly realistic pictures of solid three-dimensional objects, it is necessary to solve the "hidden line problem." Although it is easy to compute the perspective positions of all parts of a complex object, it is difficult to compute which portions of one object are hidden by another object. Of the software solutions now available,^{2,5-10} only the MAGI⁹ and the Warnock¹⁰ approaches seem to have potential as eventual real-time solutions for reasonably com-

*The work reported in this paper was performed at Harvard University, supported in part by the Advanced Research Projects Agency (ARPA) of the Department of Defense under contract SD 265, in part by the Office of Naval Research under contract ONR 1866(16), and in part by a long standing agreement between Bell Telephone Laboratories and the Harvard Computation Laboratory. The early work at the MIT Lincoln Laboratory was also supported by ARPA.

**Formerly of Harvard University

plex situations; the time required by the other methods appears to grow with the square of situation complexity. The only existing real-time solution to the hidden line problem is a very expensive special-purpose computer at NASA Houston¹¹ which can display only relatively simple objects. We have concluded that showing "opaque" objects with hidden lines removed is beyond our present capability. The three-dimensional objects shown by our equipment are transparent "wire frame" line drawings.

Operation of the display system

In order to present changing perspective images to the user as he moves his head, we have assembled a wide variety of equipment shown in the diagram of Figure 1. Special spectacles containing two miniature cathode ray tubes are attached to the user's head. A fast, two-dimensional, analog line generator provides deflection signals to the miniature cathode ray tubes through transistorized deflection amplifiers. Either of two head position sensors, one mechanical and the other ultrasonic, is used to measure the position of the user's head.

As the observer moves his head, his point of view moves and rotates with respect to the room coordinate system. In order to convert from room coordinates to a coordinate system based on his point of view, a translation and a rotation are required. A computer uses the measured head position information to compute the elements of a rotation and translation matrix appropriate to each particular viewing position. Rather than changing the information in the computer memory as the user

moves his head, we transform information from room coordinates to eye coordinates dynamically as it is displayed. A new rotation and translation matrix is loaded into the digital matrix multiplier once at the start of each picture repetition. As a part of the display process the endpoints of lines in the room coordinate system are fetched from memory and are individually transformed to the eye coordinate system by the matrix multiplier. These translated and rotated endpoints are passed via an intermediate buffer to the digital clipping divider. The clipping divider eliminates any information outside the user's field of view and computes the appropriate perspective image for the remaining data. The final outputs of the clipping divider are endpoints of two-dimensional lines specified in scope coordinates. The two-dimensional line specifications are passed to a buffered display interface which drives the analog line-drawing display.

We built the special-purpose digital matrix multiplier and clipping divider to compute the appropriate perspective image dynamically because no available general-purpose computer is fast enough to provide a flicker-free dynamic picture. Our equipment can provide for display of 3000 lines at 30 frames per second, which amounts to a little over 10 microseconds per line. Sequences of vectors which form "chains" in which the start of one vector is the same as the end of the previous one can be processed somewhat more efficiently than isolated lines. Assuming, however, two endpoints for every line, the matrix multiplier must provide coordinate transformation in about 5 microseconds per endpoint. Each matrix multiplication requires 16 accumulating multiplications; and therefore a throughput of about 3,000,000 multiplications per second. The clipping divider, which is separate and asynchronous, operates at about the same speed, processing two endpoints in slightly over 10 microseconds. Unlike the fixed time required for a matrix multiplication, however, the processing time required by the clipping divider depends on the data being processed. The time required by the analog line generator depends on the length of the line being drawn, the shortest requiring about 3 microseconds, the longest requiring about 36 microseconds and an average of about 10 microseconds.

The matrix multiplier, clipping divider, and line-generator are connected in a "pipe-line" arrangement. Data "stream" through the system in a carefully interlocked way. Each unit is an independently timed digital device which provides for its own input and output synchronization. Each unit examines an input flag which signals the arrival of data for it. This data are held until the unit is ready to accept them. As the unit accepts a datum, it also reads a "directive" which tells it what to do with the datum. When the unit has accepted

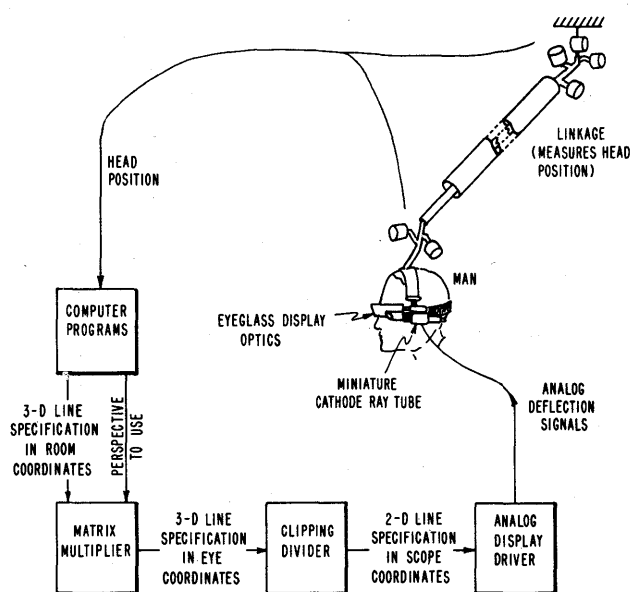


FIGURE 1—The parts of the three-dimensional display system

a datum, it clears its input flag. When it has completed its operation, it presents the answer on output lines and sets an output flag to signal that data is ready. In some cases the unit will commence the next task before its output datum has been taken. If so, it will pause in the new computation if it would have to destroy its output datum in order to proceed. Orderly flow of information through the system is ensured because the output flag of each unit serves as the input flag of the next. The average rate of the full system is approximately the average rate of the slowest unit. Which unit is slowest depends on the data being processed. The design average rate is about 10 microseconds per line.

The computer in this system is used only to process the head-position sensor information once per frame, and to contain and manipulate the three-dimensional drawing. No available general-purpose computer would be fast enough to become intimately involved in the perspective computations required for dynamic perspective display. A display channel processor serves to fetch from memory the drawing data required to recompute and refresh the CRT picture. The channel processor can be "configured" in many ways so that it is also possible to use the matrix multiplier and clipping divider independently. For example, the matrix multiplier can be used in a direct memory-to-memory mode which adds appreciably to the arithmetic capability of the computer to which it is attached. For two-dimensional presentations it is also possible to bypass the matrix multiplier and provide direct input to the clipping divider and display. These facilities were essential for debugging the various units independently.

Presenting images to the user

The special headset which the user of the three-dimensional display wears is shown in Figure 2. The optical system in this headset magnifies the pictures on each of two tiny cathode ray tubes to present a virtual image about eighteen inches in front of each of the user's eyes. Each virtual image is roughly the size of a conventional CRT display. The user has a 40 degree field of view of the synthetic information displayed on the miniature cathode ray tubes. Half-silvered mirrors in the prisms through which the user looks allow him to see both the images from the cathode ray tubes and objects in the room simultaneously. Thus displayed material can be made either to hang disembodied in space or to coincide with maps, desk tops, walls, or the keys of a typewriter.

The miniature cathode ray tubes mounted on the optical system form a picture about one half of an inch square. Because they have a nominal six tenths mil spot size, the resolution of the virtual image seen by the user is about equivalent to that available in standard

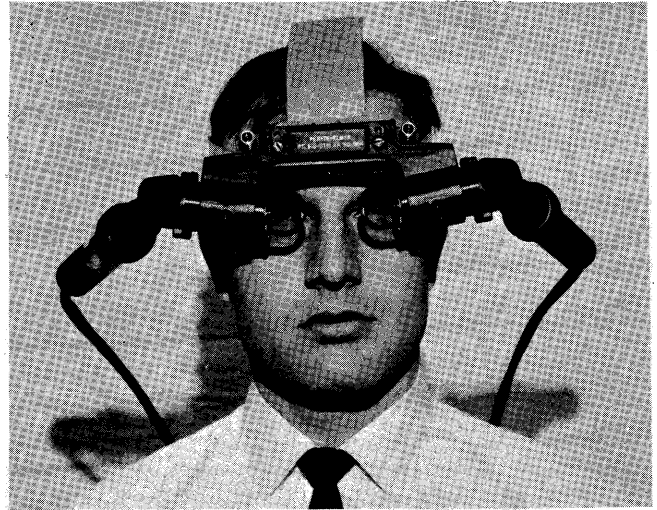


FIGURE 2—The head-mounted display optics with miniature CRT's

large-tube displays. Each cathode ray tube is mounted in a metal can which is carefully grounded to protect the user from shorts in the high voltage system. Additional protection is provided by enclosing the high voltage wiring in a grounded shield.

The miniature cathode ray tubes have proven easy to drive. They use electrostatic deflection and focussing. Because their deflection plates require signals on the order of only 300 volts, the transistorized deflection amplifiers are of a relatively straightforward design. Complementary-symmetry emitter followers are used to drive four small coaxial cables from the amplifier to each cathode ray tube. Deflection and intensification signals for the miniature cathode ray tubes are derived from a commercial analog line-drawing display which can draw long lines in 36 microseconds (nominal) and short lines as fast as three microseconds (nominal).

The analog line generator accepts picture information in the coordinate system of the miniature cathode ray tubes. It is given two-dimensional scope coordinates for the endpoints of each line segment to be shown. It connects these endpoints with smooth, straight lines on the two-dimensional scope face. Thus the analog line-drawing display, transistorized deflection amplifiers, miniature cathode ray tubes, and head-mounted optical system together provide the ability to present the user with any two-dimensional line drawing.

Head position sensor

The job of the head position sensor is to measure and report to the computer the position and orientation of the user's head. The head position sensor should pro-

vide the user reasonable freedom of motion. Eventually we would like to allow the user to walk freely about the room, but our initial equipment allows a working volume of head motion about six feet in diameter and three feet high. The user may move freely within this volume, may turn himself completely about, and may tilt his head up or down approximately forty degrees. Beyond these limits, head position cannot be measured by the sensor. We suspect that it will be possible to extend the user's field of motion simply by transporting the upper part of the head position sensor on a ceiling trolley driven by servo or stepping motors. Since the position of the head with respect to the sensor is known, it would be fairly easy to keep the sensor approximately centered over the head.

The head position measurement should be made with good resolution. Our target is a resolution of 1/100 of an inch and one part in 10,000 of rotation. Resolution finer than that is not useful because the digital-to-analog conversion in the display system itself results in a digital "grain" of about that size.

The accuracy requirement of the head position sensor is harder to determine. Because the miniature cathode ray tubes and the head-mounted optical system together have a pin-cushion distortion of about three per-

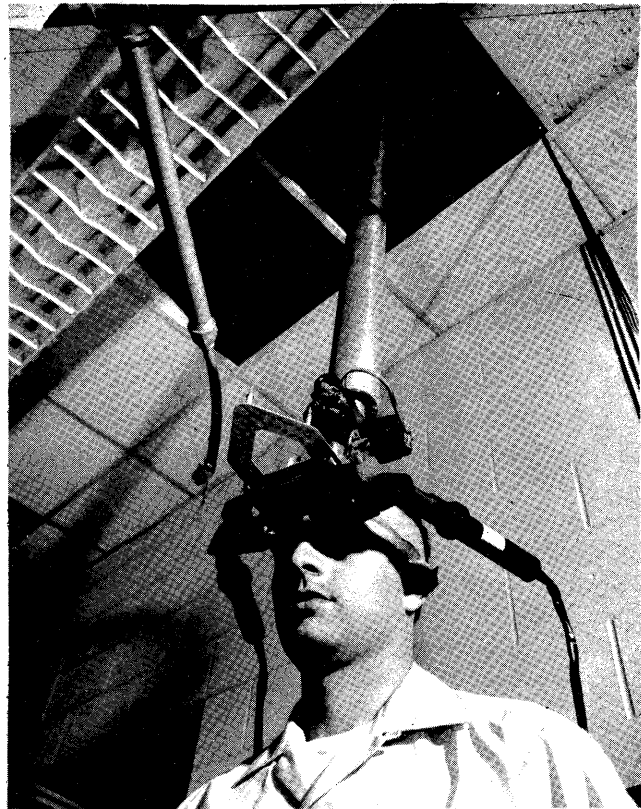


FIGURE 4—The ultrasonic head position sensor in use

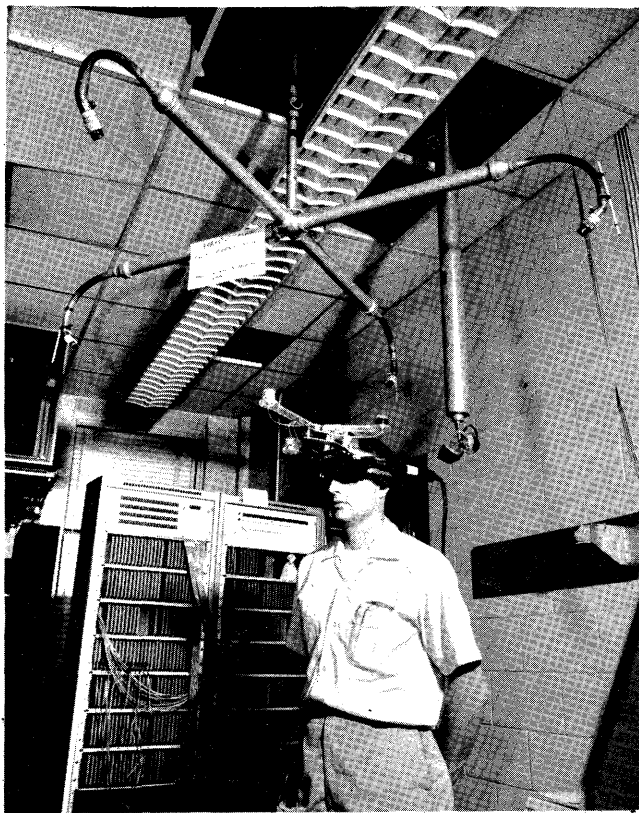


FIGURE 3—The mechanical head position sensor in use

cent, information displayed to the user may appear to be as much as three tenths of an inch out of place. Our head position sensor, then, should have an accuracy on the order of one tenth of an inch, although useful performance may be obtained even with less accurate head-position information.

We have tried two methods of sensing head position. The first of these involves a mechanical arm hanging from the ceiling as shown in Figure 3. This arm is free to rotate about a vertical pivot in its ceiling mount. It has two universal joints, one at the top and one at the bottom, and a sliding center section to provide the six motions required to measure both translation and rotation. The position of each joint is measured and presented to the computer by a digital shaft position encoder.

The mechanical head position sensor is rather heavy and uncomfortable to use. The information derived from it, however, is easily converted into the form needed to generate the perspective transformation. We built it to have a sure method of measuring head position.

We have also constructed a continuous wave ultrasonic head position sensor shown in Figure 4. Three transmitters which transmit ultrasound at 37, 38.6, and

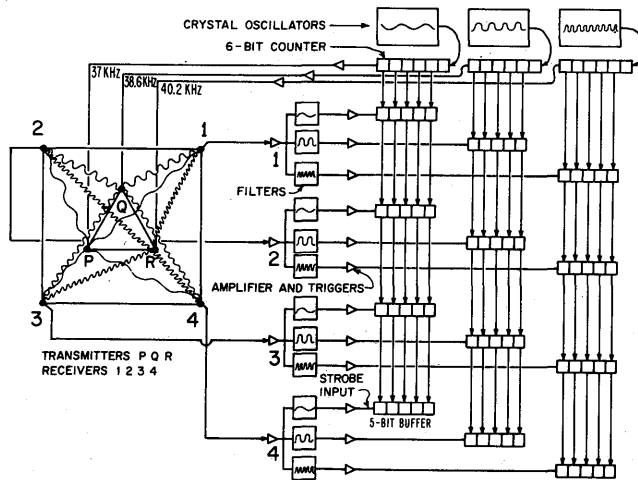


FIGURE 5—The ultrasonic head position sensor logic

40.2 kHz are attached to the head-mounted optical system. Four receivers are mounted in a square array in the ceiling. Each receiver is connected to an amplifier and three filters as shown in Figure 5, so that phase changes in sound transmitted over twelve paths can be measured. The measured phase shift for each ultrasonic path can be read by the computer as a separate five-bit number. The computer counts major changes in phase to keep track of motions of more than one wavelength.

Unlike the Lincoln Wand¹² which is a pulsed ultrasonic system, our ultrasonic head position sensor is a continuous wave system. We chose to use continuous wave ultrasound rather than pulses because inexpensive narrow-band transducers are available and to avoid confusion from pulsed noise (such as typewriters produce) which had caused difficulty for the Lincoln Wand. The choice of continuous wave ultrasound, however, introduces ambiguity into the measurements. Although the ultrasonic head position sensor makes twelve measurements from which head-position information can be derived, there is a wave length ambiguity in each of the measurements. The measurements are made quite precisely within a wave, but do not tell which wave is being measured. Because the wavelength of sound at 40 kHz in air is about 1/3 of an inch, each of the twelve measurements is ambiguous at 1/3 inch intervals. Because the computer keeps track of complete changes in phase, the ambiguity in the measurements shows up as a constant error in the measured distance. This error can be thought of as the "initialization error" of the system. It is the difference between the computer's original guess of the initial path length and the true initial path length.

We believe that the initialization errors can be resolved by using the geometric redundancy inherent in

making twelve measurements. We have gone to considerable effort to write programs for the ultrasonic head position sensor. These programs embody several techniques to resolve the measurement ambiguities. Although we have had some encouraging results, a full report on the ultrasonic head position sensor is not yet possible.

The perspective transformation

Generating a perspective image of three dimensional information is relatively easy. Let us suppose that the information is represented in a coordinate system based on the observer's eye as shown in Figure 6. If the two-dimensional scope coordinates, X_s and Y_s , are thought of as extending from -1 to $+1$, simple geometric reasoning will show that the position at which a particular point should be displayed on the screen is related to its position in three-dimensional space by the simple relations:

$$X_s = \frac{x'}{z'} \cotan \frac{\alpha}{2}$$

$$Y_s = \frac{y'}{z'} \cotan \frac{\alpha}{2}$$

If an orthogonal projection is desired, it can be obtained by making the value of z' constant. Because the perspective (or orthogonal) projection of a straight line in three-dimensional space is a straight line, division by the z' coordinate need be performed only for the endpoints of the line. The two-dimensional analog line-

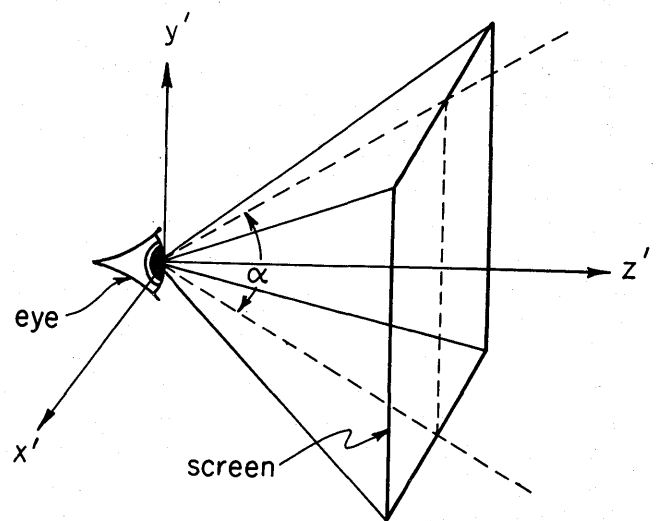


FIGURE 6—The x' y' z' coordinates system based on the observer's eye position

generating equipment can fill in the center portion of a three-dimensional line by drawing a two-dimensional line. The digital perspective generator computes values only for the endpoint coordinates of a line.

The three-dimensional information to be presented by the three-dimensional display is stored in the computer in a fixed three-dimensional coordinate system. Because this coordinate system is based on the room around the user, we have chosen to call it the "room" coordinate system. The drawing data in the room coordinate system is represented in homogeneous coordinates. This means that each three-dimensional point or end of a three-dimensional line is stored as four separate numbers. The first three correspond to the ordinary X Y and Z coordinates of three-dimensional space. The fourth coordinate, usually called W, is a scale factor which tells how big a value of X Y or Z represents a unit distance. Far distant material may thus easily be represented by making the scale factor, W, small. Infinitely distant points are represented by setting the scale factor, W, to zero, in which case the first three coordinates represent only the direction to the point. Nearby points are usually represented by setting the scale factor, W, to its largest possible value, in which case the other three coordinates are just the familiar fixed-point representations of X Y and Z.

The matrix multiplier

We have designed and built a digital matrix multiplier to convert information dynamically from the fixed "room" coordinate system to the moving "eye" coordinate system. The matrix multiplier stores a four-by-four matrix of 18 bit fixed-point numbers. Because the drawing data are represented in homogeneous coordinates, the single four-by-four matrix multiplication provides for both translation and rotation.² The matrix multiplier accepts the four 18 bit numbers which represent an endpoint, treating them as a four-component vector which it multiplies by the four-by-four matrix. The result is a four-component vector, each component of which is truncated to 20 bits. The matrix multiplier delivers this 80 bit answer to the clipping divider in approximately 5 microseconds. It therefore performs about three million scalar multiplications per second.

The matrix multiplier uses a separate multiplier module for each column. Each module contains an accumulator, a partial product register, storage for the four matrix elements in that column, and the multiplication logic. The entries of a row of the matrix serve simultaneously as four separate multiplicands. An individual component of the incoming vector serves as the common multiplier. The four multiplications for a

single row are thus performed simultaneously. For additional speed, the bits of the multiplier are examined four at a time rather than individually to control multiple-input adding arrays.

The clipping or windowing task

The job of the clipping divider is to accept three-dimensional information in the eye coordinate system and convert it to appropriate two-dimensional endpoints for display. If both ends of the line are visible, the clipping divider needs merely to perform four divisions, one for each two-dimensional coordinate of each end of the line. Enough equipment has been provided in the clipping divider to perform these four divisions simultaneously.

If the endpoints of a line are not within the observer's field of view, the clipping divider must decide whether any portion of the line is within the field of view. If so, it must compute appropriate endpoints for that portion as illustrated in Figure 7. Lines outside the field of view or behind the user must be eliminated. Operation of the clipping divider is described in a separate paper⁴ in this issue.

Like the matrix multiplier, the clipping divider is an independently-timed digital device which provides for its own input and output synchronization. It has an input and an output flag which provide for orderly flow of information through the clipping divider. If a line lies entirely outside the field of view, the clipping divider will accept a new input without ever raising its output flag. Thus only the visible portions of lines that are all or partly visible get through the clipping divider.

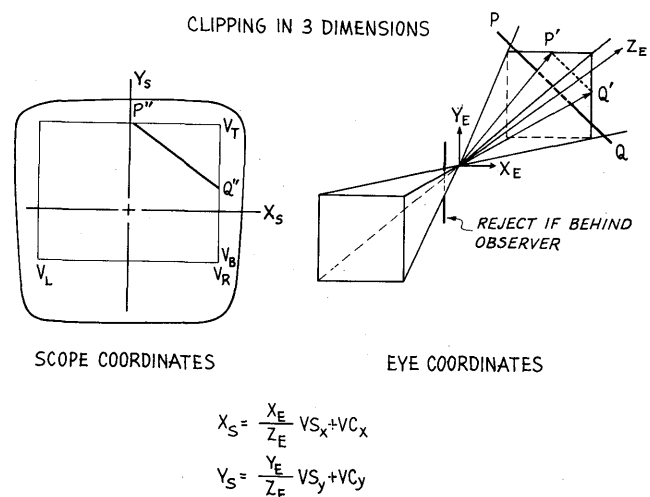


FIGURE 7—Clipping and perspective projection in three dimensions

Results

I did some preliminary three-dimensional display experiments during late 1966 and early 1967 at the MIT Lincoln Laboratory. We had a relatively crude optical system which presented information to only one of the observer's eyes. The ultrasonic head position sensor operated well enough to measure head position for a few minutes before cumulative errors were objectionable. The coordinate transformations and perspective computations were performed by software in the TX-2. The clipping operation was not provided: if any portion of a line was off the screen, the entire line disappeared.

Even with this relatively crude system, the three dimensional illusion was real. Users naturally moved to positions appropriate for the particular views they desired. For instance, the "size" of a displayed cube could be measured by noting how far the observer must move to line himself up with the left face or the right face of the cube.

Two peculiar and as yet unexplained phenomena occurred in the preliminary experiment. First, because the displayed information consisted of transparent "wire-frame" images, ambiguous interpretations were still possible. In one picture a small cube was placed above a larger one giving the appearance of a chimney on a house. From viewpoints below the roof where the "chimney" was seen from inside, some concentration was required to remember that the chimney was in fact further away than the building. Experience with physical objects insisted that if it was to be seen, the chimney must be in front.

A second peculiar phenomenon occurred during the display of the bond structure of cyclo-hexane as shown in Figure 8. Observers not familiar with the rippling hexagonal shape of this molecule misinterpreted its shape. Because their view of the object was limited to certain directions, they could not get the top view of the molecule, the view in which the hexagonal shape is most clearly presented. Observers familiar with molecular shapes, however, recognized the object as cyclo-hexane.

In more recent experiments with the improved optical system and vastly improved computation capability, two kinds of objects have been displayed. In one test, a "room" surrounding the user is displayed. The room is shown in Figure 9 as it would look from outside. The room has four walls marked N, S, E, and W, a ceiling marked C and a floor marked F. An observer fairly quickly accommodates to the idea of being inside the displayed room and can view whatever portion of the room he wishes by turning his head. In another test a small cube was displayed in the center of the user's operating area. The user can examine it from whatever side he desires.

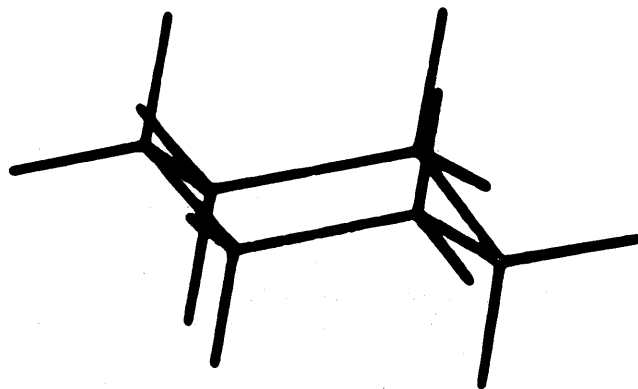


FIGURE 8—A computer-displayed perspective view of the cyclo-hexane molecule

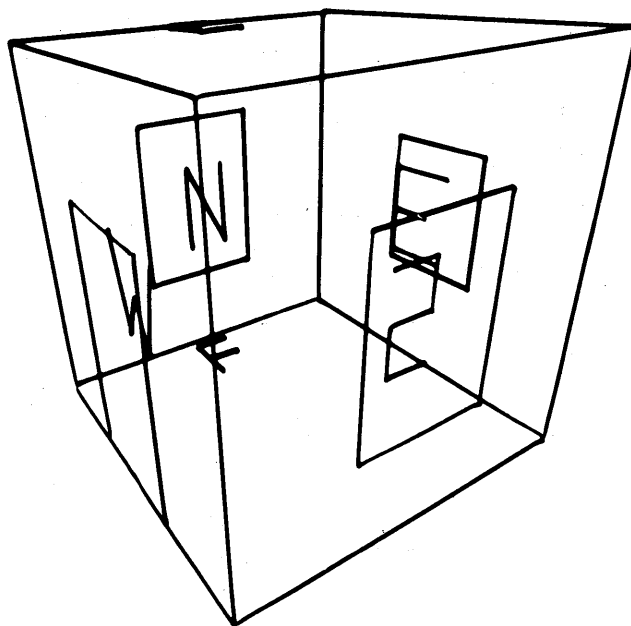


FIGURE 9—A computer-displayed perspective view of the "room" as seen from outside

The biggest surprise we have had to date is the favorable response of users to good stereo. The two-tube optical system presents independent images to each eye. A mechanical adjustment is available to accommodate to the different pupil separations of different users. Software adjustments in our test programs also permit us to adjust the virtual eye separation used for the stereo computations. With these two adjustments it is quite easy to get very good stereo presentations. Observers capable of stereo vision uniformly remark on the realism of the resulting images.

ACKNOWLEDGMENT

When I started work on the head-mounted display I had no idea how much effort would be involved. The project would have died many times but for the spirit of the many people who have become involved. The ultrasonic head-position sensor was designed and built at the MIT Lincoln Laboratory by Charles Seitz and Stylianos Pezaris and is available for our continued use through the cooperation of Lincoln Group 23. Seitz, as a Harvard employee, later designed the matrix multiplier. Robert Sproull, a most exceptionally capable Harvard Senior, simulated, designed most of, built parts of, and debugged the clipping divider. Two graduate students, Ted Lee and Dan Cohen have been an essential part of the project throughout. Our many arguments about perspective presentation, clipping, hidden-line algorithms, and other subjects form one of the most exciting educational experiences I have had. Ted Lee's programs to display curved surfaces in stereo have been the basis for many experiments. Cohen's programs to exercise the entire system form the basis of the demonstrations we can make. I would also like to thank Quintin Foster who supervised construction and debugging of the equipment. And finally, Stewart Ogden, so called "project engineer," actually chief administrator, who defended us all from the pressures of paperwork so that something could be accomplished.

REFERENCES

- 1 B F GREEN JR
Figure coherence in the kinetic depth effect
Journal of Experimental Psychology Vol 62 No 3 272-282 1961
- 2 L G ROBERTS
Machine perception of three-dimensional solids
MIT Lincoln Laboratory Technical Report No 315 May 22 1963
- 3 L G ROBERTS
Homogeneous matrix representation and manipulation of N-dimensional constructs
The Computer Display Review Adams Associates May 1965
- 4 R F SPROULL IE SUTHERLAND
A clipping divider
Proceedings of the Fall Joint Computer Conference 1968 this issue
- 5 D COHEN
A program for drawing bodies with the hidden lines removed
A term-project for course 6.539 MIT Fall 1965
- 6 H T HAYNES
A computer method for perspective drawing
Master's Thesis Texas A&M University Aug 1966
- 7 P LOUTREL
A solution to the hidden-line problem for computer-drawn polyhedra
New York University Technical Report 400-167 (Thesis) Bronx New York September 1967
- 8 A APPEL
The notion of quantitative invisibility and the machine rendering of solids
Proceedings of 22nd National Conference ACM
ACM Publication p 67 Thompson Book Company Washington DC 1967
- 9 Mathematical Applications Group Inc (MAGI)
3-D simulated graphics
Datamation February 1968
- 10 J E WARNOCK
A hidden line algorithm for halftone picture representation
University of Utah Technical Report 4-5 May 1968
- 11 Equipment installed at the Manned Space Craft Center at Houston Texas. The project is under the direction of the General Electric Company Electronics Laboratory under NASA Contract No NAS 9-3916
- 12 L G ROBERTS
The Lincoln wand
MIT Lincoln Laboratory Report June 1966
- 13 A C TRAUB
Stereoscopic display using rapid varifocal mirror oscillations
Applied Optics Vol 6 number 6 June 1967
- 14 P VLAHOS
The three-dimensional display: Its cues and techniques
Journal of the Society for Information Display Vol 2 Number 6 Nov/Dec 1965
- 15 R LAND IE SUTHERLAND
Real time color stereo computer displays
To be published in Applied Optics

A clipping divider*

by ROBERT F. SPROULL**

Stanford University
Stanford, California

and

IVAN E. SUTHERLAND**

The University of Utah
Salt Lake City, Utah

INTRODUCTION

When compared with a drawing on paper, the pictures presented by today's computer display equipment are sadly lacking in resolution. Most modern display equipment uses 10 bit digital to analog converters, providing for display in a 1024 by 1024 square raster. The actual resolution available is usually somewhat less since adjacent spots or lines will overlap. Even large-screen displays have limited resolution, for although they give a bigger picture, they also draw wider lines so that the amount of material which can appear at one time is still limited. Users of larger paper drawings have become accustomed to having a great deal of material presented at once. The computer display scope alone cannot serve the many tasks which require relatively large drawings with fine details.

On the other hand, a drawing can be represented in computer memory with very high resolution and precision. For example, if each coordinate is represented with 16 bits, a picture 65 inches square can be represented with resolution of about a thousandth of an inch. If such a picture could be displayed with its full resolution, it would be far better than can be provided on paper. Moreover, it is often convenient to represent coordinates in memory, for example in an 18 bit computer, with more than 10 bits used by the display scope, even though the additional resolution may not be seen in every view. With such great resolution available,

large and complex pictures can be represented which contain exceedingly fine details.

Unfortunately, the limitations of display equipment prevent a user from seeing the entire drawing and the fine detail simultaneously. But a sophisticated computer graphics system should provide for expansion of the picture so that any part of it can be examined in detail. The ability to expand the picture so that fine details are made visible partly compensates for the lack of resolution available in the display itself.

If the picture on the display is enlarged, parts of it may move off the screen. Programs to enlarge the drawing must compute not only the location of each part of the drawing after enlargement, but also which parts of the drawing are to appear at all. If all of a particular line or figure remains in view, it may simply be enlarged. If a figure or line moves entirely out of view, it must be eliminated from the picture. If a figure or line intersects the edge of the visible area, the part of it which is visible must be shown and the part of it outside the visible area must be eliminated.

The process of eliminating parts of a drawing which lie outside the observer's field of view has come to be known as "windowing."¹ One can think of the task as if one were looking at a large drawing through a small window, as shown in Figure 1. Everything that lies within the window should be shown, everything outside the window should be eliminated. If the window is made bigger, more material will be shown but will be correspondingly smaller on the display scope. If the window is made smaller, the material still inside it will appear on the screen correspondingly enlarged. Windowing is most difficult for parts of the drawing which are only partly visible.

There are two main methods used to accomplish

*The work reported in this paper was performed at Harvard University, supported in part by the Advanced Research Projects Agency (ARPA) of the Department of Defense under contract SD 265, and in part by the Office of Naval Research under contract ONR 1866(16).

**Formerly at Harvard University, Cambridge, Massachusetts.

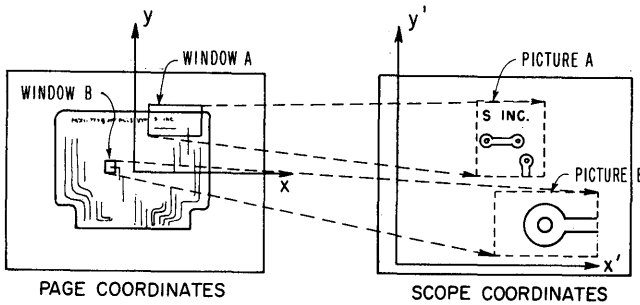


FIGURE 1—Magnification by looking at part of a large drawing through a small window

windowing: blanking and clipping. If blanking is to be used, the display scope itself must be blanked electronically whenever it is asked to display information outside the visible region. Systems which provide for blanking must provide not only for accurate display within the visible region, but also for deflection far outside the screen area. If the picture is to be very much larger than the actual scope area, the accuracy required of the electronic components involved may make them inordinately expensive. In any case, because the display must trace out both the visible and the blanked parts of the picture, the flicker rate will depend on the complexity of the total drawing regardless of how little may actually be seen.

Windowing may also be performed by clipping, the process of discovering which portions of a drawing are within the window and computing appropriate scope coordinates for them. If clipping is used, the display is given only valid visible information with the portions of the drawing outside the window already eliminated. For drawings composed of straight lines, clipping requires only enough arithmetic to compute the intersection of a line with the edge of the window. Because the clipping process requires many tests to decide whether a line intersects an edge of the window and if so which one, clipping programs are relatively slow. A typical clipping program takes between one and ten milliseconds per line clipped.

Because it is essential to perform windowing if drawings are to be enlarged, nearly all sophisticated computer graphics systems do windowing. They do a lot of windowing because the entire drawing must be processed each time the picture shown on the scope is moved or changed in *scelae*. Display equipment manufacturers are beginning to provide some hardware assistance to the windowing task, usually in the form of blanking capability. Yet windowing is still a problem because the methods previously available have been too slow. If blanking is used, the flicker rate of the display suffers; if clipping software is used, each motion or enlargement of the picture may cost several seconds of

delay. This paper describes a device which solves the windowing problem for the first time at a speed commensurate with high-performance line-drawing display equipment.

Clipping in two and three dimensions

Our clipping divider came about through a need to generate dynamic perspective images of three-dimensional objects. The head-mounted three-dimensional display project described elsewhere in this issue² calls for three-dimensional information to surround the observer. The clipping divider is necessary to perform the division required for a true perspective projection and to eliminate those parts of the three-dimensional drawing behind the observer or beside him but outside of the limited field of view provided by the display. The clipping divider has to operate fast enough to process information as it is displayed so that the picture can be updated as the user moves his head.

The material to be processed by the clipping divider is always a line or vector drawing. Each part of the drawing is made up of straight line segments specified either in terms of their absolute end point coordinates or in terms of the position of one end relative to the other. In the original three-dimensional perspective task, each absolute or relative specification is given in a three-dimensional coordinate system whose origin is at the user's eye. The three-dimensional drawing is specified with up to 20 bits for X, 20 bits for Y, and 20 bits for Z so that the resolution available at the clipping divider input is far higher than required by the scope. The high input resolution is needed whenever the observation point is placed very close to an object.

In a two-dimensional application the clipping divider accepts information about the lines or vectors of a large flat drawing. We think of the drawing as being written in memory on a large "page" of paper and call its coordinate system "page coordinates" to contrast them with "scope coordinates." The clipping divider will present on the scope only the part of the drawing within the "window." The window is a rectangle on the drawing aligned with the coordinate axes. The window is specified by giving the page coordinates of its left, right, bottom, and top edges, up to 20 bits each. These four numbers are stored internally in the "window" registers. Each XY location in the drawing may be specified in page coordinates with up to 40 bits, 20 for each axis. The coordinates stored in memory can be in a form suitable for computation rather than packed in a way peculiar to the display. In an 18 bit machine, for example, the two least significant bits of the 20 bit clipper input are made to be a copy of the sign bit so that each coordinate occupies a single word of storage. Coordinates can be treated with the ordinary arithmetic instructions of the computer. There is no need to pack X

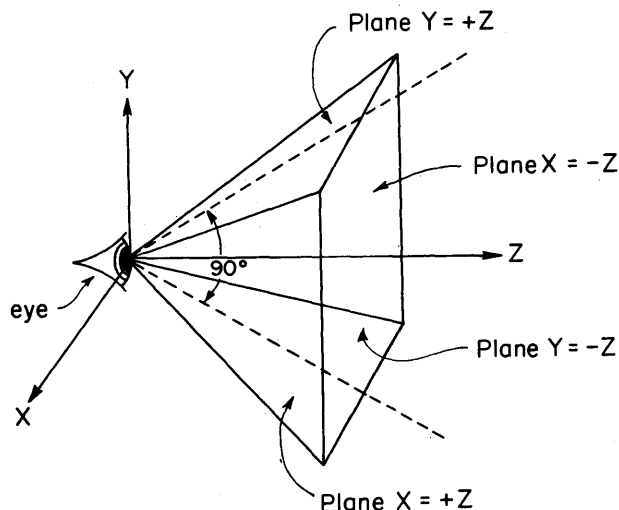


FIGURE 2—Only material inside the pyramid is visible

and Y information into a single word for use by the display. The 20 bit input resolution is useful because the clipping divider can magnify a portion of the drawing to show on the display scope.

Whereas clipping in two dimensions is by now a fairly familiar process, how to do clipping for perspective projections is less widely known. In order to present a perspective picture of material which surrounds the observer, clipping must be done in three dimensions before doing the perspective division. Clipping must precede division because the unclipped ends of three-dimensional lines may have negative or zero values of Z . Division by a negative Z value will give an erroneous position on the wrong side of the picture; division by zero or too small a value of Z will cause overflow.

In three-dimensional applications the region within which lines are visible is a pyramid whose vertex is at the eye. The left, right, bottom, and top edges of this "pyramid of vision" are the planes $X = -Z$, $X = +Z$, $Y = -Z$, and $Y = +Z$ respectively, as shown in Figure 2. The clipping process in three dimensions involves computing the intersection of each line with these four planes. The pyramid of vision encompasses a 90 degree field of view. Scaling before the clipping process can provide for other viewing angles

The clipping divider maps whatever drawing information falls within the pyramid of vision or the window onto a portion of the scope face. The portion of the scope within which information is presented is a rectangle aligned with the axes of the scope. The size and position of this rectangle, or "viewport," is specified by giving the scope coordinates of its left, right, bottom, and top edges, as shown in Figure 3. These four numbers are stored internally in the "viewport" registers. If three-dimensional information is being presented, the

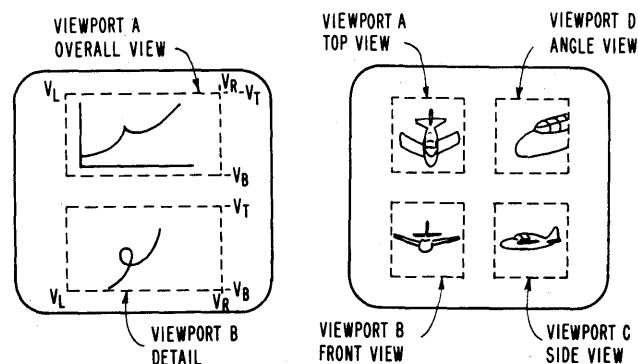


FIGURE 3—Multiple viewports in two and three dimensions

viewport will contain a perspective picture of the part of the three-dimensional drawing which falls within the field of view. If two-dimensional information is being presented, the viewport will contain an enlarged version of the information which falls within the window. The ability to map information onto a part of the scope face rather than all of it is important if several views of a single drawing are to be presented simultaneously.^{3,4}

The midpoint algorithm

The clipping divider utilizes a new algorithm for solving the windowing problem. We call this algorithm the "midpoint" algorithm because it involves computing the midpoint of the line. The midpoint is easily found by adding together the endpoint coordinates and shifting the sum right one bit. If implemented in software, the midpoint algorithm would be slower than a direct geometric computation of the intersection of the line and the edge of the window. Hardware which implements the algorithm, however, is able to capitalize on the fact that additions are much easier to perform than either multiplication or division.

The clipping divider distinguishes three kinds of lines:

- 1) lines with neither end in view,

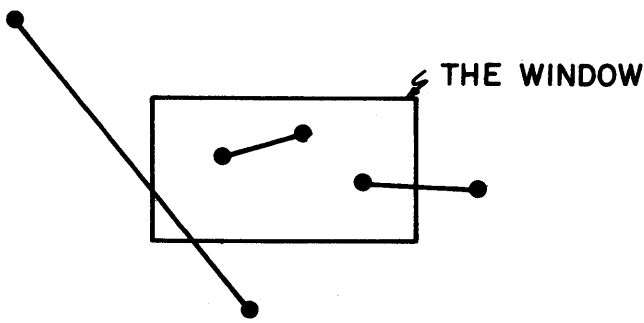


FIGURE 4—Three end point cases

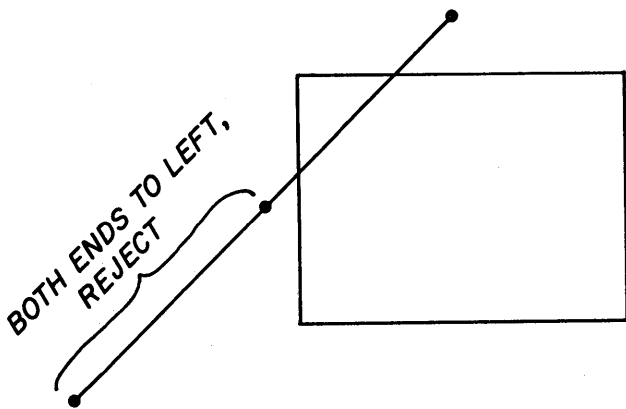


FIGURE 5—If midpoint is not within the window, one half can always be rejected

2) lines with only one end in view, and
 3) lines with both ends in view,
 as shown in Figure 4. In each case, the operations performed reduce the line to a shorter line of a simpler case.

For lines of the first case with neither end in view, we check to see if some portion of it could possibly be in view. Obviously if both ends of the line are:

- a) to the right of the window,
- b) above it,
- c) to the left of it, or
- d) below it

then no portion of the line could possibly be seen and the line can be rejected. In the three-dimensional case, some time can be saved by also rejecting lines if both start and end have negative Z values. If the line passes this "trivial test," we compute its midpoint.

The midpoint of the line is either inside the window or outside. If the midpoint is inside the window, we can treat the line as two segments of case two, each of which has one end, the common midpoint, in view. If the midpoint is not within the window, it divides the

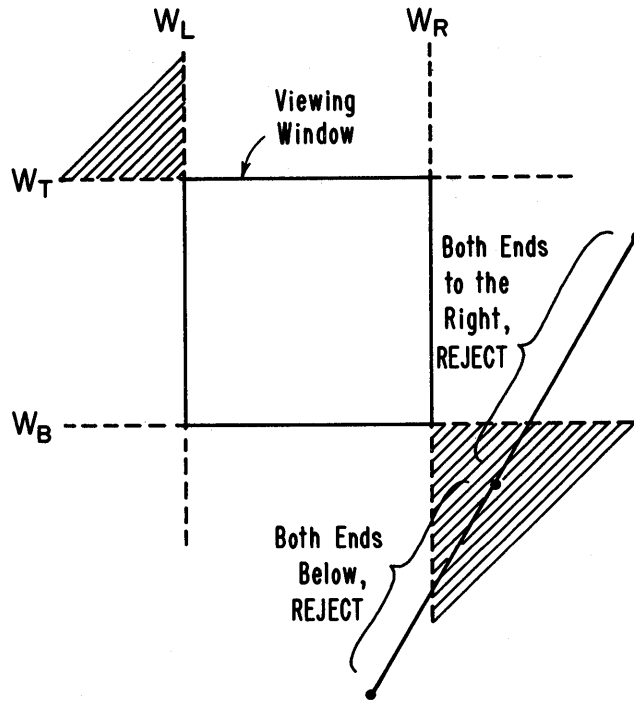
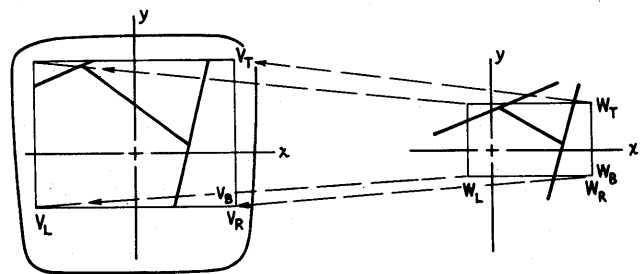


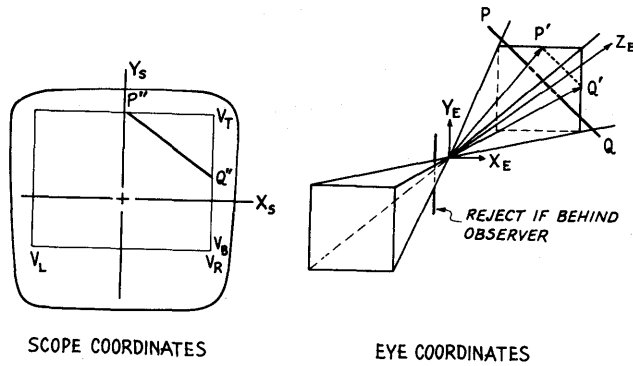
FIGURE 6—Simple rejection criteria for positive slope lines

line into two pieces, only one of which can possibly pass through the window. As shown in Figure 5, the trivial test on each of the pieces tells which to reject, leaving a shorter line neither of whose ends is in view. If the trivial test indicates that both halves should be rejected, no part of the line passes through the window. Thus lines with positive slope will be rejected if any point is detected within the regions shown shaded in Figure 6. Lines with negative slope will be rejected if



SCOPE COORDINATES		PAGE COORDINATES
$VC_x = \frac{V_R + V_L}{2}$	$X_s = \frac{X_p - WC_x}{WS_x} VS_x + VC_x$	$WC_x = \frac{W_R + W_L}{2}$
$VC_y = \frac{V_T + V_B}{2}$	$Y_s = \frac{Y_p - WC_y}{WS_y} VS_y + VC_y$	$WC_y = \frac{W_T + W_B}{2}$
$VS_x = \frac{V_R - V_L}{2}$		$WS_x = \frac{W_R - W_L}{2}$
$VS_y = \frac{V_T - V_B}{2}$		$WS_y = \frac{W_T - W_B}{2}$

FIGURE 7—Clipping in 2 Dimensions



$$X_S = \frac{X_E}{Z_E} VS_x + VC_x$$

$$Y_S = \frac{Y_E}{Z_E} VS_y + VC_y$$

FIGURE 8—Clipping in 3 Dimensions

any point is found within similar regions at the other corners.

For lines of the second case with only one end of the line in view, we again compute the midpoint of the line. If the midpoint is outside the window, half of the line can be eliminated. If the midpoint is inside the window, it is closer to the edge of the window than the original point and still, of course, on the line. We continue to compute midpoints within the segment which intersects the window edge to make a logarithmic search for the place where the line penetrates the edge of the window.

Finally, having reduced the line to the third case where both ends are within the window, albeit on the edge, we convert these endpoint coordinates to coordinates suitable for display on the scope. This conversion involves division by the window size in two dimensions or by the Z depth in three dimensions, and multiplication by an appropriate factor to account for the size and position of the viewport. These conversions are shown in Figures 7 and 8. Because the points used in the division are guaranteed to be within the window or pyramid of vision, overflow will never occur.

Window-edge coordinates

During the clipping process, information about a line is represented in a special coordinate system based on the edges of the window. Each point is represented as four numbers, each of which tells how far the point is from one edge of the window. These four numbers can be thought of as a four-component vector whose components are given by:

$$[X - W_L, X - W_R, Y - W_B, Y - W_T]$$

in two dimensions, or

1001	0001	0101	1001	1101	0101
1000	0000	0100	1011	1111	0111
1010	0010	0110	1010	1110	0110

FIGURE 9—Values of the “out code” in and around the window for positive Z (left) and negative Z (right)

$$[X - (-Z), X - (+Z), Y - (-Z), Y - (+Z)]$$

in three dimensions. This “window-edge” coordinate system makes it very easy to tell if a point is inside or outside of the window. The ordinary coordinates of the point are easily retrieved from its window-edge representation by adding or subtracting components.

The sign bits of the four components of the window-edge representation contain all the information required to test the position of a point relative to the window or pyramid of vision. If the signs of the four components of the window edge representation are + - + - respectively, the point is visible. For any other combination of signs, the point is outside the viewing region. We have found it convenient to think of the position of a point relative to the window in terms of a simple four bit code derived from the signs of the window-edge representation. This four bit “out code,”

$$OC = [S_L, \bar{S}_R, S_B, \bar{S}_T]$$

The sign bits of the right and top components have been complemented so that “one” indicates a position outside the window. Figure 9 shows the out codes for different positions around the window or pyramid of vision.

Whether or not a line should be rejected can be determined by the logical intersection of the four-bit “out codes” for its start (subscript s) and end (subscript e). If both start and end are to the left of the window, for example, both out codes will have a “one” in their first component, their intersection will be non-zero, and the line can be rejected. The trivial rejection criterion is thus:

$$\text{If } (OC_s \text{ and } OC_e) \neq 0 \text{ then reject.}$$

Similarly, if the mid-point of a line is not on the screen, the intersection of the out codes for the start, end, and midpoint tells which part of the line to reject.

$$\text{If } (OC_s \text{ and } OC_m) \neq 0 \text{ then reject first half.}$$

$$\text{If } (OC_e \text{ and } OC_m) \neq 0 \text{ then reject second half.}$$

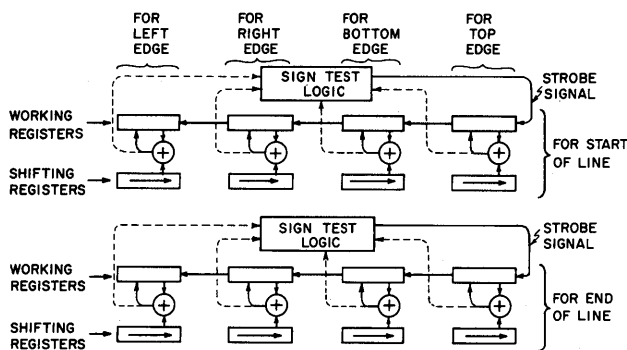


FIGURE 10—Hardware configuration for clipping

If both halves of the line are rejected, of course, the line is completely eliminated, as was shown in Figure 6. The same "out code" tests serve for both the two- and three-dimensional clipping. In the three-dimensional case, lines which pass behind the observer are automatically rejected by the same tests.

Hardware

The clipping divider contains eight individual adders arranged in two groups of four. Each adder has associated with it a working register and a shifting register, as shown in Figure 10. One group of four adders is used to determine where the line enters the field of view, the other group of four adders is used to determine where the line leaves the field of view. After the clipping process is complete, the adders are rearranged into four groups of two. Each pair of adders does a division and multiplication to provide for scaling in two dimensions or perspective division in three dimensions. The clipping divider also contains registers which hold the position of the window edges and the viewport edges, and the coordinates of a starting position to use for the line.

The clipping operation is begun by loading the working registers with the window edge coordinate representation of the two ends of the line. If the line is specified as a relative vector, the specified displacements are added to the starting coordinates to find the end of the line. Enough adders are available to do the addition of all coordinates simultaneously. The shifting or "delta" registers are loaded with the displacement required to go from one end of the line to the other. If absolute specifications for the ends of the line were given, this displacement is computed as their difference. When the setup process is finished, each group of adders has been provided information about the absolute coordinates of its end of the line and the displacement required to get to the other end.

The possibility of trivial rejection is checked next. If

both ends are outside the window, their "out codes" may show that the line can be trivially rejected. If both ends are in the window, the clipping process can be omitted. If the line is a non-trivial case, computation proceeds.

Each step in the clipping process involves shifting the delta registers one place to the right. After the first shift, the delta registers contain the displacement required to reach the midpoint of the line. The adder output will then be the midpoint coordinates. If the "out code" tests described in the previous section are satisfied, the midpoint value replaces the endpoint value in the working registers. If the out code tests fail, the midpoint is discarded. Accepting the midpoint as a replacement for the endpoint is equivalent to eliminating the half of the line next to the endpoint. Because each step starts by shifting the delta registers one place to the right, each step considers a line that is half as long as the previous one. The clipping process is complete when the delta registers contain zero.

The clipping process as implemented here is essentially a vector version of ordinary division. A "quotient" could be generated by recording successive "zeros" or "ones" according to the acceptance or rejection of each successive midpoint. The quotient would be a binary fraction representing the ratio of the length of the clipped-off part of the line to the total length of the line. Because we do not want the quotient, we do not bother to record it. We are only interested in the coordinates of the window edge intersection. Notice that when the clipping process is complete, the component of the window edge specification which corresponds to the edge intersected will be zero. Scalar division hardware also reduces the numerator to zero. Unlike a scalar division, however, the vector division described here has other components which are non-zero. These other components carry the answer information we want.

After the clipping process is complete, the working registers contain the ends of the visible segment of the line in window edge coordinates. These coordinates must be converted to the appropriate scope coordinates to position the displayed line properly on the scope picture. The sum of pairs of window edge coordinates can be used to find the position of the point relative to the center of the window, WC. For example:

$$\begin{aligned}(X - W_L) + (X - W_R) &= 2X - (W_L + W_R) \\ &= 2(X - WC_X).\end{aligned}$$

In two dimensions, the difference of pairs of window edge coordinates can be used to find the size of the window, WS, for scaling.

$$(X - W_L) - (X - W_R) = (W_R - W_L) = 2(WS_X).$$

In three dimensions the difference can be used to find the depth information, Z , for perspective division.

$$(X - (-Z)) - (X - (+Z)) = 2(Z) .$$

Thus in both two and three dimensions the clipping divider divides the sums of pairs of window edge coordinates by their differences.

The transformation used in going from the clipped endpoints to the scope also involves the size and position of the viewport, as was shown in Figures 7 and 8. The transformations involve both division by the window size or Z coordinate and multiplication by the viewport size. The division and multiplication are performed simultaneously by pairs of coupled adders. One adder with its associated shifting and working registers is used as an ordinary scalar divider. The other adder with its working and shifting registers provides for the multiplication. Instead of recording the bits of the quotient as they are generated by the divider, the bits are used immediately to control addition of the multiplicand to the accumulating product in the multiplier, as shown in Figure 11. If the sign test in the scalar division is successful, the output of both adders replaces their respective working registers. This simultaneously provides a new dividend for the next trial, and a new partial product closer to the answer.

Interfacing for the clipper

The clipping divider was designed to be part of a complete display system. The clipping divider is provided input data by a memory interface channel which fetches the data from memory. The channel is capable of interpreting codes in the information it gets from memory as special instructions, some of which it uses to direct the actions of the clipper. The clipping divider delivers its output to an ordinary two-dimensional line-drawing display. The clipping divider is intended only to provide the very considerable arithmetic capability required between memory and the display scope.

The clipping divider is an independent separately-timed, digital computing device. It watches an input flag which is raised whenever input data are available. As soon as its previous task is complete, it will accept the input data and clear the input flag. Along with the input data it accepts a 16 bit "directive" which indicates what the clipping divider is to do with the data. If no output is generated as a result of the task assigned, the clipping divider returns to a waiting state until the input flag is again raised. If some output is generated, the clipping divider raises an output flag and waits for the output to be accepted before it will again accept input data.

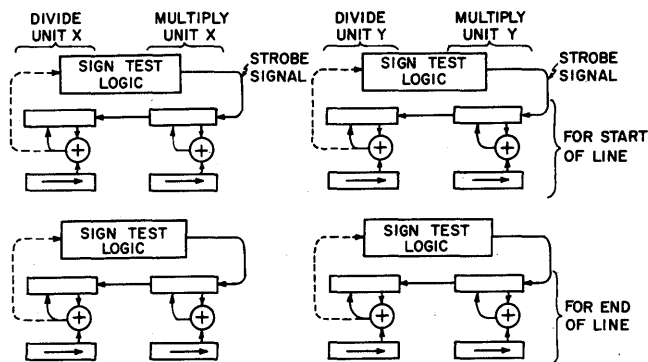


FIGURE 11—Hardware configuration for scaling

The clipping divider was designed to run with a 250 nanosecond clock. It takes 42 clock cycles to complete a worst case clipping task which at design speed would be ten and one half microseconds. Many tasks, however, take far less time. Trivial rejection of a line, for example, takes only 3 clock cycles. Non-trivial rejection of a line takes between 5 and 21 clock cycles, depending on how many steps are required to determine that the line can be rejected. If both ends of the line are inside the window, clipping can be omitted, and only the scaling or perspective division need be done. If both ends of the line are visible, the clipping divider will scale it correctly in 25 clock cycles. As this is written, the clipper is operating at about half of the design speed.

Windows within windows

It is often useful to structure information for display on a CRT. Subpictures or symbols, such as a resistor in an electrical drawing, an integral sign in a mathematical expression, or a standard part on a mechanical drawing need only be stored in computer memory once. A symbol may be displayed in many different positions on the CRT by multiple calls on its single definition just as a program subroutine may be called from many places. The memory interface channel which delivers data to the clipping divider keeps track of subroutine returns, nesting of subpictures, and previous window or viewport sizes.

The process of displaying a subpicture on the scope can be thought of as the combination of two transformations. The first transformation will place a replica of the symbol, possibly reduced in size, on the working drawing in memory. The second transformation will paint a picture on the scope, possibly with magnification, of the symbol on the drawing. The first transformation maps all the information inside a "master" rectangular area of the definition space onto a rectangular region or "instance" rectangle within which the symbol is to appear on the page. The second transformation maps whatever

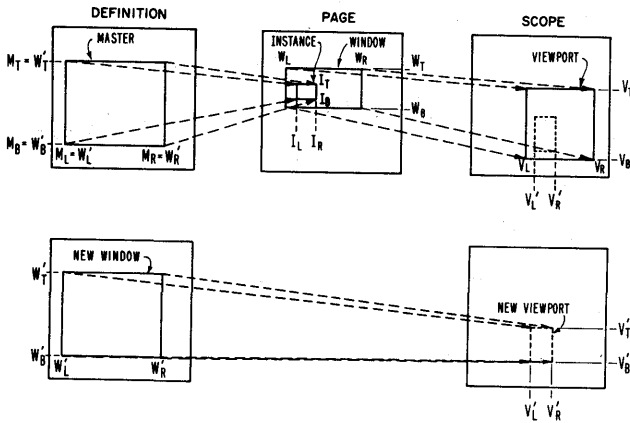


FIGURE 12—The two transformations for a subpicture (top) can be replaced by a single transformation (bottom)

portion of the instance is visible within the current window area onto the appropriate viewport on the scope face. These two transformations are shown at the top of Figure 12.

The clipping divider can perform both transformations required for displaying subpictures in a single step. If the structure of the drawing calls for nested subpictures, the entire set of transformations required by any nesting can be handled at once. The scaling transformation used in the clipping divider,

$$X' = \frac{X - WC_X}{WS_X} VS_X + VC'_X,$$

is rich enough so that any combination of two or more such transformations is a single transformation of the same form. All that is required to combine transformations when entering a display subroutine is to compute and load the new composite clipping limits.

The job of computing new clipping limits involves many tests to distinguish between cases. For example, suppose a portion of the symbol is inside the window and a portion is outside. During the display of such a symbol the clipping divider should use a new window, W' , and a new viewport V' , each smaller than before. On the other hand, if the symbol is entirely inside the window, only the viewport will be reduced in size. If none of the symbol is visible, then there will be no new clipping limits and, in fact, there is no reason to do the subroutine at all. Some examples of these cases are shown in Figure 13.

Because the computation of clipping limits is similar in kind to the clipping computations for lines, it is easily implemented with the same hardware. A special control mode has been provided for performing this computation. The memory interface channel stores the previous window and viewport values in a pushdown

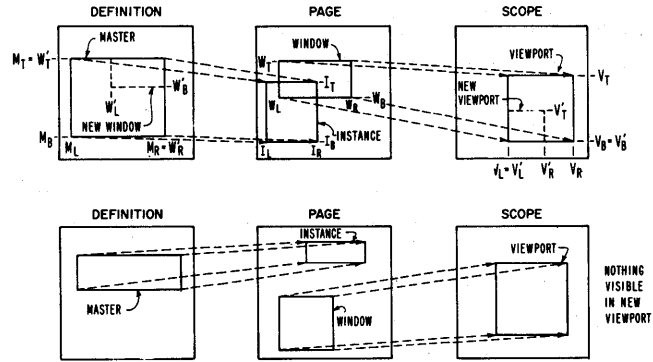


FIGURE 13—Finding the new window (W') and viewport (V') from instance (I) and master (M)

stack prior to display of a symbol. The channel then gives the clipping divider data from memory describing the size and position of the symbol. The clipping divider can establish the new composite clipping limits automatically.

The matrix multiplier

We have also designed and built a digital four-by-four matrix multiplier. Because we represent data in homogeneous coordinates, a single four-by-four matrix multiplication can account for both translation and rotation. Together, the matrix multiplier and clipping divider can present perspective views of three dimensional objects tumbling in real time. The combination can also be used to display curves as will be described in the next section.

The matrix multiplier uses a separate multiplier module for each column of the matrix. Each module contains an accumulator, a partial product register, storage for the four matrix elements in that column, and the multiplication logic. The entries of a row of the matrix serve simultaneously as four separate multipliers. An individual component of the incoming vector serves as the common multiplier. The four multiplications for a single row are thus performed simultaneously. For additional speed, the bits of the multiplier are examined four at a time rather than individually to control multiple-input adding arrays.

Display of curves

The matrix multiplier and clipping divider can be used for generating a wide variety of curves quite rapidly. Suppose, for example, that we have stored in computer memory a collection of vectors of the form $[t^3, t^2, t, 1]$ for which t varies uniformly from 0 to 1. The first of these vectors will, of course, be $[0, 0, 0, 1]$ and the final one will be $[1, 1, 1, 1]$. If these vectors are

multiplied by a particular four by four matrix, as shown below, the resulting vectors will be cubic polynomials in t where the coefficients of the polynomials are the entries of the matrix.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} = \begin{bmatrix} (a_{11} t^3 + a_{12} t^2 + a_{13} t + a_{14}) \\ (a_{21} t^3 + a_{22} t^2 + a_{23} t + a_{24}) \\ (a_{31} t^3 + a_{32} t^2 + a_{33} t + a_{34}) \\ (a_{41} t^3 + a_{42} t^2 + a_{43} t + a_{44}) \end{bmatrix}$$

Although the process of generating the four cubics can easily be thought of as a matrix multiplication of data stored in memory, the four cubics are actually generated by difference equation methods. Each new point on the curve requires only three additions per cubic equation, or twelve additions in all. The equipment does not make any references to memory during generation of a curve

Because the clipping divider divides the X and Y values it is given by two separate Z values (which are usually made to be the same), the resulting positions of the points on the display will follow equations of the form:

$$X_s = \frac{(a_{11} t^3 + a_{12} t^2 + a_{13} t + a_{14})}{(a_{21} t^3 + a_{22} t^2 + a_{23} t + a_{24})}$$

$$Y_s = \frac{a_{31} t^3 + a_{32} t^2 + a_{33} t + a_{34}}{a_{41} t^3 + a_{42} t^2 + a_{43} t + a_{44}}$$

These expressions differ from those used by Robert's curve drawing display^{5,6} in that cubics rather than quadratics are used and there are separate denominators for X and Y. If connected by short straight line segments, the points generated in this way can adequately represent a curve. The family of curves that can be generated includes all of the conic sections. It also includes a wide variety of curves with inflection points, such as are shown in Figure 14. The matrix multiplier and clipping divider described in this paper can be used to generate such curves in a few hundred microseconds.

Although it is easy to see how the curve drawing system operates, it is not so easy to find the matrix which corresponds to a given desired curve. The mathematics for finding this matrix is more complicated than would be appropriate to discuss here. Suffice it to say that the matrix required to draw a particular desired curve can

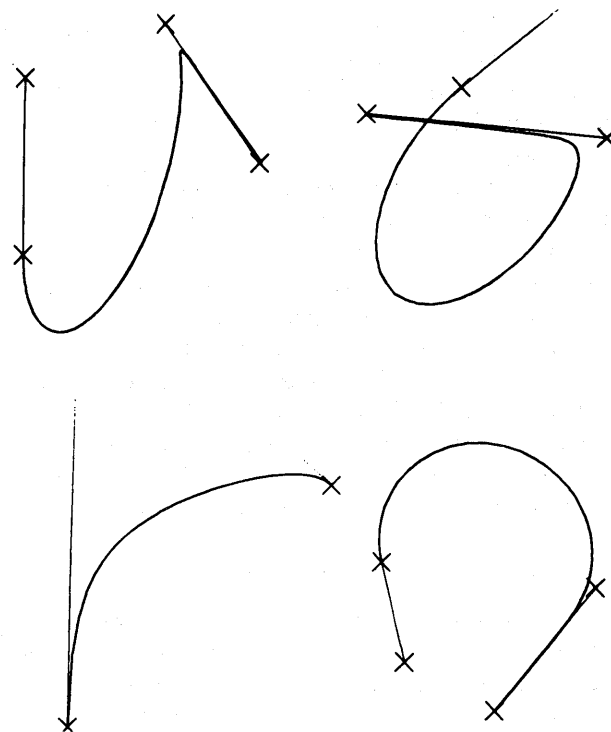


FIGURE 14—Some examples of curves obtainable with the equipment

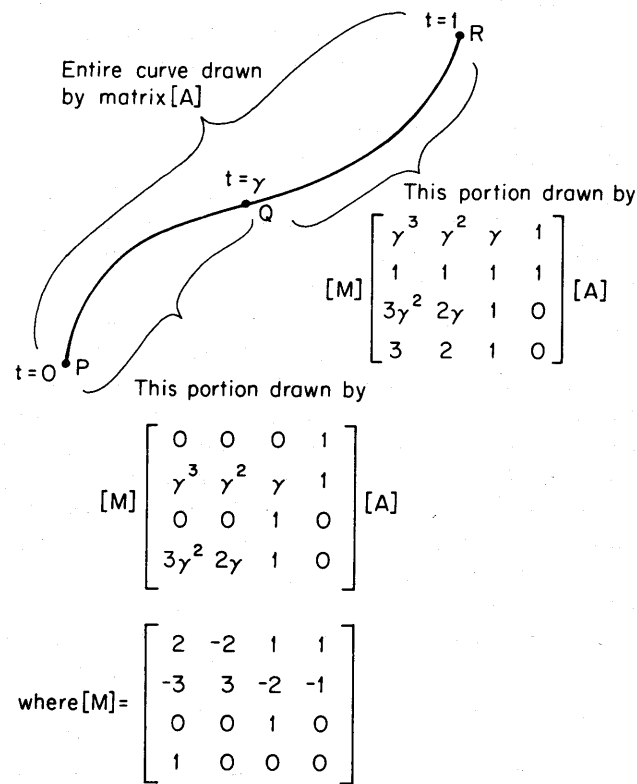


FIGURE 15—Partitioning a curve into sections

be found from many alternative geometric specifications. A curve may be specified by the position and tangent direction at its beginning and end, and by the requirement that it pass through two additional points. Alternatively, the curve may be made tangent to specified lines at its ends and be forced to pass with a given slope through a single additional point.

Methods are available for manipulating the matrices which specify the curves.^{7,8} For example, suppose a particular matrix draws a particular curve from point P through point Q to point R. We might wish to partition the curve at point Q so as to draw it in two sections, each identical in shape to part of the original curve. Multiplying the full-curve matrix by a selected partitioning matrix as shown in Figure 15 will produce the matrix required for the corresponding part.

The Warnock hidden line algorithm

John Warnock at the University of Utah has recently invented a new algorithm for solving the hidden line problem.⁹ The computation time required by the Warnock Algorithm grows more slowly with picture complexity than has ever been the case before. The Warnock Algorithm breaks the picture down into successively smaller "windows" within which the solid objects are examined. If there is nothing of interest within a particular window it need not be further subdivided. If, however, the picture within a certain window happens to be very complex, that window will be subdivided for more detailed examination. Ten levels of binary subdivision will, of course, suffice to produce pictures with a resolution of 1024 lines.

The basic operation of the Warnock Algorithm is to detect whether any edge of a polygon passes through a window. If no edge of the polygon passes through the window, Warnock's program must detect whether the polygon surrounds the window or lies entirely outside the window. The clipping divider described in this paper does this basic operation of detecting whether an edge passes through a window very quickly. In many cases, the line can be trivially rejected as outside the window after only three clock cycles. If the window is fairly large, a midpoint of the line may fall within the window after only three or four more clock cycles, or less than two microseconds. The full clipping process is necessary only if the line just nicks the corner of the window.

A special mode has been provided in the clipping divider for use with the Warnock Algorithm. In this mode, the clipping divider merely announces whether or not a particular line passes through the window; it does not bother to compute the intersections of the line with the edges of the window, nor to transform the resulting information into scope coordinates. Additional equipment is provided in the clipping divider to detect

whether or not a sequence of lines surrounds the window. We believe that with the clipping divider hardware, Warnock Algorithm programs may be possible which will do the hidden line computation in real time.

CONCLUSIONS

Use of a clipping divider makes a fundamental improvement in the logical characteristics of a display. With a clipping divider, a display system can be thought of by its programmer as capable of presenting a magnified image of any portion of a very large picture. The programmer can be entirely free of the bit-packing and resolution difficulties all too common in conventional displays. The picture itself can be represented in the coordinate system most convenient to the computer. For instance, in a computer with a 24 bit word, the coordinates of the endpoints of lines can be represented with the full resolution available in 24 bits. There is no need for the programmer to pack or unpack information into a "display file." He merely specifies the left edge, the right edge, the bottom edge, and the top edge of his window in the same coordinate system used for representing the picture; the display will present on the screen whatever part of the picture is contained within that window. The programmer needs to know about the peculiar coordinate system of the scope only to set the viewport. Because the clipping divider scales its output to be within the range specified as the viewport, display equipment with any origin convention or data width can easily be accommodated. We believe that the freedom from size and resolution limitations, bit-packing, coordinate conversion, and separate display files provided by the clipping divider is well worth the investment required.

ACKNOWLEDGMENTS

We would like to thank the many people who have been involved in the clipper project. Charles Seits designed the high-speed matrix multiplier and provided essential criticism of the clipper design. Ted Lee and Dan Cohen, both graduate students, have been an invaluable part of the project throughout. Lee's work on curves and surfaces, soon to be a PhD thesis, inspired in large part by Coons of MIT, is but hinted at in this paper. Cohen's imagination and thoughts about perspective presentation, "clipping," hidden-line algorithms, and other subjects have had a strong effect on this project. He stimulated invention of the midpoint algorithm and first suggested that our equipment could be used to draw curves.

REFERENCES

- 1 W DWYER.
Windows, shields and shading
Proceedings of the 6th Meeting of UAID Washington DC

-
- p 258 October 16-19 1967
- 2 I E SUTHERLAND
A head-mounted three-dimensional display
Proceedings of the Fall Joint Computer Conference 1968
this issue
 - 3 W R SUTHERLAND
The on-line graphical specification of computer procedures
Massachusetts Institute of Technology Lincoln Laboratory
Technical Report No 405 May 1966
 - 4 T E JOHNSON
SKETCHPAD III: Three-dimensional graphical communication with a digital computer
MIT Electronic Systems Laboratory Technical Memorandum
ESL-TM-173 May 1963
 - 5 L G ROBERTS
Conic display generator using multiplying digital-analog converters
IEEE Transactions on Electronic Computers
 - 6 H BLATT
Volume EC-16 Number 3 June 1967
Conic display generator using multiplying digital/analog decoders
Presented at the Fall Joint Computer Conference
Anaheim California Fall 1967
 - 7 S A COONS
Surfaces for computer-aided design of space forms
Project MAC Report MAC-TR-41 MIT June 1967
 - 8 T M P LEE
Three-dimensional curves and surfaces for computer display
Thesis in preparation at Harvard University
 - 9 J E WARNOCK
A hidden line algorithm for halftone picture representation
University of Utah Technical Report 4-5 May 1968
 - 10 ADAMS ASSOCIATES
Computer display fundamentals
The Computer Display Review Revised March 31 1968

A low cost computer graphic terminal*

by MALCOLM MACAULAY

Information Electronics Limited
Canberra, A.C.T Australia

INTRODUCTION - UNSW GRAPHICAL DATA SYSTEM

Faculty and students of the School of Electrical Engineering, Department of Electronic Computation of the University of New South Wales have been conducting research and development in the field of computer graphics for some time. At this writing several investigators are concurrently working on both hardware and software aspects of a continuing program. The computer graphic television terminal described in this paper is a component developed for the experimental system that holds promise for use in applications requiring low cost and flexibility in use.

Figure 1 is a block diagram of the UNSW Graphical Data System employing the University's IBM 360/50 operating in a multi-programmed, time-sliced manner. A one million byte core storage unit is shared by the central computer and an interface computer, INTERGRAPHIC.¹ The latter machine is designed to provide a

more efficient and economical means to communicate data in on-line multiple graphic console systems than is available commercially. Using fast internal logic and storage in a general-purpose organisation, INTERGRAPHIC can support twenty conventional graphic consoles or one hundred television terminals.

The IBM 360/50 System, well-known production commercial equipment, does not require further description. The configuration at the University Computation Centre is not unusual for 360/50 installations. Paper tape reader, line printer and card reader/punch equipments are connected to the multiplexer IO channel. The other two IO channels serve four discs and four tape drives each. One million bytes of core storage having an eight microsecond cycle time and dual access feature is installed. Communication between the 360/50 system and the graphical data terminal control equipment is accomplished by exploiting this dual storage referencing attribute. Messages for the terminal system are stored in particular locations in the LCS by the 360/50 CPU. In similar manner, INTERGRAPHIC stores messages for the central computer in other particular LCS locations. Both computers are able to operate, to a large extent, as if they were independent of one another. The normal batch processing of computer jobs at the UNSW Computation Centre proceeds apace, delayed only when terminal traffic is very heavy.

The INTERGRAPHIC Computer is both versatile and fast. 3 - 5 nanosecond integrated circuit logic is sequenced from microprograms held in a 100 nanosecond read only memory. Operating in the diagrammed system, INTERGRAPHIC assumes the graphical communication tasks that are burdensome to a conventional general-purpose computer. It acts as a satellite time-shared machine that generates display vectors and

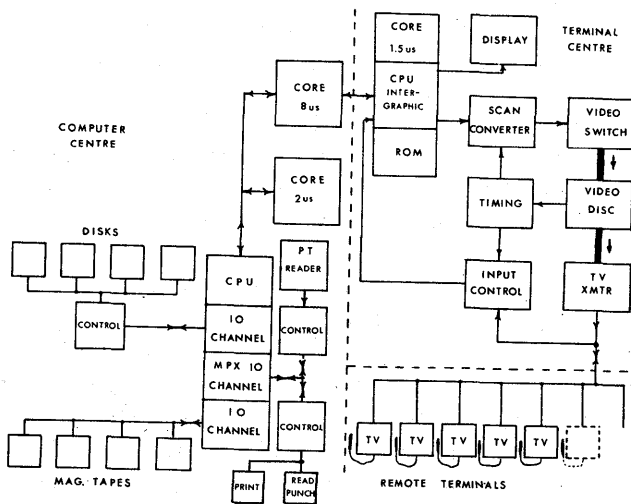


Figure 1—Block diagram—UNSW graphical data system

*The experimental equipment described in this paper and other related equipment are the subject of a patent application filed by Unisearch Limited, the research and development company of the University of New South Wales.

symbols, performs the picture scaling, shifting and rotating tasks and processes lightpen and raster stylus inputs for all terminals, both conventional and TV type.

In addition to the INTERGRAPHIC Computer, the central graphic terminal control system comprises the picture conversion, regeneration and distribution elements. A small electrostatically deflected CRT is used to present the images generated by INTERGRAPHIC. One complete image as intended for display at one terminal is generated within one TV field time, the generation being synchronised by pulses derived from the timing track of the video disc recorder. The same timing pulses control the TV sweep generator thus keeping the INTERGRAPHIC Computer and TV systems in step. The video disc has a potential capacity of 64 channels. Console identification information from INTERGRAPHIC controls the video distributor to record the newly made image on the appropriate track. During the field time immediately following the TV camera is actuated to generate the video signals required. With all terminals operating at maximum speed in an experimental 13 channel system, each terminal receives fresh information every 0.52 seconds. Such energetic simultaneous usage of all 13 terminals is a rare, virtually impossible event.

Picture distribution may be performed by conventional TV RF broadcast techniques in applications where individual picture presentations are not necessary. In such a case, the terminal operators might interact individually by transmitting their raster stylus data to the graphical data centre via telephone circuits. For applications having individual picture requirements, signals from the video disc are used to modulate oscillators operating on the frequencies of the standard TV channels. The high frequency signals thus generated are distributed by coaxial cable in the same fashion employed by CATV systems and multi-channel closed circuit TV installations.

Advantages of television display

The standard commercial television receiver offers a number of advantages as a computer graphic terminal. Brown² has suggested that a unified system born from the marriage of computer and television technology may well be the principal information medium of the near future. Systems using computer television terminals have been described with a listing of significant benefits: (3,4,5,6)

1. Economy
2. Communication simplicity
3. Maintainability
4. Universality
5. Human factors advantages

Television receivers are an order of magnitude cheaper than conventional computer CRT terminals. This low cost, which is an outstanding attribute, is achieved largely from the very high volume of production of equipment for consumer use. The nature of the techniques used also work to TV's cost advantage. Since television scanning proceeds in a regular fashion a tuned transformer in the horizontal deflexion circuit delivers real power for rectification and use for the CRT accelerating voltage. Random scan and electrostatically deflected display systems require more expensive high voltage power supplying means.

The transmission of TV signals by both wire and wireless is readily accomplished. Offering both low cost and superior performance, video wire distribution systems are seriously competing with wireless broadcasting in Great Britain.⁷ Television line distribution components for both CATV and HF networks are designed and manufactured in reasonable quantities. Of importance in relation to computer systems is the fact that TV display scanning is controlled by synchronising pulses transmitted with the picture information thus providing a means to maintain system time readily. The raster stylus circuits described later in this paper exploit these regular clocking pulses to derive coordinate data from the display surface. Although not used in the experimental system, TV colour burst signals could be employed to provide a high synchronisation resolution.

The maintenance advantages of using a common household item, a TV receiver, are obvious. Component parts are cheap and commonly stocked in most neighborhoods. Trained servicemen are available on call. In a system such as the UNSW Graphical Data Network, maintenance monitors may be installed to insure the quality of picture signals being transmitted.

Unlike conventional computer displays, TV receivers may be used with inputs from TV cameras and flying spot scanners. A computer terminal employing conventional television is equally useful for viewing educational video tapes and for family entertainment. The provision for colour display and the potential for monochrome picture tonal gradation make TV technology of interest for further exploitation in computer terminal use.

TV offers advantages from the human factors point of view. In one reported experiment⁸ subjects were found to prefer TV images to high resolution images, especially for line drawings. The same investigators pointed out the need to invert display image polarity for operator comfort as ambient illumination varied. Such image inversion, although impossible or impractical in conventional computer display systems, is readily accomplished with TV. Further advantages of TV are found in the relatively high picture regeneration rate

assuring freedom from flicker and the availability of ready operator control of both brightness and contrast to his personal taste.

Disadvantages of television display

One of the disadvantages is that TV, being primarily a low-cost entertainment medium designed to compromise picture quality and bandwidth considerations does not possess high resolution adequate for all computer terminal needs. This deficiency is somewhat offset in a computer environment by the ability to scale the image readily and to mark coordinates with high resolution digital numeric data. Due to the nature of the consumer goods market, rather than inherent weakness in the system, common TV receivers provide less picture information and more geometric distortion than may be obtained.

In any display technique using CRTs, the transient nature of the light output from the moving spot requires that the picture be restored at a regular rate. The characteristics of the human eye are such that if the image is presented at a frequency above the so-called "critical fusion frequency", the picture will appear to be present continuously. TV practice is to provide a field frequency related to the power supply mains frequency rather than the somewhat lower critical fusion frequency. In Australian television practice the field frequency is 50 Hz. Although a modern electronic computer has been used to generate TV video at higher rates⁹ there is little time left for other useful work when so employed. Although TV requires a higher image refresh rate than conventional computer display, this disadvantage is more than offset by the ready availability of video disc storage devices to perform the image maintenance function.

Scanning the picture from top to bottom, left to right in two interlaced fields per frame, has the effect of dissecting the image in an awkward manner from the computer point of view. Symbols, vectors and other normally associated groups of picture elements are disjointed and scattered in time. Although the dismemberment is done in a consistent manner expressible in a reversible algorithm, the timing is such that storage capacity for a full picture field is needed for general use. In the UNSW Graphical Data Network this storage is found in the phosphor persistence of the small display CRT and electrical charges on the target of the camera tube.

Experimental graphic terminal

An experimental graphic terminal was constructed by modifying a standard commercial television receiver. Means were provided, to be described later, to extract

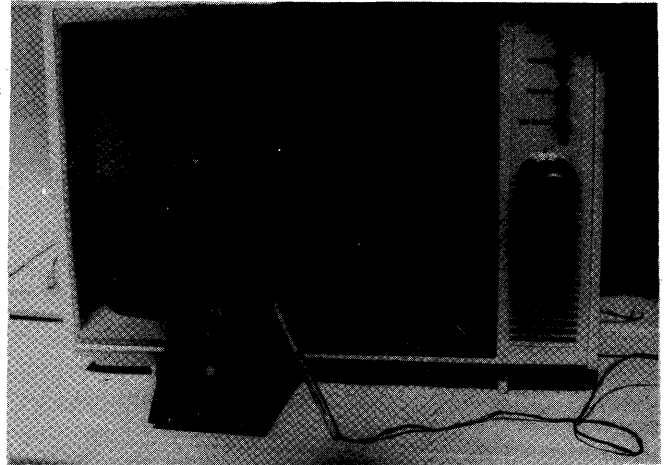


Figure 2—Experimental graphic terminal

horizontal and vertical synchronising pulses and to mix video signals from the raster stylus with TV video. Figure 2 shows the appearance of the completed unit.

The terminal block diagram, Figure 3, gives an overview of the operating fundamentals. The physical location of the raster stylus on the TV screen is determined by counting horizontal synchronising pulses from a high frequency oscillator to yield the horizontal dimension, then counting pulses from a high frequency oscillator to yield the vertical dimension. In the interest of economy, only one counter is used, its input being alternated each field interval from either the horizontal synchronising pulse line or the output of the high frequency oscillator. The count accumulated during a given field interval is transferred to a shift register at the end of the field. The vertical synchronising pulses

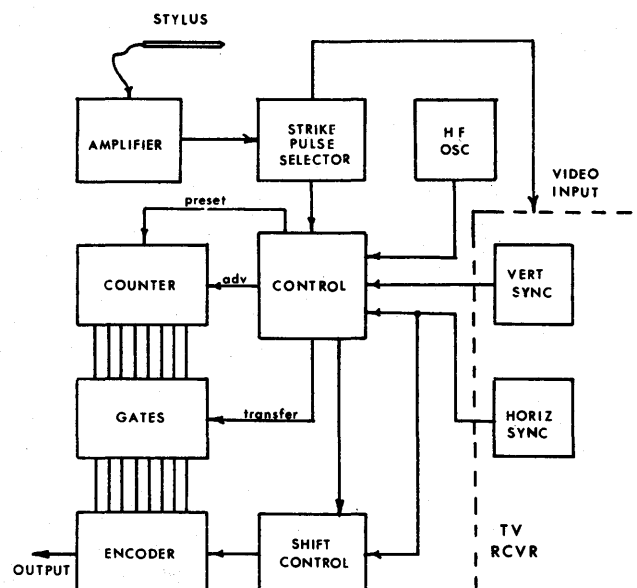


Figure 3—Terminal—Block diagram

are the signal for transfer. The count is then converted into a serial pulse train of low frequency, about 500 Hz. This serial pulse train may be transferred by ordinary telephone lines. Twenty-five raster stylus coordinate pairs are generated each second.

Standard mass-produced commercial integrated circuit modules were used to perform almost all of the circuit functions. The amplifiers for video signals and raster stylus and vertical pulses were constructed from discrete components. All the circuits were mounted on an experimental circuit board, 10 X 12 cm. Wiring was done by hand using tinned copper wire and insulating sleeving for each connection. Although labourious, this technique had the virtue of permitting easy alteration while allowing for a high component packing density. Figure 4 is a photograph of the completed item.

The raster stylus was made from a discarded ball point pen. The ink tube and point were removed and the tip cone enlarged by drilling to accommodate the photodiode. The diode load resistor was connected at the diode terminals and a small segment of plastic tubing was added to give the assembly rigidity. Low impedance circuitry permitted the elimination of shielding from the raster stylus connection wires. Figure 5 is a photograph of the raster stylus prototype.

The total power consumption of the circuits is 2.3 watts, being 650 ma at 3.5 volts. The total component costs, less power supply, for the complete circuit is \$A63-02 (\$70.58 US).

Terminal logic

Figure 6 gives the logic symbols used in the diagrams of this paper. Three IC module types were used: Type 900, a low impedance inverting amplifier; Type 914, a

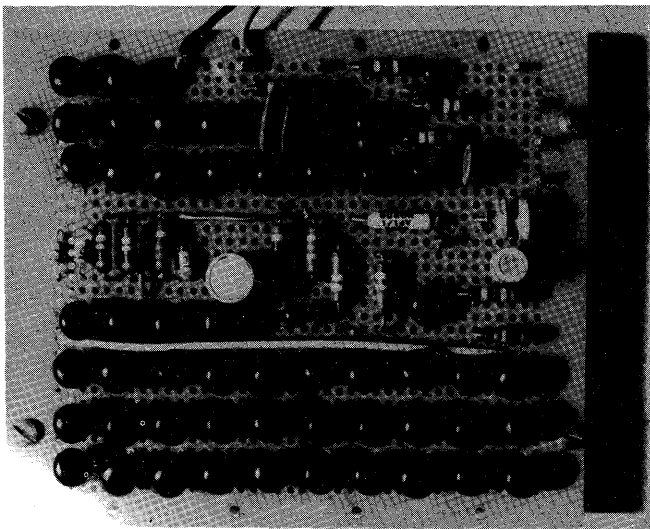


Figure 4—Terminal circuit board prototype

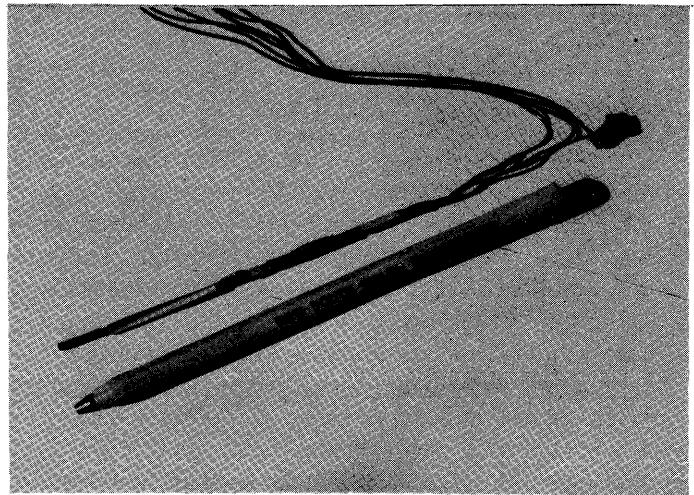


Figure 5—Raster stylus

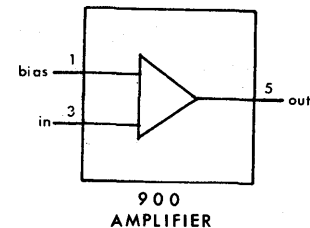
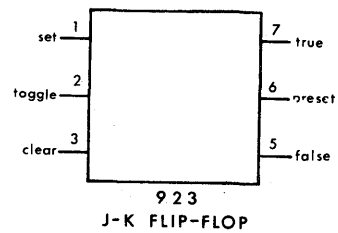
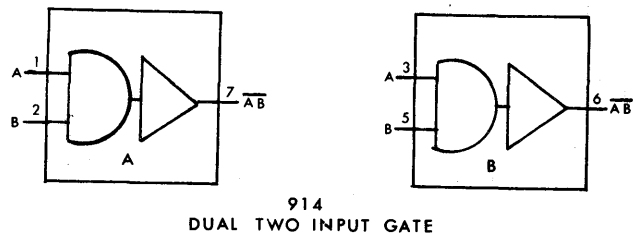


Figure 6—Logic symbols—IC modules

dual two input gate element and Type 923, a JK flip-flop.

Horizontal and vertical synchronising pulses were extracted from the TV receiver by the simple circuits shown in Figure 7. A wire connected at the output of the vertical integrator (the junction of R67, R68 and C66) conveyed the vertical pulses to an emitter-follower amplifier to provide a power gain adequate to drive an IC logic module. Horizontal synchronising pulses were extracted by adding a 330 ohm resistor in series with the 8.2k ohm collector load resistor of the sync separator transistor of the TV receiver. This means provides a low impedance circuit point from which pulses may be taken.

The HF Oscillator used to establish the horizontal coordinate data is diagrammed in Figure 8. The dual gate integrated circuit is connected to operate as an astable multivibrator. The inverter performs both waveform shaping and load isolation functions. The oscillator operates at a frequency of approximately 4 MHz., providing 240 counts during the active TV scanning interval.

Figure 9 is a diagram of the circuits associated with the control of preset and advance pulses to the counter. Flip-flop G4 acting with gate C1 permits HF Oscillator pulses to provide the advance signals during alternate television field intervals. During the y-coordinate determining interval horizontal sync pulses toggle flip-flop A1 providing counter advance pulses at $\frac{1}{2}$ horizontal sweep frequency. During the x-coordinate determining interval HF Oscillator pulses reach the advance line. At this time horizontal sync pulses are present on the preset line thus clearing the counter at the end of each display sweep. Master preset pulses derived from the ver-

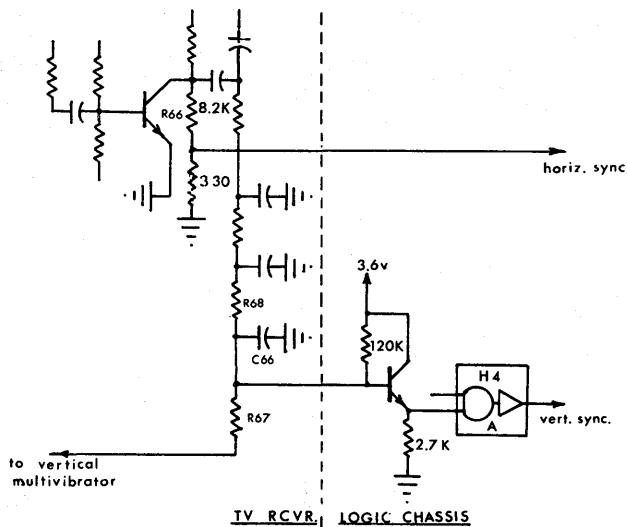


Figure 7—Synchronising signal extraction circuits

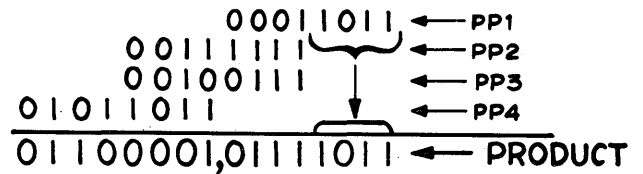
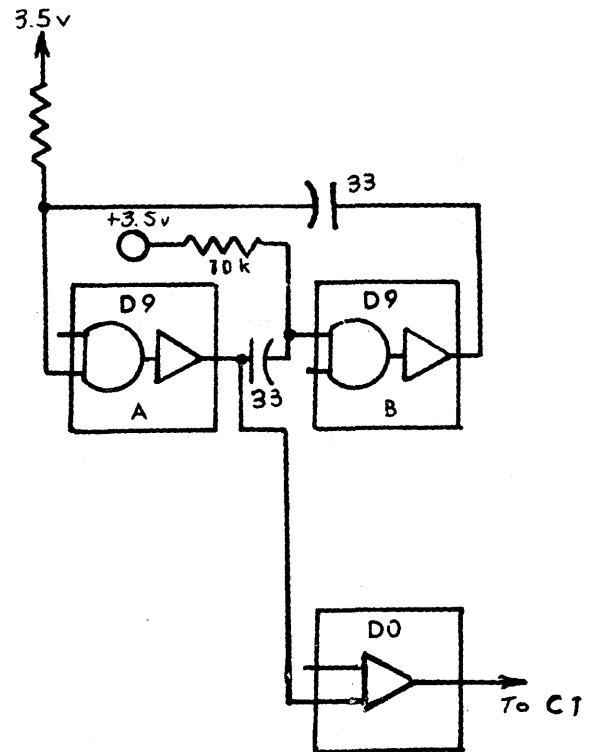


Figure 8—HF oscillator

tical sync pulse appear on the preset line at the onset of each coordinate determining interval.

Figure 10 is a block diagram of the logic used in the counter, gates and encoder. The counter is a conventional 8 stage modulo 256, count-up, binary ripple counter. The transfer gates are operated by a pulse on the transfer line prior to the master preset pulse, thus assuring the capture of the count accumulated in the registers before preset action. The encoder register is a conventional shift-right register, the timing established by pulses on the shift line that are derived from the horizontal sync signals. The least significant bit, B1, has its set input connected to the positive supply voltage and its

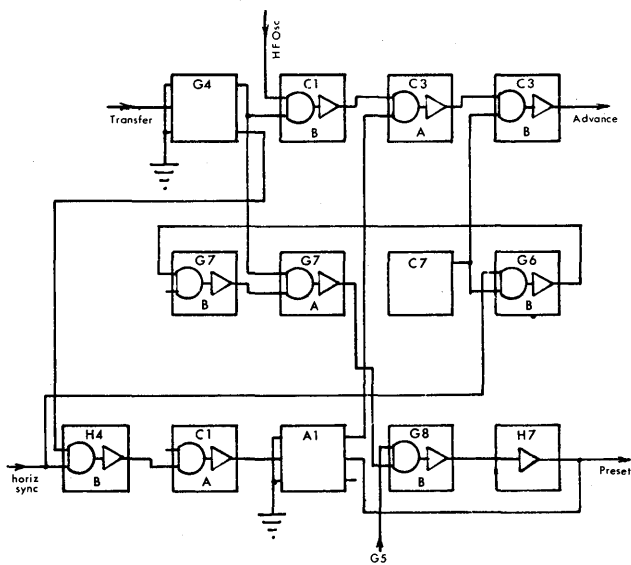


Figure 9—Advance and preset control circuits

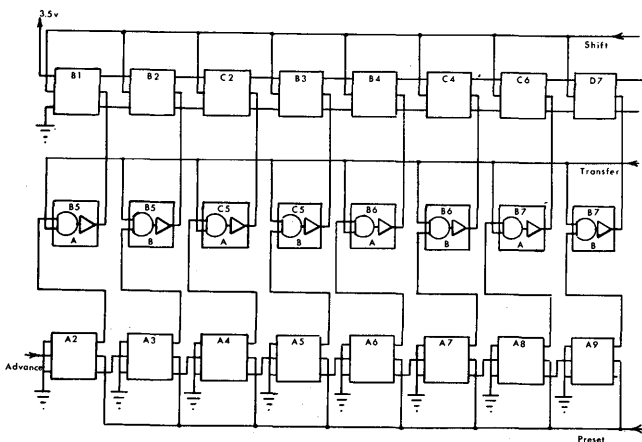


Figure 10—Counter, gates and encoder

clear input connected to ground so that each stage is set to true at the end of the encoding interval. The transfer gates provide a signal to the individual shift register stages only during the transfer interval if the corresponding counter bit is true. In other words, the complement of the count is transferred to the shift register.

The shift and transfer control circuit diagrammed in Figure 11 makes use of both horizontal and vertical synchronising pulse inputs. Horizontal pulses at the output of amplifier G8 are counted by a binary ripple counter comprising five stages, I0 H0, G0, H9 and I9. Counted modulo 32, the resulting signal is used to actuate the shift line and controls the timing of the coordinate data encoding. One-shot multivibrators G3 and H5 establish the timing of transfer and master preset pulses.

An audio frequency output pulse wavetrain that can

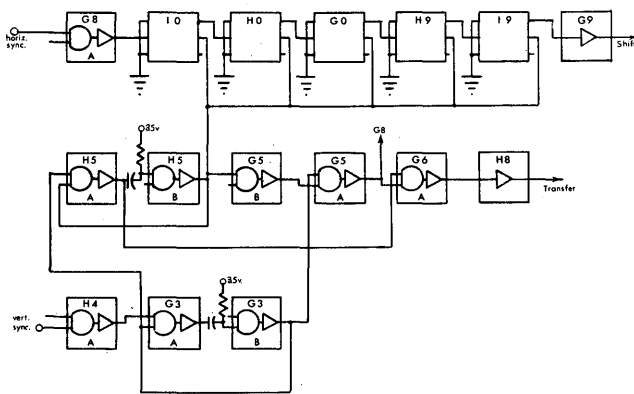


Figure 11—Shift and transfer control circuits

be transmitted over voice grade circuits is simply provided by combining the output of the encoder from flip-flop D7 with the shift frequency signals from flip-flop I9. The transfer pulse from G6 is mixed with these signals to provide a reference signal. A typical output wavetrain is shown in Figure 12 and the wiring of the dual gate in Figure 13.

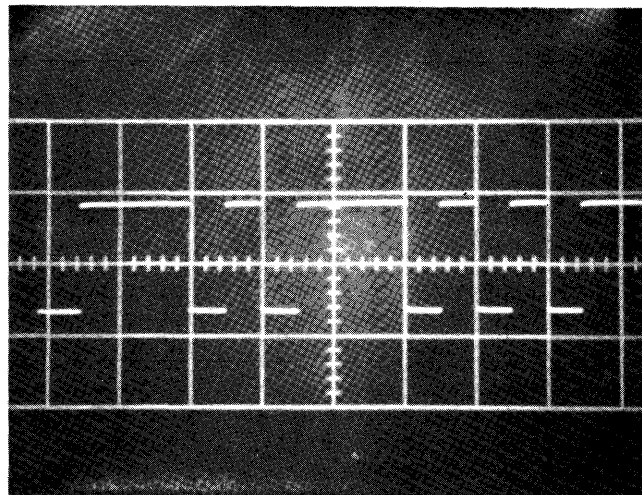


Figure 12—Audio output wavetrain

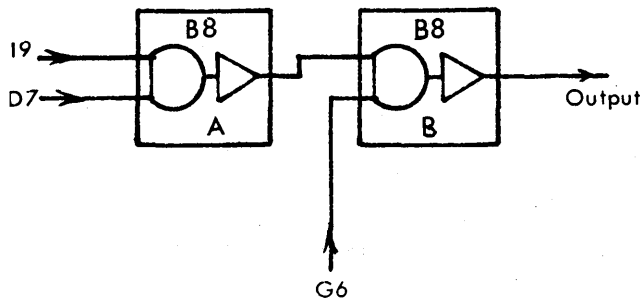


Figure 13—Output wavetrain gate

The raster stylus

In operation with a conventional computer driven display system, a lightpen must perform the following functions :

1. Collect light from a small area of the CRT screen and transfer a "strike" signal to the computer.
2. Notify the computer, by operator action, that the lightpen is positioned at the coordinates desired.
3. Notify the computer by operator action that the lightpen position is to be tracked.
4. Provide indication to the operator that the lightpen is properly aimed and focused.

Commercially available lightpens employ fibre-optic cables and photomultipliers or phototransistors. Light from a source or sources at the end of separate terminally coaxial portion of the fibre-optic cable is focused on the CRT face by the lens through which the detected light must pass, thus signaling proper focus and aiming to the operator. The construction of a typical modern production lightpen is described by Locascio, Karanza and Dalton.¹⁰

For the designer, the lightpen light collection design problem is complicated by many uncertain factors. Lightpen sensitivity is the function of the instantaneous brightness of the display, the luminous intensity of the light at the photosensor and the photosensor response.¹¹

The properties of entertainment grade CRTs reflect a lack of precise control of phosphor material distribution, screen thickness and aluminizing coating. The resulting display, although quite acceptable for entertainment use, does not always exhibit the predictable behaviour of CRTs used in most computer display terminals. In the light of the obvious analytical difficulties, an experimental approach was taken to the raster stylus design. It was found that a common commercial photodiode performed satisfactorily with the circuit shown in Figure 14. The light collection was performed by the integral lens of the photodiode.

Since the television raster is scanned in a consistent manner from left to right and top to bottom, the photodiode detects the light emitted by the passage of the electron beam during several sweeps, i.e., from lines both above and below the desired point. The number of detections and consequent amplifier output pulses is determined by the size of the cone of light accepted by the photodiode lens and the intensity of the light emitted by the CRT phosphor. The amplifier output pulse amplitude is thus influenced by the uncertain factors aforesaid. In addition, vertical movement, either in the television raster from circuit instabilities or in stylus position by operator action will alter the response signal amplitudes markedly. Modulation of the beam current density by high tension fluctuations related to

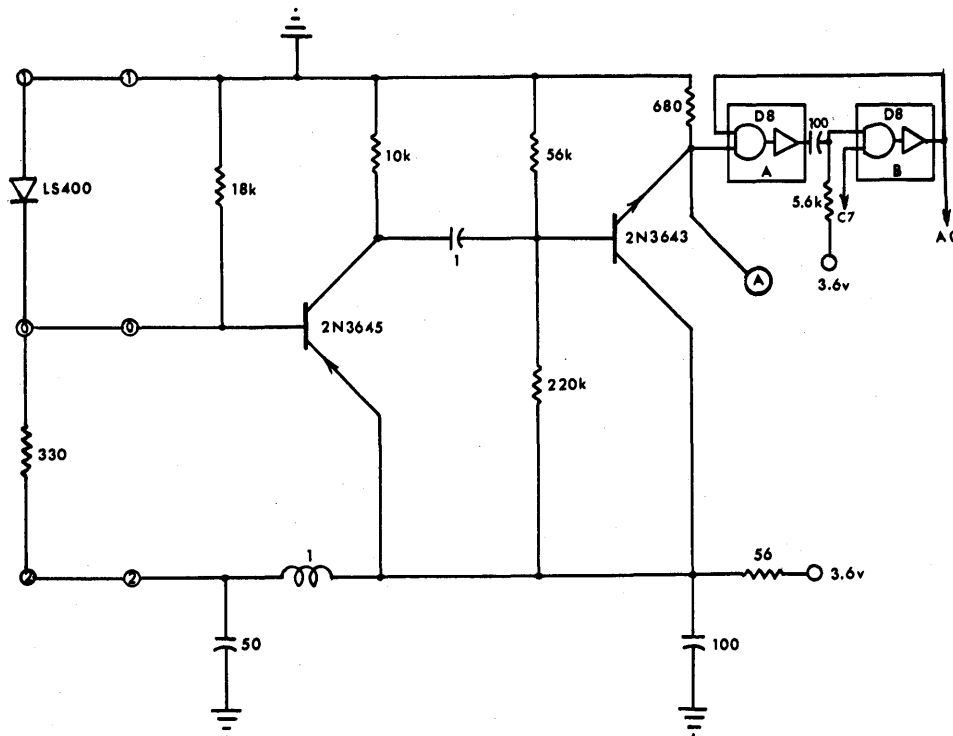


Figure 14—Raster stylus amplifier and pulse shaper

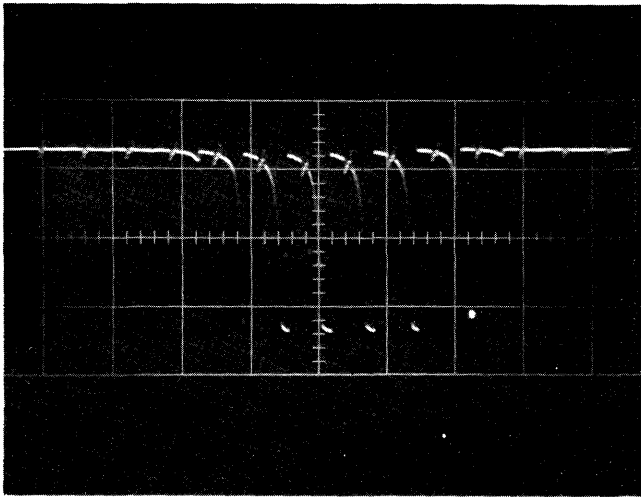


Figure 15—Raster stylus response signal $v = 0.5 \text{ v/cm}$ $h = 50 \text{ microsec/cm}$

mains voltage or circuit instabilities and adjustments to the Set Black and Picture controls of the TV receiver are also significantly influential.

Figure 15 shows typical raster stylus response signals at the output of the amplifier (Point A of Figure 14). At the level of light then prevailing, six pulses of sufficient amplitude to trigger the monostable multivibrator were evident. At lower levels fewer pulses would be of amplitude sufficient to reach the triggering threshold of the multivibrator. At higher light levels, more pulses would cause triggering. As aforementioned, the pulse amplitude is unpredictably time variant and the multivibrator triggering action consequently unreliable. Experiments showed, however, that one and more reliable pulses could be assured by proper setting of the TV receiver Set Black and Picture controls. These settings were within the range that yielded the most acceptable displayed pictures. Further, the unreliable pulses were

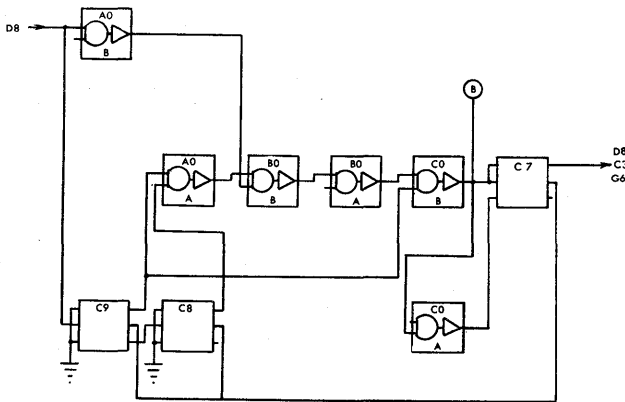


Figure 16—Raster stylus logic circuit

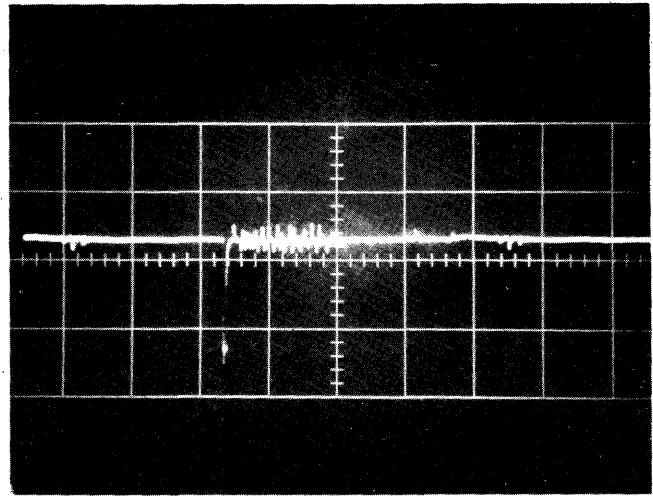


Figure 17—Selected raster stylus pulse $v = 5 \text{ microsecs/cm}$ $h = 1.0 \text{ v/cm}$

derived from the passage of the CRT beam on the lines either above or below the desired point. Thus it was possible to use digital pulse selection rather than optical field narrowing. The simple algorithm is to count pulses, ignoring all but the third pulse from the multivibrator output. The raster stylus logic circuit of Figure 16 accomplishes the pulse selection. Figure 17 shows the pulse selected, the waveform measured at point B.

The operator is aided in adjusting the controls of the TV receiver by observing the behaviour of a point of action display spot created by mixing the selected pulse with the TV receiver video signals. If the light level is too low, the spot will either disappear completely or "dance" irregularly on the screen. Too much light will not hamper the raster stylus performance but will tend to cause the display to defocus and to dazzle the operator with its brightness.

In operation, the raster stylus is held against the face of the CRT. The aforementioned point of action display spot is evident as a black line, about 1 mm. long displaced about 3 mm. to the right of the stylus tip. This displacement is an advantage since the operator's fingers are thereby made less likely to interfere with his view of the screen. Most conventional commercial systems deliberately offset the tracking cross by computer program to accomplish the same effect. As a consequence of the logic circuits, the operator cannot write on either of the top two lines of the TV raster. Similarly, the leftmost edge of the screen has a vertical band, about 3 mm. wide, in which raster stylus action may not be obtained. This forbidden zone results from the time delays in the photodiode and amplifier. The limitation is not significant, less than 1% of the total screen area is excluded.

CONCLUSIONS

The experimental work described in this paper has demonstrated the economy and utility of computer graphic terminals using standard television receivers. Although user opinion from operational experience must be the final arbiter, it is plausible to infer that TV technical performance will prove adequate for most uses. Television receivers, because of the ability to display both computer generated and conventional TV pictures, offers unequaled versatility. No alternative technique appears to offer equal effectiveness at comparable cost.

Television maps picture element position information into time variant electrical signals. The raster stylus exploits this position-time isomorphism to control the counting of a digital register from which picture element position information is reported to a computer. Raster stylus use does not require either computer tracking programs nor lightpen inking or tracking cross service. Tracking cannot be lost. The raster stylus writes directly on the display surface. Remote operation is made possible since computer interrupt signals are not employed with their critical timing which requires short, wide-band communication lines.

ACKNOWLEDGMENTS

The work described in this paper is based on experiments performed during the author's tenure as a Visiting Fellow at the University of New South Wales. Inspiration for the direction of investigation came from many sources, both internal and external. Especial thanks are due M.W. Allen whose interest and encouragement advanced the progress of the work significantly. The Staff of the UNSW Electrical Engineering School, G.A. Rose and G. Bowen in particular, made many helpful suggestions. My son, Michael, contributed greatly with his skill in engineering drawing and is responsible for the logic and circuit diagram illustrations.

REFERENCES

- 1 G A ROSE
Intergraphic A microprogrammed graphical interface computer
IEEE Transactions on Electric Computers
Vol EC-16 No 6 December 1967 pp 773-784
- 2 G H BROWN
Television's role in tomorrow's world
IEEE Spectrum October 1967 pp 56-58
- 3 H C HENDRICKSON
The display control complex of the manned space mission control center
Information Display May June 1967 pp 52-58
- 4 S GRAHAM
Using a standard television monitor as an alphanumeric display
Information Display May June 1967 pp 59-61
- 5 S B GRAY
A computer time-shared display
Information Display
January-February 1966 pp 50-51
- 6 M MACAULAY
Input output terminals for the computing utility
Master of Engineering Thesis University of New South Wales
Kensington N. S. W. Australia April 1968
- 7 R P GABRIEL
Wired broadcasting in Great Britain
IEEE Spectrum April 1967 pp 97-105
- 8 C K CLAUER A S NEAL R L ERDMANN
Evaluation of some display parameters with human performance measures
Society for Information Display, Seventh National Symposium
on Information Display, Technical Session Proceedings
Boston Massachusetts October 1966 pp 15-23
- 9 M MACAULAY
Low-cost terminals using television techniques
National Radio and Electronics Engineering Convention
Abstracts IREE Australia May 1967 p 222
- 10 J T LOCASCIO G L KARANZA J J DALTON
Obtaining light pen versatility
Information Display November December 1967 pp 36-39
- 11 T STUPAR
Characterization of light pen sensitivity
Information Display May June 1967 pp 67-69

Computer simulation of a nonlinear blood flow model

by HERBERT A. CROSBY*

University of Missouri
Rolla, Missouri

and

MURLIN K. KLUKIS*

Douglas Space Systems Center
Huntington Beach, California

INTRODUCTION

Derivation of vascular system equations

The following equations are proposed in identifying a portion of the vascular system of a human. The derivation of the equations is similar to the approach used for a waterhammer analysis and is based on those of Dr. Victor Streeter,¹ who has written extensively on the problem of waterhammer. The approach that is followed here is similar to that used for elastic waterhammer, which considers the flow of a fluid in an elastic pipe. In the human vascular system there is also the flow of a fluid in an elastic pipe. With the proper parameters and boundary conditions it is reasonable to assume that these derivations will lead to a set of equations which can be used to describe the dynamical properties of the vascular system. In order to apply these equations to the flow of blood it is necessary to make the following assumptions:

1. The blood vessel and elastic tube have a constant modulus of elasticity and a constant wave velocity.
2. The blood vessel is not permeable and is cylindrical with a constant internal diameter at rest.
3. Blood flow is laminar after leaving the upper portion of the aorta.
4. The loss of energy due to friction between the fluid and the walls of the blood vessel is proportional to the square of the velocity.

5. There are no discontinuities. This is to say that there is no branching along the section of blood vessel under consideration.

The first assumption is reasonable because the blood vessel stress-strain curve is linear over a significant range, and many investigators agree that this assumption does not introduce any significant error.² It is possible to include the change of the modulus of elasticity in the equations, but it introduces a considerable amount of complexity and would mean having variable coefficients instead of constant coefficients in the equations.

The second assumption is necessary in order to have convenient boundary conditions. When the blood leaves the left ventricle of the heart and enters the aorta, at the semilunar valve leading into the aorta some turbulence occurs.³ After leaving this valve the flow becomes laminar until it comes to a discontinuity; therefore, assumptions three and five are needed. Assumption four has been shown to be true for elastic tubes by Streeter and others.²

Let us begin by considering the flow of a fluid in an elastic tube. In the waterhammer problem analysis there is a sudden change in pressure at the entrance of a tube due to an opening or closing of a valve. The sudden opening or closing of this valve will cause a pressure at the distal end of the tube that is greater than up stream, while the velocity of the fluid decreases as it flows down stream.¹ In the case of blood flow in the aorta there is also a rapid change in pressure (approximately 130 mm Hg in 0.2 to 0.3 seconds). The vascular system responds with a higher pressure down stream along with a

*The authors were formerly at Southern Illinois University, Carbondale, Illinois.

decrease in the velocity of the blood. The pressure does decrease as the blood moves nearer to the capillary bed. This correspondence indicates the possibility that with proper boundary conditions and careful selection of parameters one can use the elastic waterhammer equations to describe and identify the vascular system parameters.

In the derivation Newton's equation of motion and the equation of continuity will be applied to an element of fluid within the confines of an elastic, cylindrical vessel. The derivation follows:

The dependent variables are functions of distance and time.

- H = pressure head = H(x, t)
- V = velocity (averaged at a cross section) = V(x, t)
- P = pressure = P(x, t)

The independent variables are,

- x = distance along the tube
- t = time

Equation of motion

Sum the forces in the x-direction, considering only one-dimensional flow. (See Figure 1)

$$\begin{aligned}
 PA - [PA + \frac{\partial}{\partial x} (PA)\Delta x] - \tau\pi D\Delta x + \frac{P\partial A}{\partial x} \Delta x \\
 = \rho A\Delta x \frac{dV}{dt} \\
 - A\Delta x \frac{\partial P}{\partial x} - P\Delta x \frac{\partial A}{\partial x} - \tau\pi D\Delta x \\
 + P\Delta x \frac{\partial A}{\partial x} = \rho A\Delta x \frac{dV}{dt} \tag{1}
 \end{aligned}$$

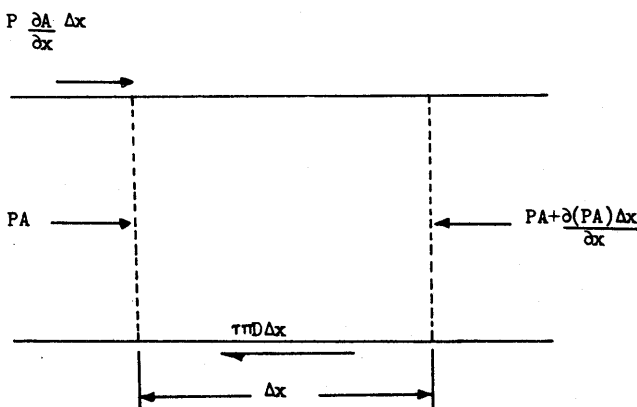


FIGURE 1—Free body diagram for a segment of fluid

Dividing through by the mass of the element of fluid, ($\rho A\Delta x$), and simplifying

$$-\frac{1}{\rho} \frac{\partial P}{\partial x} - \frac{4\tau}{\rho D} = \frac{dV}{dt} \tag{2}$$

For steady turbulent flow, $\tau = \rho fV^2/8$. This assumes that the friction factor in unsteady flow is the same as in steady flow.¹

Substituting for τ and $P = \rho gh$, expanding the acceleration term, and substituting into equation (2),

$$g \frac{\partial H}{\partial x} + V \frac{\partial V}{\partial x} + \frac{\partial V}{\partial t} + \frac{fV^2}{2D} = 0 \tag{3}$$

In the arterial system there is negative flow which must be considered; therefore, V^2 is written as $V|V|$ to provide the proper sign.

The equation of motion becomes,

$$g \frac{\partial H}{\partial x} + V \frac{\partial V}{\partial x} + \frac{\partial V}{\partial t} + \frac{f}{2D} V|V| = 0 \tag{4}$$

Equation of continuity

The equation of continuity states that the mass flow into the boundary minus the mass flow out of the boundary is equal to the rate of change of mass within the boundary. (See Figure 2)

Applying the continuity equation to the control volume,

$$\rho AV - [\rho AV + \frac{\partial}{\partial x} (\rho AV) \Delta x] = \frac{\partial}{\partial t} (\rho A\Delta x) \tag{5}$$

Simplifying, expanding, and dividing through by the

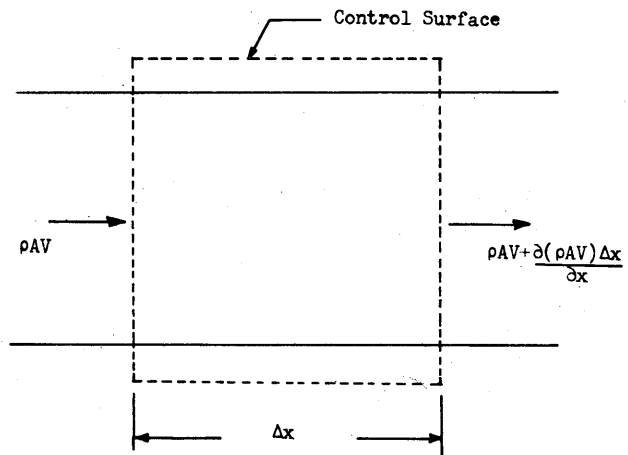


FIGURE 2—Control volume for a segment of fluid

mass of the element, $\rho A \Delta x$,

$$-\frac{\partial V}{\partial x} - \frac{V}{A} \frac{\partial A}{\partial x} - V \frac{\partial \rho}{\partial x} = \frac{1}{A} \frac{\partial A}{\partial t} + \frac{1}{\rho} \frac{\partial \rho}{\partial t} \quad (6)$$

Reorganizing the terms,

$$\frac{1}{A} \left(V \frac{\partial A}{\partial x} + \frac{\partial A}{\partial t} \right) + \frac{1}{\rho} \left(V \frac{\partial \rho}{\partial x} + \frac{\partial \rho}{\partial t} \right) + \frac{\partial V}{\partial x} = 0 \quad (7)$$

Note that the first two terms in the parentheses constitute the total derivative of A, and the second two terms in the parentheses constitute the total derivative of ρ ; therefore, the continuity equation becomes,

$$\frac{1}{A} \frac{dA}{dt} + \frac{1}{\rho} \frac{d\rho}{dt} + \frac{\partial V}{\partial x} = 0 \quad (8)$$

Next it is necessary to derive an equation for the change in area in terms of pressure. This derivation makes use of the relationship between hoop stress in the tube wall and circumferential strain. (See Figure 3). The tension, T, is equal to PD/2, where P is pressure and D is the internal diameter. The thickness of the tube wall is assumed to be small compared to the tube diameter.

$$T = PD/2$$

Differentiating both sides with respect to time,

$$\frac{dT}{dt} = \frac{D}{2} \frac{dP}{dt} + \frac{P}{2} \frac{dD}{dt}$$

Assuming a very small change in diameter with respect

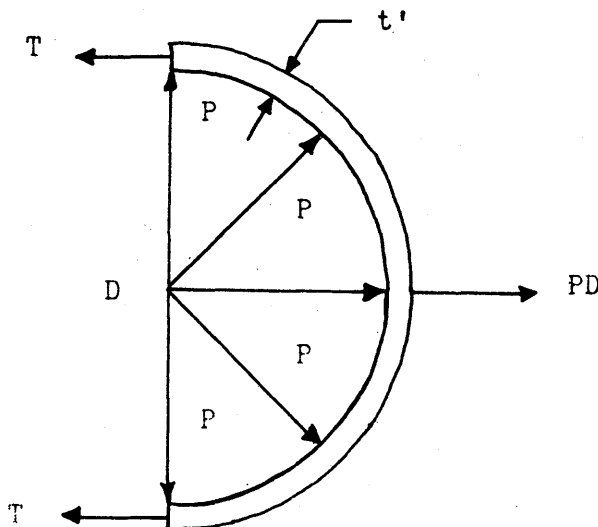


FIGURE 3—Hoop stress for a thin walled cylinder

to the very large change in pressure, the equation becomes,

$$\frac{dT}{dt} = \frac{D}{2} \frac{dP}{dt}$$

Dividing both sides of the equation by the wall thickness, t' , one obtains the rate of change of unit stress. From this equation the relation between change of area and change in pressure can be obtained.

$$\frac{1}{A} \frac{dA}{dt} = \frac{D}{t'E} \frac{dP}{dt}$$

To obtain a relationship between the density of the fluid and the pressure on the fluid use is made of the definition of bulk modulus of elasticity, K.

$$K = -\frac{dP}{dV/V} = \frac{dP}{d\rho/\rho}$$

or,

$$\frac{1}{\rho} \frac{d\rho}{dt} = \frac{1}{K} \frac{dP}{dt}$$

The equation of continuity becomes,

$$\frac{dP}{dt} \left[\frac{1}{K} + \frac{D}{t'E} \right] + \frac{\partial V}{\partial x} = 0 \quad (9)$$

The "waterhammer" expression for the pressure wave velocity will reduce the equation to a more convenient form.

$$a^2 = \frac{K/\rho}{1 + (K/E)(D/t') C_1}$$

Where:

C_1 = constant of the tube. For elastic tubes and blood vessels C_1 is unity.¹

Substituting a^2 into equation (9), expanding, and substituting $P = \rho gH$

$$\frac{V \partial H}{\partial x} + \frac{\partial H}{\partial t} + \frac{a^2}{g} \frac{\partial V}{\partial x} = 0 \quad (10)$$

The two derived equations, (4) and (10), are nonlinear partial differential equations. No analytical solution is known for these equations; therefore, it is necessary to use computer methods to obtain a solution.

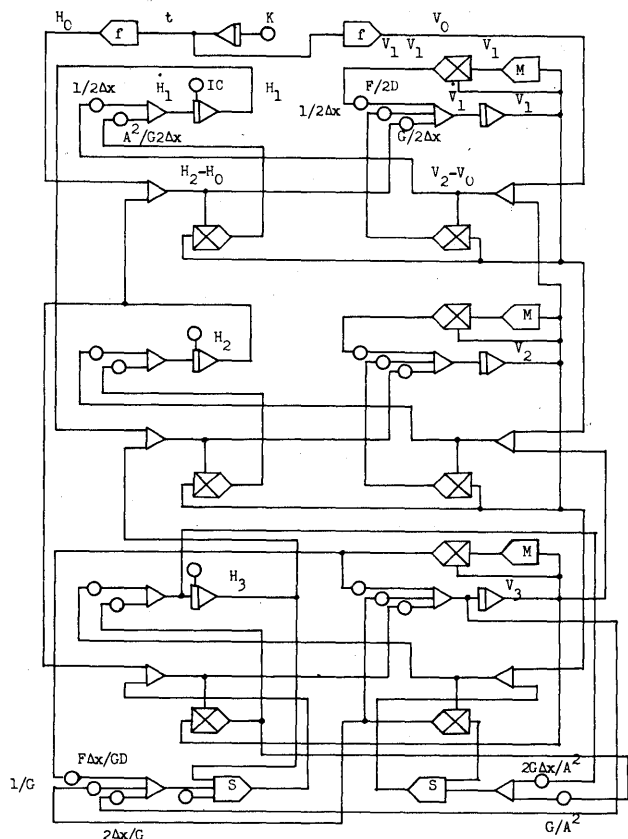


FIGURE 4—Pactolus flow diagram for simulation of the aorta

Simulation technique

Pactolus simulation

The equations derived in the text have been simulated using a continuous systems simulation program, Pactolus, for the IBM 7040 digital computer. The purpose of the simulation was to substantiate the equations derived and to find a method of solution previously unattempted (See Figure 4). Pactolus was written by two members of the IBM Corporation.⁴ Each computing block in the program is a subprogram. The coding is similar to the flow charts used in programming an analog computer. The methodology for constructing the simulation diagram is identical for Pactolus and the analog computer. Pactolus for the IBM 7040 computer has a wide range of available computing blocks for all of the major operations available on an analog computer. The IBM 7040 Pactolus provides the system analyst with ninety-nine computing blocks to work with. This is equivalent to a medium sized analog computer with hybrid features.

Parameters to be identified are those which determine the functioning or malfunctioning of the system. Those of interest in identifying the vascular system are:

1. The flow rate of blood at any time and at any position on the blood vessel.
2. The pressure at any position on the blood vessel as a function of time.
3. To be able to predict the shape of the pressure wave at any position on the blood vessel.

The pressure and velocity functions of the left ventricle were generated using function generators. These generators provide the forcing functions at the entrance to the aorta or vessel under investigation (See Figures 5 and 6). The distal boundary conditions are found by selecting a distal pressure and identifying the corresponding velocity. The boundary conditions at the distal end of the vessel under consideration were also calculated by the use of special blocks in the Pactolus program.

To test the validity of the proposed vascular system equations, data for the human aorta were used. The simulation described here is for a one hundred centimeter length of aorta with the pressures and velocities at 25 centimeter increments from the entrance to the aorta predicted.

To simulate the vascular system equations on the

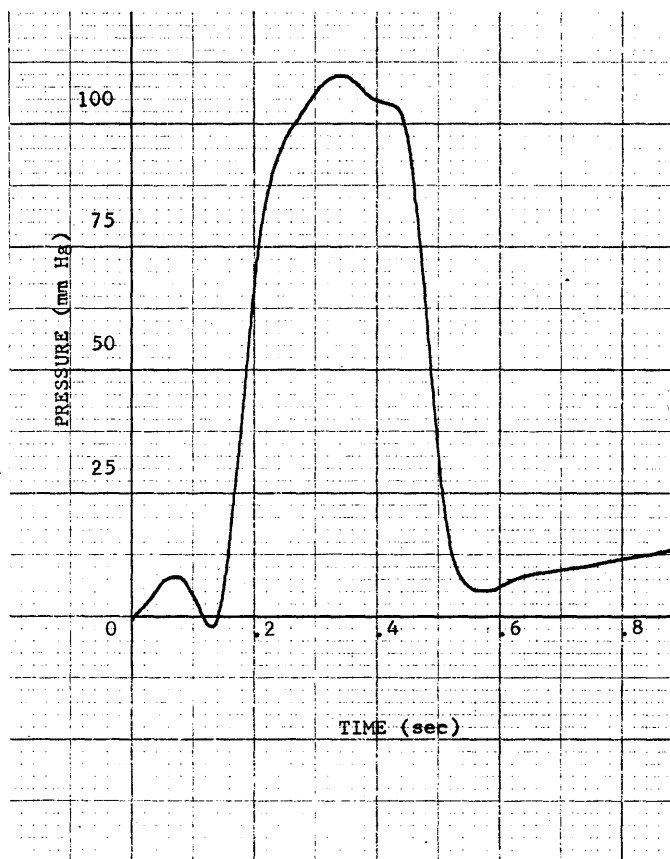


FIGURE 5—Generated left ventricle pressure

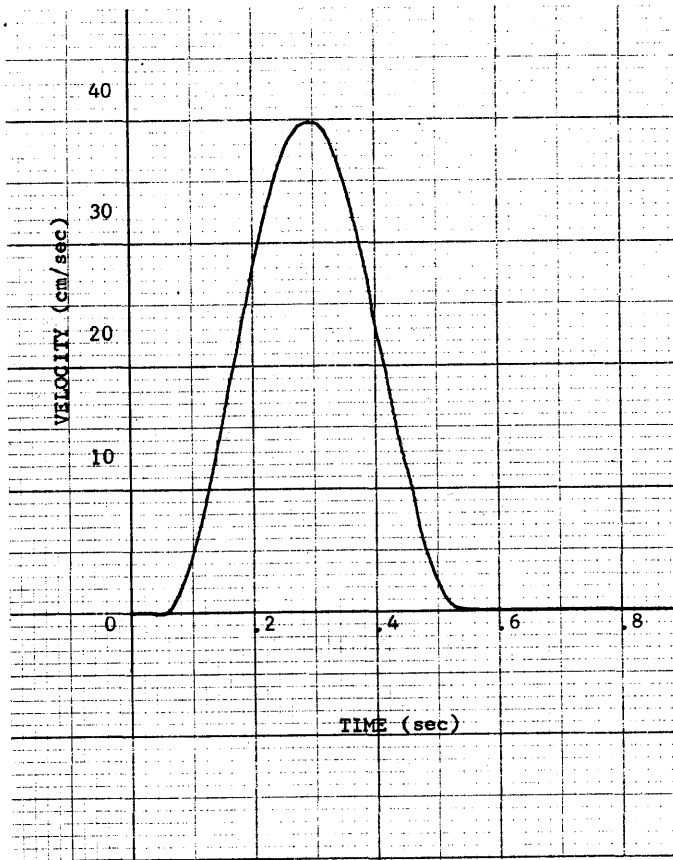


FIGURE 6—Generated left ventricle velocity

digital computer using Pactolus, they must be written in finite difference-differential form. Using a central difference formulation the equations become,

$$\frac{dH_n}{dt} = -V_n \frac{(H_{n+1} - H_{n-1})}{2\Delta x} - \frac{a^2 (V_{n+1} - V_{n-1})}{g \cdot 2\Delta x} \quad (11)$$

$$\frac{dV_n}{dt} = -V_n \frac{(V_{n+1} - V_{n-1})}{2\Delta x} - g \frac{(H_{n+1} - H_{n-1})}{2\Delta x} - \frac{f V_n |V_n|}{2D} \quad (12)$$

By writing the equations in this manner they remain nonlinear, but they are now only functions of time and the index, n. Note that the last term in (12) is written to incorporate reverse flow. Equations (11) and (12) must be solved for each increment. Each equation will then describe a point in a section of the aorta or vessel. The equations used for solving the distal boundary conditions are,

$$V_{n+1} = -\frac{g(2\Delta x)}{a^2} \frac{dH_n}{dt}$$

$$-\frac{g}{a^2} V_n (H_{n+1} - H_{n-1}) + V_{n-1} \quad (13)$$

$$H_{n+1} = -\frac{2\Delta x}{g} \frac{dV_n}{dt} - \frac{V_n}{g} (V_{n+1} - V_{n-1}) \quad (14)$$

$$-\frac{f(\Delta x)}{gD} V_n |V_n| + H_{n-1}$$

Before proceeding, the values for the wave velocity, a, and the friction factor, f, must be determined. The wave velocity is given by,

$$a^2 = \frac{K/\rho}{1 + KDC_1/Et'} \quad (15)$$

The wave velocity is derived in the waterhammer problem in which the compressibility of the fluid is taken into consideration; however, in a blood flow analysis the compressibility of the blood is negligible when compared to the compressibility of the blood vessel wall. In this paper the bulk modulus of the tube is considered instead of the bulk modulus of the fluid. This is a reasonable approach because the structure of the aorta wall influences the traveling pressure wave and the instantaneous fluid velocity.

The equation for calculating the bulk modulus, sometimes referred to as the modulus of volume elasticity,⁵ is,

$$K = \frac{Et'}{D} \quad (16)$$

The other constant to be determined is the friction factor. It is assumed that flow is laminar after leaving the semilunar valve. Some investigators have indicated that the Reynolds number, R, for flow in the aorta is between 1000 and 2000.³ This suggests that the flow is laminar. In this simulation the friction factor for laminar flow in smooth tubes was used.

$$F = 64/R \quad (17)$$

The data for the segment of aorta under consideration are:

- L = 100 centimeters
- n = 3
- Δx = 25 centimeters
- T' = 0.159 centimeters
- D = 1.27 centimeters
- E = 3 × 10⁷ dynes/cm²
- K = 3.76 × 10⁶ dynes/cm²
- ρ = 1.06 gm/cm³
- a = 1330 cm/sec

$$\begin{aligned} a^2 &= 1.77 \times 10^6 \text{ cm}^2/\text{sec}^2 \\ g &= 980 \text{ cm}/\text{sec}^2 \\ R &= 1000 \\ f &= 0.064 \end{aligned}$$

The boundary condition at the entrance to the aorta is the pressure from the left ventricle, which is generated by a function generator. The other boundary condition at the entrance of the aorta is the velocity function. The velocity function at the entrance to the aorta is nonlinear and indicates reverse flow. This reverse flow is identified in the pressure waveform of the human by a notch in the descent of the pressure waveform and is called the dicrotic notch.

As discussed previously the pressure at the distal end of the tube is selected, and the boundary condition for the velocity at the distal end is predicted. This can be done for an infinite number of pressures; hence, predicting a complete range of solutions. The technique is also reversible, a velocity can be selected and a pressure determined that will fit the required boundary conditions. This method is difficult to implement using Pactolus and an alternative method using the special blocks was implemented to calculate the distal boundary conditions. The special blocks are subprograms written especially to calculate these boundary conditions.

The results are those expected for flow in an elastic arterial system. The pressure at the distal end is higher than that at the first station, and the velocity is lower than the first station (See Figures 7 and 8). The phenomena is discussed in a previous section of this paper. These results indicate that the equations derived can describe the pressure and velocity for a segment of an elastic blood vessel. It has been shown that knowing one boundary condition at the distal end the other boundary condition at that end can be found by a random search for a stable solution or by special blocks for calculation of the distal boundary conditions.

The Pactolus simulation technique and new continuous system simulation programs can be of great value in the simulation of the vascular system. The convenience of having the equivalent of a medium sized analog-hybrid computer to use and a rapid technique for solving the identifying equations will help in the complete identification of the vascular system. Future simulations will be performed to more accurately defined vascular system parameters.

Elastic tube experiments

Procedure

Experiments using an extra flexible polyvinyl tube and pure gum rubber, latex, tube were carried out. The experiments were used to verify the results of the

simulations and to confirm the validity of the vascular system equations. Pressures were measured as functions of time at the entrance and distal end of a one meter length of tube, and the velocity of the pressure wave was determined. A roller pump was used to generate the fluid forcing function. The fluid was water.

Pressures were measured using semiconductor strain gages. Two gages were used for each measurement. One gage was used as the active transducer, and the other, a dummy gage, was used for temperature compensation. Semiconductor strain gages are very sensitive to temperature variations so a constant temperature environment was constructed by completely enclosing the transducer and the measured portion of tubing with a shield of styrofoam. Semiconductor strain gages were chosen because of their high gage factor, 130, and resistance to fatigue. The two gages were connected as half of a resistance bridge circuit, and two precision helipot were connected as the other arms of the bridge. The two transducers were placed one hundred centimeters apart. The first transducer (at the upstream end) was placed approximately seventy-five centimeters from the pump outlet, thus eliminating any unwanted effects

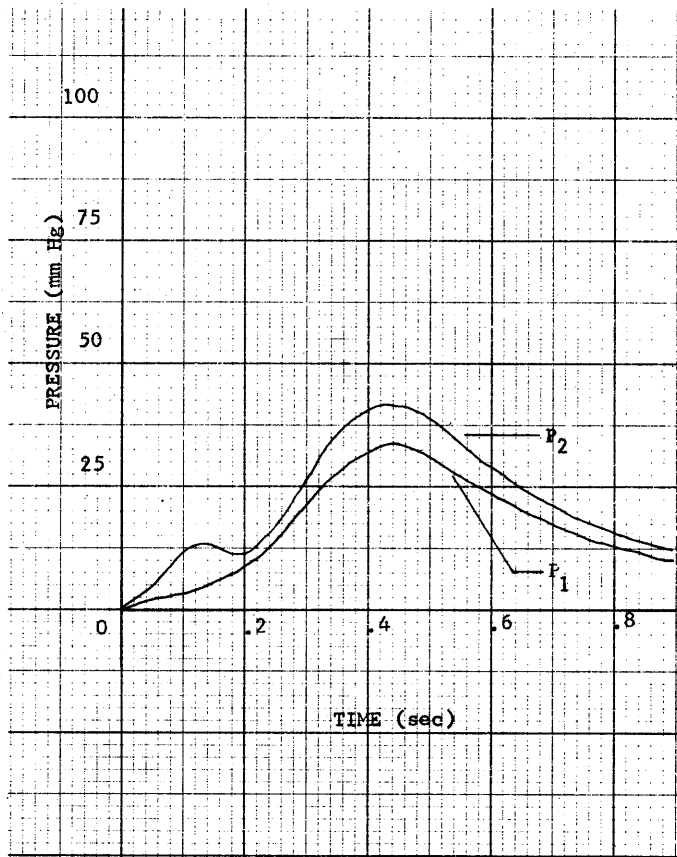


FIGURE 7—Predicted pressures for Pactolus simulated aorta

due to the pump. These particular strain gages were designed for constant current (10 milliamps) excitation, since a constant current source helps to eliminate the nonlinearity of the gages. The output of each resistance bridge was connected to an x-y recorder, and the pressure waves were then recorded as functions of time for the entrance and exit of the tube under investigation.

The pressure wave velocity was determined using an electronic counter. Each of the outputs of the resistance

bridges, one at the input and one at the output of the one hundred centimeter segment of tube, was connected to a vacuum tube voltmeter. The outputs of the voltmeters were patched to an analog computer and amplified. These amplified signals were transmitted to the electronic counter. The counter was used to obtain the time interval for the pressure wave to travel from the first transducer to the second transducer. Knowing the distance between the transducers and the time

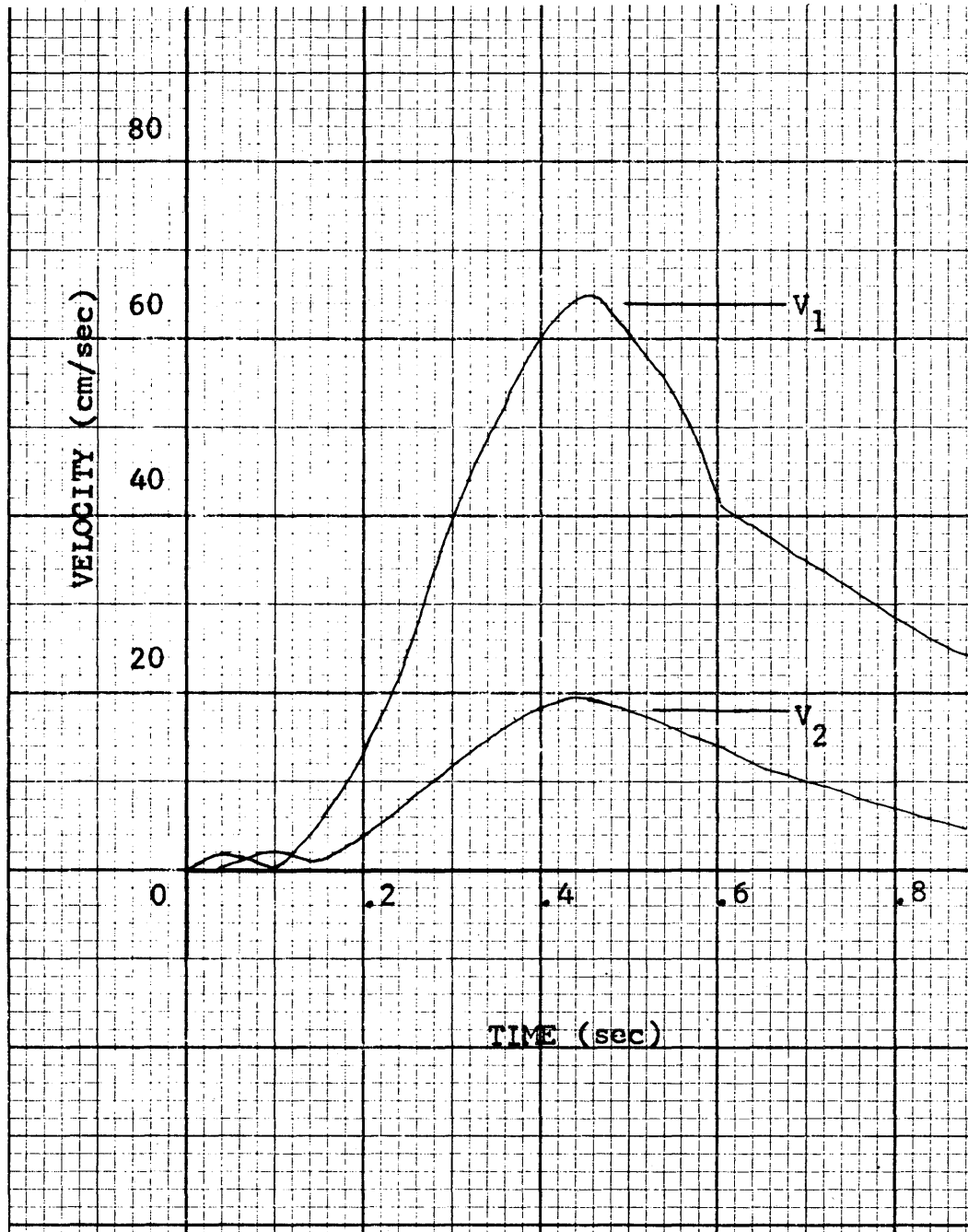


FIGURE 8—Predicted velocities for Pactus simulated aorta

interval, it was a simple calculation to obtain the average pressure wave velocity.

Results

The recordings of the pressures as functions of time for the flexible polyvinyl tube are shown in Figure 9. The pressure wave for the entrance, P_1 , is more rounded at the summit than the outlet pressure, P_2 . The evidence of negative flow is also visible in both waveforms. The small notch in the waveform as the pressure descends indicates negative flow. In the aorta of the human this notch is called the dicrotic notch. Note, the notch is more distinguishable in the pressure wave for the outlet of the tube. This phenomenon is also seen in the human aorta. The pressure wave peaks downstream, and the amplitude of the wave increases. This also occurs in the human aorta. The wave velocity calculated was ten meters per second, a value that is in the physiological range.

The pressure wave forms for the latex rubber tubing are given in Figure 10. Latex tubing is more rigid than the polyvinyl used, and the test results confirm this. The pressures recorded are not as large in amplitude. The notch on the descending portion of the waveforms

is less noticeable on the input pressure and is almost damped out on the exit pressure. This indicates a larger modulus of elasticity for the latex tubing. Again the results are similar to those obtained from the human aorta. The pressure wave velocity was calculated and found to be approximately twenty meters per second. This velocity is close to the physiological maximum for the aorta.

Pressures found by experimentation with the elastic tubes are similar in form to those measured in the human aorta and to those predicted by simulating the vascular system equations. The polyvinyl tubing appears to be a better model for the elastic aorta, where as the latex tubing appears to be a better model for the more muscular arteries. The results of these experiments are encouraging. They substantiate the validity of the proposed mathematical model for blood flow in a segment of artery and provide encouragement for future simulations.

SUMMARY

This paper is the presentation of a technique to be used in identifying the vascular system. The results presented indicate that the mathematical model is valid as a

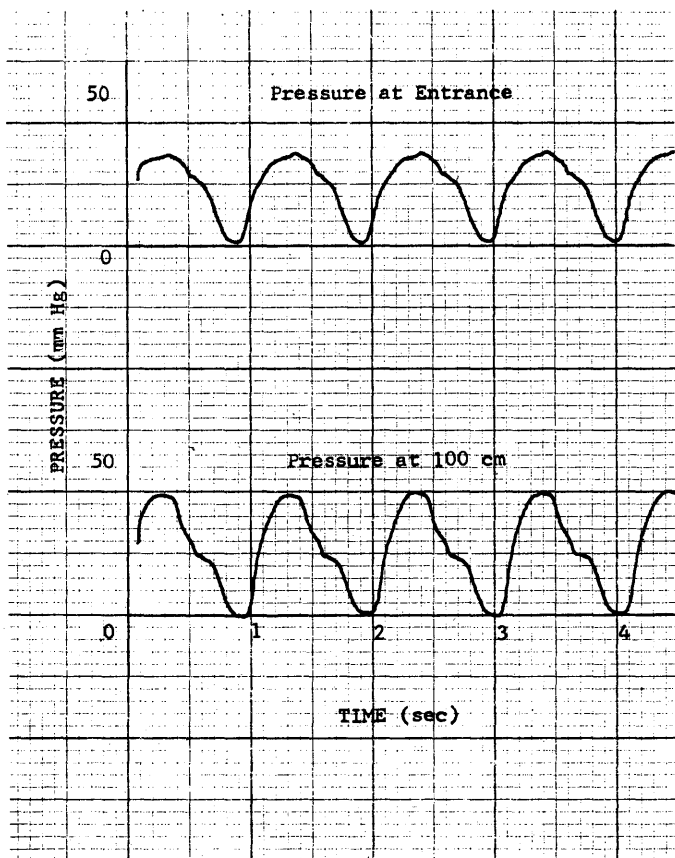


FIGURE 9—Measured pressures for polyvinyl tube

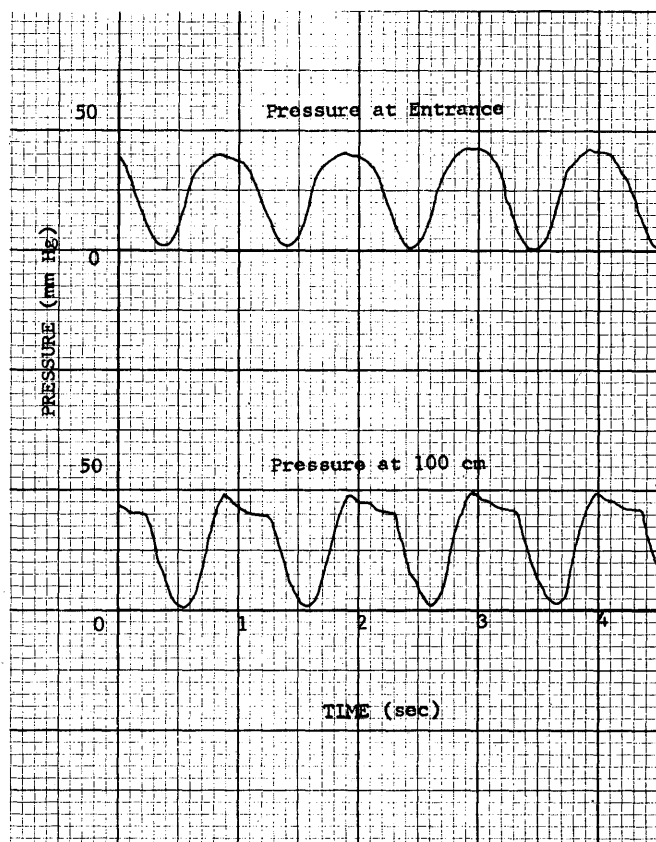


FIGURE 10—Measured pressures for latex tube

representation of the flow of blood in an elastic blood vessel. Digital computer continuous system programming methods proved adequate for simulating the complex equations. The Pactolus program predicted the velocity and pressure functions for the aorta. The possibility of using digital simulation programs is enhanced by the future development of digital continuous system simulation programs.

The future of the model and technique is hopeful. If the vascular system is satisfactorily identified the physiologist and the engineer will learn much about the mechanisms of circulation. Prediction of the velocity, pressure, and change in shape of the pressure wave will be valuable in determining the functioning or malfunctioning of parts of the vascular system. By expanding the techniques introduced in this paper it will be possible to accurately predict the condition of a vascular component.

REFERENCES

- 1 V L STREETER
Fluid mechanics
McGraw-Hill Book Company Inc New York 1966 4th ed
- 2 V L STREETER W F KEITZER D F BOHR
Pulsatile pressure and flow through distensible vessels
Circulation Research 13 5 1963
- 3 D A McDONALD
Blood flow in arteries
Edward Arnold Ltd London 1960
- 4 R D BRENNAN H SANO
A digital analog simulator program for the IBM 1620
Proceedings Fall Joint Computer Conference 299-312 1964

5 V HARDUNG

Propagation of pulse waves in viscoelastic tubings
Handbook of Physiology Williams and Wilkins Co Baltimore
1963 Vol 11

APPENDIX

Variables and constants used in derivation of system equations

- a = pressure wave velocity
- D = inside diameter of the vessel
- E = Young's modulus of elasticity
- K = bulk modulus of elasticity
- t^1 = wall thickness of the vessel
- P = pressure
- H = pressure head
- V = average velocity of the fluid
- ρ = density of the fluid
- t = time
- x = distance along the tube
- Δx = differential length of tube
- A = cross sectional area of the tube
- τ = maximum shearing stress at the tube wall
- π = constant (pi)
- F_x = force in x-direction
- M = mass of element under consideration
- f = friction factor
- V = volume
- T = tension
- C_1 = unity (constant) related to tube material

A computer system for real-time monitoring and management of the critically ill*

by DAVID H. STEWART, DAVID H. ERBECH
and HERBERT SHUBIN

University of Southern California
Los Angeles, California

INTRODUCTION

During the past decade a number of specialized hospital units have been developed for monitoring and care of critically ill patients. These units, which include coronary and intensive care facilities, trauma, renal dialysis and post surgery recovery wards, now provide up to five percent of beds in acute community hospitals.¹

Patients with acute circulatory or respiratory failure are often referred to these specialized units. An example is the patient in circulatory shock. This condition may result from the complications of a "coronary," from severe bleeding from a duodenal ulcer, or from a number of other disease processes. The patient in shock characteristically has low blood pressure and reduced blood flow. With inadequate circulation to his brain he may become stuporous or comatose. His respiration may fail and kidneys cease to put out urine.

Assessment of the circulatory and respiratory status of such a critically ill patient requires measuring a number of variables: hemodynamic (i.e., arterial and venous pressure, blood flow and volume), electrical (i.e., the electrocardiogram), blood gases (i.e., oxygen, carbon dioxide and pH) and blood constituents (i.e., potassium). Repeated assessment of these variables is required since the critically ill patient is not in a steady state, but is very labile, and undergoes rapid and often unpredictable changes in status.

*This investigation was supported by grants from The John A Hartford Foundation, Inc., New York and the U.S. Public Health Service (HE-05570 and HE-07811—National Heart Institute) and Division of Hospital and Medical Facilities (HM-GM 00533).

Current problems

The organization and operation of critical care units require a major commitment of highly trained physicians, nurses and technicians, as well as substantial physical resources, in order to care for a relatively small number of patients who require numerous diagnostic and therapeutic services. Efficiency of operation is needed to reduce the work load on the professional personnel. Present manual methods of data collection and logging impose great demands on the staff, particularly during night hours and weekends when staffing is reduced.

Clinicians are concerned not only with the acquisition, but with the timely organization, analysis, and display of data to assure their immediate usefulness at the bedside. The availability of such information has been life saving, particularly in coronary care units where mortality has been reduced from 30 to less than 15 percent. Precise documentation of reduction in mortality rates in more generalized critical care units is as yet not as clearly established.

Experience

The Shock Research Unit, a specialized clinical research facility, has been developed for the triple purpose of rendering intensive care to seriously ill patients, studying underlying mechanisms of the disease process, and developing new techniques of evaluating seriously ill patients. In mid-1963, a computer was obtained and a system developed for monitoring patients.^{2,3,4} Algorithms were derived to convert electrical signals from transducers attached to patients into their cus-

tomary physiological units.^{5,6,7} With this system 11 primary measurements and 25 derived variables are recorded and displayed with a frequency which ranges from once a minute to once every twenty-four hours. These variables are summarized in Figure 1.

The present system operates on a 24-hour basis and is run exclusively by clinical personnel with minimal intervention by computer and engineering staff. Over 400 patients have been monitored and studied on a routine basis with the system. The feasibility and reasonableness of using computers to collect and analyze physiological data in a clinical environment has been demonstrated. An indication of its value is provided by the procedure for determining cardiac output. The time required for measurement of the indicator dilution curve and calculation of the cardiac output, which was approximately 45 minutes by manual methods, has been reduced to only 5 minutes with the computer. Experience with this system has shown that certain procedures with respect to treatment, which are currently performed manually, such as the administration of fluids and medications, are adaptable to computer control.

A shortcoming of the system has been the single

Primary Signal	Sensor	Displayed Variables
Electrocardiogram	Surface electrodes (Sanborn)	heart rate
Arterial pressure	Strain gauge transducer (Statham)	a. systolic pressure b. diastolic pressure c. mean pressure d. variation between highest and lowest systolic pressures in read interval e. pulse rate f. pulse deficit
Venous pressure	Strain gauge transducer (Statham)	a. mean pressure b. respiration rate
Optical density of arterial blood	Densitometer (Gilford)	a. cardiac output b. cardiac index c. work done by heart d. stroke work e. vascular resistance f. central blood volume g. mean circulation time h. appearance time
Temperatures	Thermistor (Yellow Springs Instrument Co.)	a. rectal b. skin, finger c. skin, toe d. skin, thigh e. skin, arm f. ambient air
Urine output	Shock Research Unit design	a. urine output/hour b. urine output last 5 min. c. cumulative volume
Manual input	Manual entry unit	a. PO ₂ b. PCO ₂ c. pH d. oxygen saturation e. plasma volume f. red cell volume g. medications and fluid administered h. height, weight, age

FIGURE 1—Primary and displayed variables

channeled unbuffered data path. Dynamic variables, such as arterial and venous pressure and the electrocardiogram, do not require sampling at high rates, the maximum being below 1000 sp/s. However, the analysis process requires repeated readings of moderately complex wave forms. For example, arterial pressure, under some circumstances, may be sampled almost continuously for 30 seconds. When the volume of blood pumped by the heart (cardiac output) is calculated, it is necessary to sample the densitometer for as long as eighty seconds. In addition to these considerations, it has been necessary to monitor more than one patient concurrently. This period of processor dependence upon I/O may occur during the time when the process is critically real time and this may result in loss of medical information which is needed immediately by the physician at the bedside.

The record keeping responsibilities of the physician, nurse and technician have posed a major problem. Numerous forms have been required for the history, findings on physical examination, progress reports, routine and special laboratory studies. This has created reams of poorly organized, non-cross-referenced data. The initial system has been incapable of prolonged dialogue for such manually entered data.

Based on our experience with the initial system over a 5-year period, we have developed a new system for use in the environment of the critically ill patient. A major goal of this system is to automate the critical care environment, with specific attention to the following tasks:

Monitoring and measurement

1. Monitoring of cardiovascular, respiratory and metabolic variables.
2. Measurement of blood samples.
3. Measurement of red blood cell and plasma volume.
4. Measurement of intake and output of fluids.
5. Measurement of body weight.

Control

1. Infusion of fluids and medications under computer control.
2. Regulation of ventilators with respect to the pressure and volume settings and the relative concentration of gases they deliver.
3. Control of body heating and cooling devices.
4. Control of cardiac pacemakers.

- Control of the flushing procedures for intra-vascular and urinary catheters.

Records

- Comprehensive record keeping, including entry, organization, storage, return and display of non-sensed data which must be available for a well organized clinical facility.

Functional requirements of the system

The requirement for multi-usage systems design is apparent in the critical care environment. The computer needs to simultaneously acquire data, control many processes in the ward and manage the retrieval and display of information on both the current and previous status of the patient.

Real time acquisition and display

To provide input to the computer, analog devices are attached to the patient to sense physiological activity. These signals are continuously presented on a bedside oscilloscope by signal conditioning hardware as shown in Figure 2. The signal conditioners amplify the signals and perform a preliminary analysis of wave forms. The output of these conditioners is attached to the multiplexer, then to the A/D convertor interfaced to a computer channel. The computer must also be capable of real time analysis of these signals. As an example, the computer must be capable of recognizing individual components of the EKG signal (i.e., the QRS wave), find the pulse wave

in the arterial pressure signal, calculate maximum, minimum, and mean arterial pressure, before the next heart beat occurs. Although the heart rate is normally about 80 per minute, rates in excess of 160 per minute are often encountered in the critically ill patient.

An on-line blood chemistry unit is being developed for use at the bedside. This will enable the computer to monitor PO_2 , PCO_2 , pH and electrolytes as well as blood volume measurements. The timing requirements for this type of monitoring are not based upon the real time correspondence with the body but rather upon the techniques of measurement. Measurement can be as simple as the sampling of the output of an electrometer connected to an electrode or as complex as spectral analysis.

The computer's ability to analyze signals at beat-to-beat rates is an asset during moments of crisis and during procedures such as drug administration. However, if such detailed information were accumulated continuously, the physician would be overwhelmed by a flood of data. There is a need, therefore, to provide a schedule of monitoring and reporting which will insure that patients are under constant surveillance, but the physician and computer are relieved of an overwhelming data management problem. The monitoring schedule under development at the Shock Research Unit allows for two modes of operation. The physician is able to request beat-by-beat measurements. The routine mode of monitoring, however, is based upon the ability to define or predict the amount and rate of change to be expected in any given variable. Heart rate and rhythm are monitored almost continuously because abrupt changes in these variables may be followed by rapid deterioration in patient status. However, skin and rectal temperature are monitored at 15-minute intervals because significant changes in these signals are generally of longer duration. After the data are converted into physiological units, they are analyzed to determine its significance. Is the change measured in a particular variable to be expected statistically and does it correspond to changes in other variables? If the variations are statistically non-significant then the data are discarded. However, if there has been a significant change this is logged in the computer records. The ward staff is immediately alerted and informed as to which variable has changed.

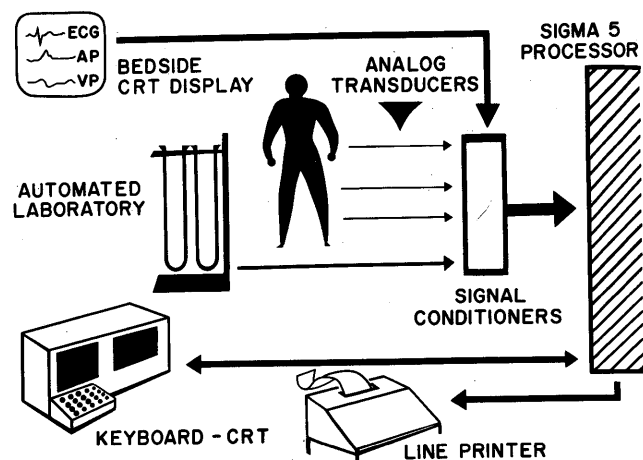


FIGURE 2—Real time acquisition and display

Digital control

Figure 3 shows diagrammatically a number of new concepts which in part have their rationale in the advances in process control engineering in critical processes. The time has come, technologically, when computers can be programmed to assist and control medical procedures under the direction of the physician and nurse. Computers can control the rate of fluid and medication administration through actuation of pumps and injectors. Such a pump is now in engineering phases at the Shock Research Unit. This pump is intended for long term administration of fluids or medications from a bottle. The nurse will be able to request the computer to administer the fluid or medication at specified rates. The computer will then check a rate table to see if the rate is unusual. If so, it will ask the nurse to confirm this rate. If not, it will adjust the pump speed much more accurately than is done by the drops per minute method currently in general use. It will make entries to the fluid intake log and then monitor the bottle for low level. An increase in central venous pressure may indicate over-administration of fluids. If central venous pressure (which is computer monitored) should rise above a predetermined level, the pump can be either shut off or slowed down and the medical staff notified. It is a similar technological problem to have the computer adjust and monitor specialized equipment for assisting the patient's respiration. The interaction of the nurse, computer and ventilating device would be similar to that of the pump and would include safety

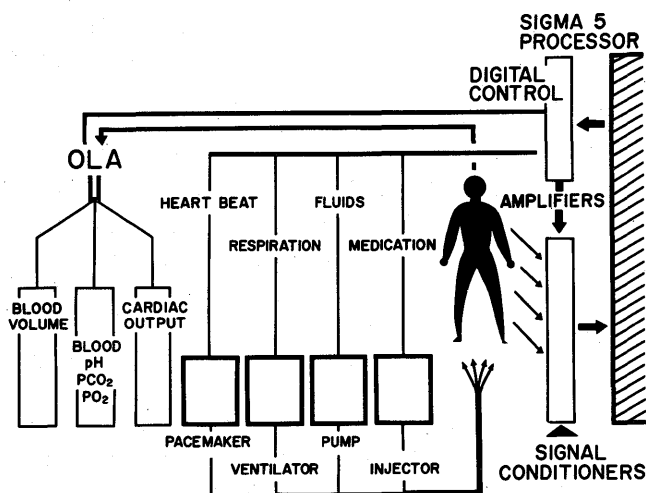


FIGURE 3—Real time control

mechanisms and confirmation of orders before adjustments are made.

Some control devices, such as cardiac pacemakers, are already in use in many coronary care units. Control and monitoring of these devices, if given to the computer, can insure a level of reliability previously unobtainable.

The chemical analysis subsystem now under design at the Shock Research Unit is called the "On-Line Analyzer" (OLA). With the OLA system the computer will be able to control withdrawal of blood samples, reinfusion of blood samples, and mixing of samples with chemicals where necessary. Results of these chemical tests will be input directly to the computer. This will permit moderately complicated procedures to be done at intervals related to their clinical usefulness rather than their complexity.

Another advantage of digital control is the ability to adapt signal conditioning units to the course of the patient. An example of this adaptability would be the control of amplifiers and filters such that obscured regions of a signal may be highlighted for computer analysis. Another feature is the ability of the computer to perform sensor calibration checks automatically, thus freeing nurses and paramedical personnel from time-consuming chores which remove them from the patient.

Data management

The data management system must be versatile in structure and in its ability to meet the needs of the clinically and the statistically oriented staff (Figure 4.) Numerical and textual data are stored in a form easily retrievable for ward display and for later statistical analysis. It has been noted that the ability to retrieve textual as well as numerical information in a neat format will greatly help the physician in the care of the patient.

Displays. Included in the system are displays located at each bedside and in the physician's study. Display requirements for an intensive care ward present a multifaceted problem. The solution is found in providing two types of displays.

1. Large screen TV

Currently available devices used for interaction with the computer system (keyboard CRT displays) are not sufficient for all requirements, in that they fail to provide a sufficiently large display, having been de-

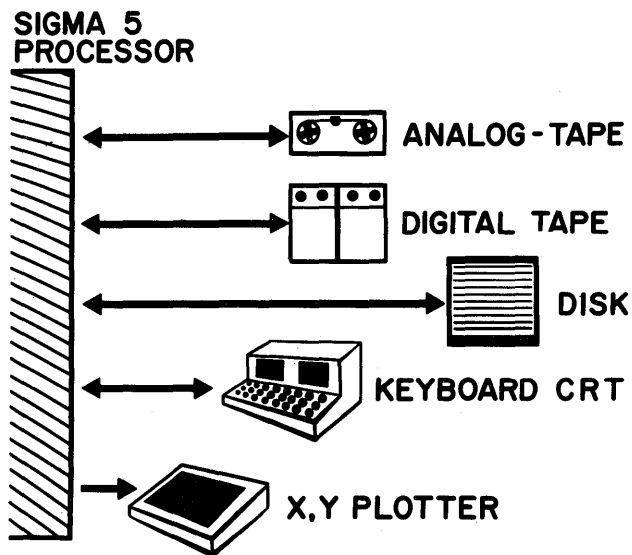


FIGURE 4—Data management

signed for desk top usage. Personnel in the ward are usually on the move and must be able to read displays accurately from varying distances. If emergency information must be projected on a keyboard display, competition between the user and the computer system obviously must be avoided. The solution has been to acquire computer-controlled character generators for driving large screen closed circuit television monitors which are strategically located to provide a display visible throughout the room. This device is only available to the computer and displays only the most current significant information concerning the patient.

2. Keyboard displays

Physician's and nurse's records are entered into the computer system through the use of a keyboard device providing 2,048 characters of display for selection of options and message formation. From this bedside terminal, the physician reviews the entire patient file and enters comments and impressions. Nurses enter information which include drugs administered, procedures performed and non-sensed data regarding the patient's condition. The information is entered in a multiple choice format. This is necessary because few ward personnel can type competently. While free text would function as narrative, experience in our own unit shows that information is gathered

more dependably when a detailed format is followed than when users have a free choice as to which information is recorded. Another approach which has been considered is the use of a clerk to enter the notes of the physician, nurse and technician into the computer. The use of multiple forms, even in this manner, is inconsistent with the principle of having all information available at one source. Since the use of such a system is subject to the adequacy of staffing, delays in entry of information may also be interposed.

In addition to the alphanumeric display, a storage CRT for graphical display of all data available at the keyboard is provided. From this CRT terminal the physician requests the computer to locate wave form data on an analog tape recorder and display it on the oscilloscope associated with the terminal. Hard copy of all information in the system is also logged on off ward teletypes for the purpose of back up, a critical requirement in the event of machine failure. A high speed line printer develops the entire hard copy patient file at the time of discharge for inclusion in the general hospital records.

Plotter. There are several reasons for hard copy graphical display. Foremost is the fail safe problem. Physicians make extensive use of trend plots and these are available at the bedside on the CRT displays. However, in order to insure that graphical display describing the patient's course is available in the event of machine failure it is necessary to generate hard copy on a continuing basis. Second, hard copy display is required for the hospital record. Traditionally nurses have kept some form of trend recordings on the patient's vital signs. Trend plotting of major parameters for fail safe, proceeds automatically without need of request. In this system the physician or nurse can develop graphical presentation on the CRT not only of the vital signs but also of a number of other variables of interest. They can then request that this be prepared on hard copy for the hospital record, or for study at areas remote from these facilities.

Analog Tape. During monitoring, the physician can remotely engage an analog tape recorder to record wave forms as they are sensed from the

patient. However, the amount of information which the physician finds useful from such extensive recording requires a substantial editing effort. The computer is also programmed to switch on the tape recorder when a major change occurs in a variable and switch it off when the values for the variable return to specified limits. Under both options, the point on the analog tape where items of interest are stored, is automatically recorded, thus facilitating tape editing under computer control.

Software

In solving the functional requirements, the following philosophies have arisen. Some special programs may have their own hardware interrupt priority level, but programs executed by the program scheduler do not have a priority until they are initiated. These priorities are determined by the need to minimize fluctuations in the actual initiation frequencies and insure the regularity of acquisition of time sequenced data (Figure 5). Within classes of priorities, the actual activation position in the queues is determined by the requested execution frequency, with items of highest frequency coming first.

The problem in many real time systems is that the total execution time of the programs to be scheduled is greater than the recycling time of the program executing with the highest frequency and, therefore, programs of a lesser priority are periodically interrupted and displaced from their cycling interval.⁸ However, this is not a problem here since the total execution time easily fits into a scheduler frequency determined by the program of highest frequency. The problem is that the processing in each program must wait for a series of timed analog inputs and the need is thereby created for two types of scheduling so that these waits may be interleaved with other processing.

Monitor type functions are usually executed at the same priority level as the program requesting them. All non-analog I/O for any one device is run at one level to relieve the re-entrancy problems. Each actively monitored bed has its own core buffer to insure patient file updating speed as well as safety. This buffer is written onto the disk after each data update group as well as other control information has been calculated to insure that there is no loss of valuable data in case of system failure. This output is of course overlapped with

Analog scheduler clock

Program scheduler clock

Monitor time clock

I/O interrupt

Special external interrupts

Analog scheduler

Highest program priority

Keyboard display handler

Intermediate program priority

Program scheduler and loader

Lowest program priority

FIGURE 5—Relative hardware priorities

processing, but input is not overlapped except by higher level routines.

The various tables and queues throughout the system are list structured and utilized so that no garbage collection is needed. As items are deleted their positions are set to "unused," and new assignments are always made from the physical head of the list.

Core memory has been divided into three sections. The monitor functions and I/O handlers reside in the first section. The second section contains the schedulers, the resident foreground programs, and the necessary modules from the Fortran systems library. The third section is for non-resident foreground, or background since the area is large enough for the Fortran compiler to run. Some of the applications programs are described in the references.^{5,6,7}

Schedulers

The multiprogramming philosophy adopted requires simultaneous processing of programs and the reading of many analog lines for several patients at data rates up to 1000 times per second. Analog pre-processing is being used, but moderate rate digital sampling and analysis cannot be entirely replaced. To facilitate a versatile selection of both channels and frequencies and allow for the timed reinitiation of various monitoring programs, a program scheduler and an analog scheduler have been incorporated into the system.

Cycling of the initiation phase of the program scheduler is determined by its private hardware clock. It picks information from the program queue to determine the proper sequence for program executions. Servicing programs for the keyboard displays allow the clinical staff to insert or remove programs from this queue. When execut-

ing programs request analog I/O, they enter requisite information into a table known as the analog table, and return control to the program scheduler which searches the program queue for the next program awaiting execution (Figure 6A, 6B). The analog scheduler, running on a separate hardware clock and at a higher hardware priority, continually scans the analog table at a high rate for I/O requests at varied frequencies and channels, interleaving these requests. No attempt is made to time slice among applications programs of an equal priority except at these analog I/O request points. The overhead and complexity of going any further with time sharing far outweigh any value which might be realized. The execution speed of the processor used allows for the completion of the computations involved in far less time than significant change can occur in variables being monitored. However, as some of the

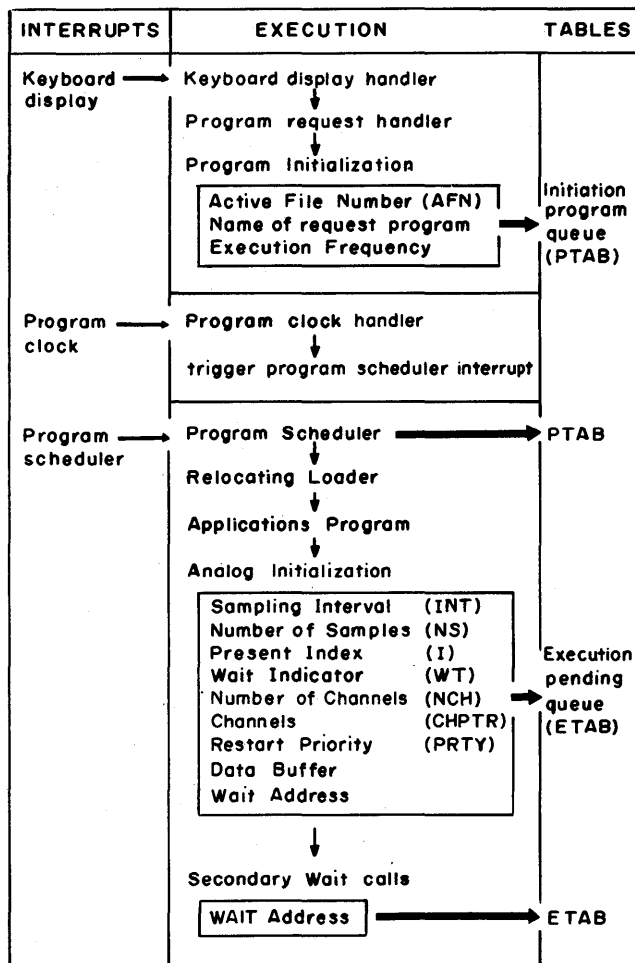


FIGURE 6a—Program initialization flow

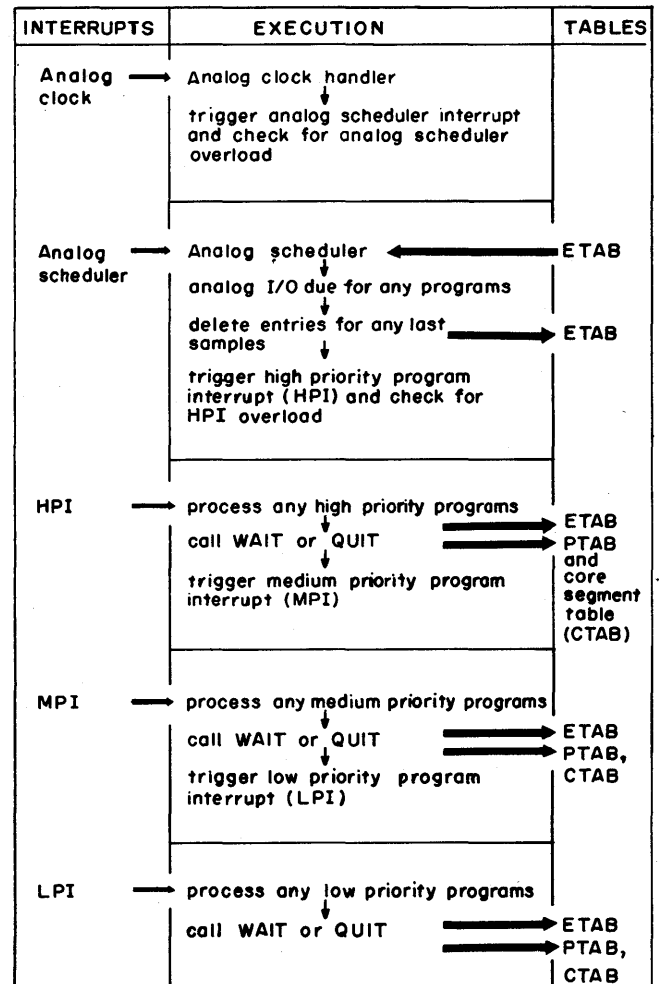


FIGURE 6b—Program and analog scheduler flow

analog I/O requires sampling to run many seconds at low rates (e.g., 80 seconds for cardiac output at 20 samples per second), processing must continue during these long waits.

An initialization procedure is used to insert programs into the program initiation queue (PTAB). This initialization routine uses four parameters—an active file number, a program name, a program reinitiation interval, and the keyboard display unit associated with this copy of the program. Programs are assigned to the first “unused” entry in PTAB. Using the reinitiation interval, the program is given an activation sequence number determined by placing it into the list structured queue with the re-establishment of pointers such that the program having the short reinitiation (or highest frequency) interval is at the head of the list. The scheduler reinitiation rate is an integral divisor of all the possible applications program rates.⁸ Also placed in the PTAB is a pointer to a list of analog channels to be read. Since there are inevitably times when one channel or piece of equipment is not functioning properly, these channel numbers are stored in a list which can be modified without recompilation of the applications programs. There is a library table which contains the names of all available applications programs and their location on the disk. This table is searched by program name and an index or program number is determined and placed into the program table. This is used by the relocating loader when the program is to be initiated.

Programs are picked from PTAB, loaded, and execution is started. Execution continues until either the analog scheduler or higher level programs interrupt, or until this program requests analog I/O, keyboard display I/O or change in execution priority. These requests are made through Fortran calls to assembly language sub-routines which use hardware trapping to access servicing routines. When analog I/O is requested the servicing routine places into the execution pending queue (ETAB) a reset interval frequency (INT), the number of separate read requests (RN), and a pointer to the channel numbers for each read (CHPTR), as shown in Figure 6A. A structure similar to that proposed by Fitzwater and Schweppe⁹ has been implemented to allow applications programs dynamic control of the amount of I/O overlap and execution priority. One parameter supplied to the analog scheduler

is an index (I) which is updated by the scheduler after each read. If the applications program informs the scheduler that it desires to have control returned to it before the completion of all I/O ($wt = \text{false}$), an IF statement is positioned after the analog call in the program to compare this I/O index with a processing index. If the processing index is equal to the read index, the program calls the WAIT function. Priority of execution (PRTY) may be higher or lower than the program scheduler and may vary from one analog request to another within one program.

Requests in the ETAB are entered by reinitiation interval (frequency) just as they are in the program queue. There is a multi-priority structure as shown in Figure 6B for determining the return of control, after completion of pending analog input. When each application program requested analog I/O, its restart priority level was inserted into ETAB. Three sections of the analog scheduler at three lower priority levels search this queue to determine if there are any programs requesting execution. The highest priority of execution that can be requested is for programs that must execute a section of code between each read. The next priority of execution available is at a lower interrupt level for programs which have a higher priority than the program scheduler, but do not need to execute between analog reads. The lowest execution priority level is below the program scheduler. When all analog I/O pending for a program has been completed, the program is assigned to the hardware priority level it requested, its entry in ETAB is deleted, and control is returned to this level if no higher level is waiting. Further analog requests by this program create a new entry in ETAB. When control is returned to each applications program, execution is continued until either the program requests more analog I/O via the analog initialization routine, or until the program calls the program deletion routine.

Data file system

One of the most important indices of medical data is time. As physicians review patients' records, they group information into “time lines” and then compare these against each other. (E.g., Given a patient's mean arterial pressure and blood volume at time 1, and the fact that the patient was administered a drug, is it significant that at time 2 he now has a higher blood pressure and

blood volume?) Groups of variables (e.g., hemodynamic variables) are usually considered together and called "summaries." One group recorded at the same time may be called an instance in a summary.

In the data management system there are two types of summaries. Type I is numeric information derived either directly or indirectly from the analog monitoring, and Type II is textual information which is inserted through the keyboard display devices. Type I information is stored as half word (16 bit) integers, and Type II as standard hexadecimal alphanumeric. Each instance of a summary contains a time, and a forward and a backward pointer to other instances of that summary. Instances are stored sequentially as they occur in time and are interleaved with instances of other summaries (Figure 7).

To facilitate the storage and retrieval of data, there is an outline for each patient file (Figure 8). This outline is the first record in each file and indicates which summaries are included in the file, and explicitly which parameters are being monitored for Type I summaries. Type II summaries have only their names in the outline and all structure exists in the data records. Information in the Type II summaries consists of

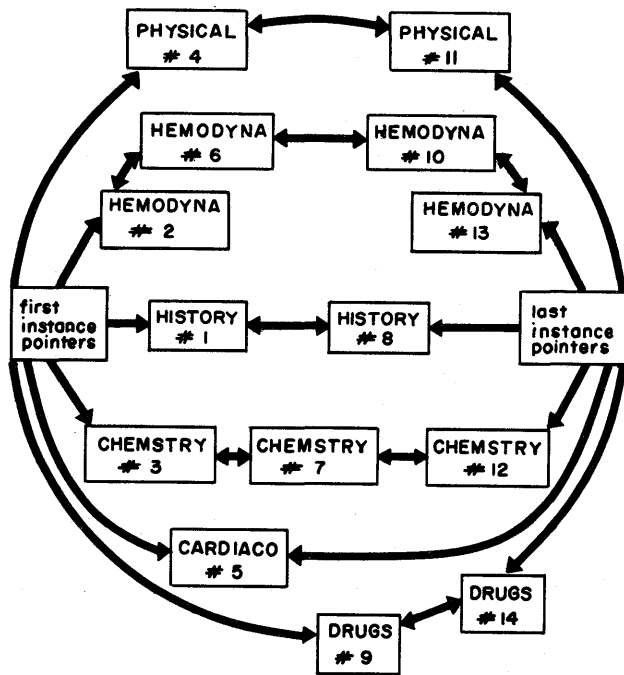
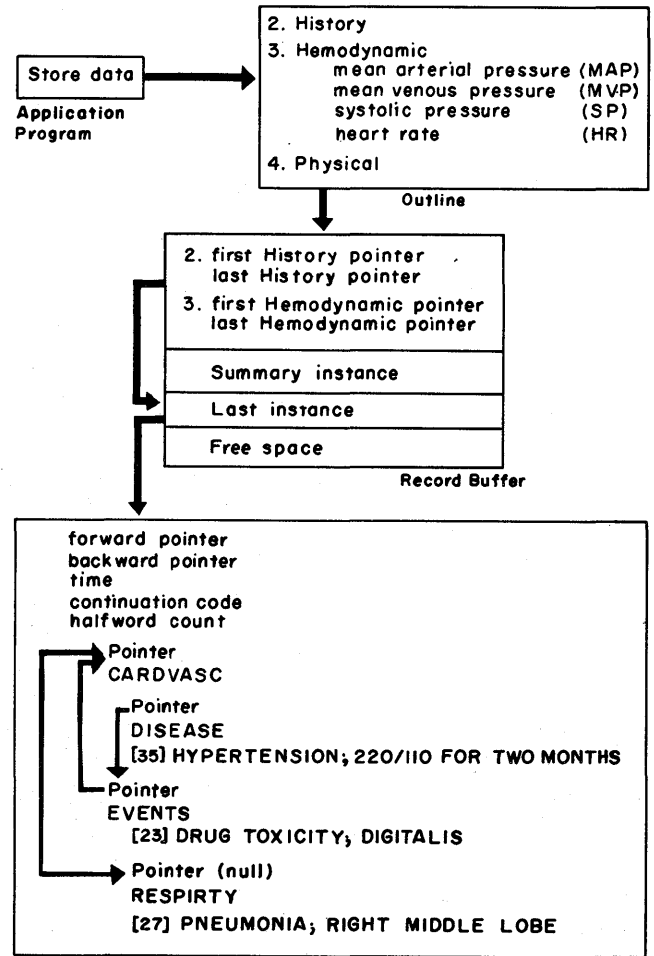


FIGURE 7—One physical record



A Textual Summary Instance

FIGURE 8—Data storage flow

pointers, names and values (Figure 8). Names (or qualifying names) are like indentation levels in an outline, and have a length of 8 hexadecimal characters. The values contain the actual textual information of interest and are of variable length. The first byte of the value is the character length of that value. There is a pointer with each name at each level, and this permits searching through the qualifying names at each level without searching through all of the text. Qualifying names can be appended to the beginning of the value and at any level the set of all subordinate names and values is a value. This allows the user to know as little about the outline structure as the main heading, or as much as the entire structure.

Each patient file can consist of many physical records where each physical record, for the purposes of buffering, is of a fixed length of 360 words of 32 bits. Each bed in the ward has its own buffer, but all other active files alternate the

use of a common disk I/O buffer. Instances of summaries of the patient file are stored into a buffer until it is approximately 95 percent full. Although the alphanumeric summaries were designed, because of their length, to allow for continuation in subsequent records, numeric summaries were not. Therefore, 100 percent buffer usage is uncertain. It is also desirable to allow space for record modification. There is a list of first and last instance pointers at the beginning of each data record where these pointers are in the same order as their respective summaries in the outline.

All active patient files are kept on the disk. There is a table in core and on the disk identifying physical records on the disk as patients' or unused records. Files of discharged patients are kept on magnetic tape, and there is a dictionary indicating where patient files are found on tape. Therefore, when a study is to be done on an old file, the file dictionary is scanned by patient number, and the system requests the pertinent tape to be mounted if it is not already mounted. The system then determines which disk records are available, and buffers from tape to disk. Oldest or least used files are automatically deleted from disk to make room if necessary. An active patient file table is maintained and is updated to include the patient number, active file number, present status of the file, and pointers into the patient's record. The first entries in the active patient file table are reserved for the active beds in the ward. The rest of the entries are shared by the terminals for any patient file of interest to the user of the terminal.

The rationale behind this structure is that no external dictionary is needed to retrieve any information. All structure is internal, and the user language is constructed such that the structure can be readily available to the inquirer. No two patients have identical files, and to force all files into one format would be time and space consuming. Even if two patients have the same monitored parameters, they will certainly have different lab tests, histories, physicals, etc., and they certainly will be stored at different time intervals.

Hardware

This description of computer and associated hardware (Figure 9) is included, but is not necessarily intended as a suggestion of the size of system required for intensive care monitoring.

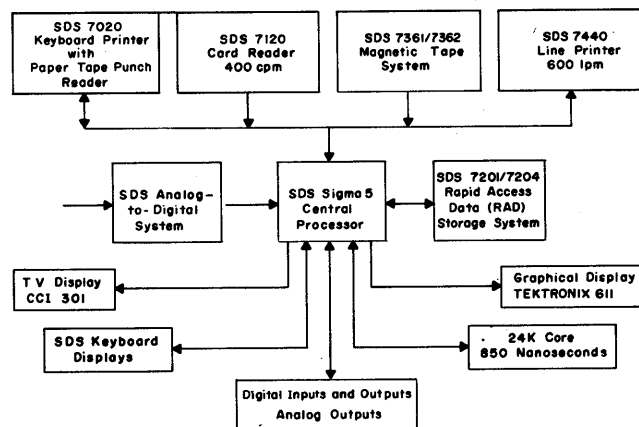


FIGURE 9

In fact, the computer system described is part of a larger systems study of the entire intensive care facility. System studies are being conducted to quantify the hardware and software needs as functions of both degrees of illness and number of patients. Attempts to simulate are fruitless at the present time, since the interface between clinical medicine and computer has not been sufficiently explored.

The SDS process control computer system consists of:

- A. *A Sigma 5 central processing unit:*
 1. 24,000 words of core storage
 2. 850-nanosecond cycle time
 3. memory protection
 4. 4 interval timers
 5. 40 levels of hardware interrupt
 6. additional register block (total 32 general purpose registers)
- B. *Peripheral I/O equipment:*
 1. rapid access data (RAD) storage system (17 m. sec. average access)
 2. 2 magnetic tapes
 3. 1 card reader 400 cpm
 4. line printer 600 lpm
 5. 2 teletypes
- C. *Data acquisition equipment:*
 1. multiplexers and their controls for switching input lines
 2. data channels for buffering input and output
 3. analog-to-digital converter (24kc with random addressing)
 4. digital-to-analog converters
 5. direct digital I/O for sensing and closing switches

D. Display equipment:

1. keyboard display
2. closed circuit display with character generator
(Computer Communication Inc. CC301)
3. storage CRT (Tektronix 611)
4. digital plotter (CalComp)

ACKNOWLEDGMENTS

We would like to express our appreciation to M. H. Weil, Director of the Shock Research Unit and to J. A. Trotter, H. R. Joly and L. D. Cady, Jr., with whom various aspects of this subject were discussed.

The contributions of Mrs. Edith O'Neill, Mrs. Bess Strauss, and Mrs. Helene Wheeler in preparation of this manuscript are also gratefully acknowledged.

REFERENCES

- 1 Commission for Administrative Services in Hospitals (C A S H) 4777 Sunset Boulevard Los Angeles California 90027 Internal Report 1966
- 2 M A ROCKWELL H SHUBIN M H WEIL
Shock III a computer system as an aid in the management of critically ill patients
Comm of the ACM Vol 9 no 5 May 1966
- 3 R L PATRICK M A ROCKWELL
Patients on-line
Datamation Vol II no 9 Sept 1965
- 4 R E JENSEN H SHUBIN P F MEAGHER M H WEIL
On-line computer monitoring of the seriously ill patient
Med & Biol Engng Vol 4 pp 265-272 1966
- 5 H SHUBIN M H WEIL M A ROCKWELL
Automated measurement of arterial pressure in patients by use of a digital computer
Med & Biol Engng Vol 5 pp 361-369 1967
- 6 H SHUBIN M H WEIL M A ROCKWELL
Automated measurement of cardiac output in patients by use of a digital computer
Med & Biol Engng Vol 5 pp 353-360 1967
- 7 H SHUBIN M H WEIL
Efficient monitoring with a digital computer of cardiovascular function in seriously ill patients
Annals of Internal Medicine Vol 65 no 3 Sept 1966
- 8 H WYLE G J BURNETT
Management of periodic operations in a real-time computation system
Proc FJCC 1967
- 9 D R FITZWATER E J SCHWEPPE
Consequent procedures in conventional computers
Proc FJCC 1964

Computer system for research and clinical application to medicine

by T. ALLAN PRYOR, REED M. GARDNER and W. CLINTON DAY

*University of Utah and Latter-day Saints Hospital,
Salt Lake City, Utah.*

INTRODUCTION AND GENERAL PHILOSOPHY

Since June, 1964, a Control Data 3200 computer system has been installed in the Latter-day Saints Hospital in Salt Lake City, Utah, under support from NIH grant FR-00012. This system in its inception was used to develop research programs and time-sharing software for use by the medical community in the Salt Lake City area. As a result, a software and hardware system called MEDLAB has been developed. Using this system, research programs were developed for cardiovascular studies. It soon became apparent that the programs which were being developed could also be used in a clinical environment.

The first clinical application of the system was used in the heart catheterization laboratory. These programs involved pressure analysis, oxygen saturation analysis, dye dilution studies, etc. With the system in the catheterization laboratory new needs arose to satisfy the routine day-to-day clinical application of the computer. It was not now possible to make the program and hardware changes necessary in research without disrupting the 24 hour clinical service.

Without a memory protect system available on the 3200, programs being debugged could easily destroy the results and programs of another user, frustrating particularly the clinical user, who might be unaware of the problems involved in developing new programs.

As development continued new clinical programs rapidly became available, such as intensive care monitoring and patient screening programs. A new system design had to be developed. For the clinical applications to become effective in patient care situations, maximum reliability is required and can be provided only with back-up hardware.

The computer system described in this paper was developed to serve the two needs of this facility. A description of both the hardware and software and two

clinical applications presently in operation at the facility are presented.

Hardware configuration

The system used for both research and clinical applications is made up of three computers located at the L. D. S. Hospital. A block diagram of the system is shown in Figure 1 which shows the two computers—a CDC 3200 and 3300, the 3200 being used for research and program debugging while the 3300 is used strictly for operational clinical applications; the small Digital Equipment Corporation PDP-8/S computer is used as a teletype buffer driver to provide hard copy at distant hospital sites. Although the 3300 has expansion capabilities which the 3200 does not have such as paging memory, memory protect, etc., the 3300 used has essentially the same capability as the 3200 computer. Therefore, both machines are hardware and software compatible and communicate through common disc units.

There are three pieces of equipment identical on each machine which are critical for hardware and software interchangeability. These are:

- (1) The disc storage units.
- (2) The REDCOR Corporation read and write interfaces, which are the adapters for communicating with the remote terminals and the handling of the physiological signals coming either from a patient or an experimental set up. The read interfaces for both machines are identical and analog signals are presented in parallel to the analog multiplexers of both interfaces. Thus, a program can be debugged, checked out and made operational on the 3200 research system then transferred to the clinical system with no change of channels or program. If the analog-to-digital (A-to-D) converter or computer system for the

clinical system fails, the clinical operating system can be transferred to the 3200 machine with a minimum of rewiring (approximately ten minutes required for change over after the problem is diagnosed) and the assurance that the analog signals will be correct.

- (3) The write interfaces on both machines are identical and are connected such that they can be transferred from one machine to the other with a minimum of difficulty (approximately ten minutes).

The two machines are different in their hardware configurations since the 3200 is used for program compilation, printing and magnetic tape capability for program and data storage. The extra peripheral equipment on the research machine includes a 1,000 card per minute card reader, a 1,000 line per minute printer, three high-speed magnetic tapes and one special-purpose high-speed A-to-D converter. The 3300, or clinical machine, on the other hand, has a special output interface for the small PDP-8/s computer used to drive teletypes at the remote sites for hard copy reporting of clinical and experimental information. Both machines are capable of operating remote terminals both from sites within the hospital and remote sites at other hospitals, experimental laboratories and clinics.

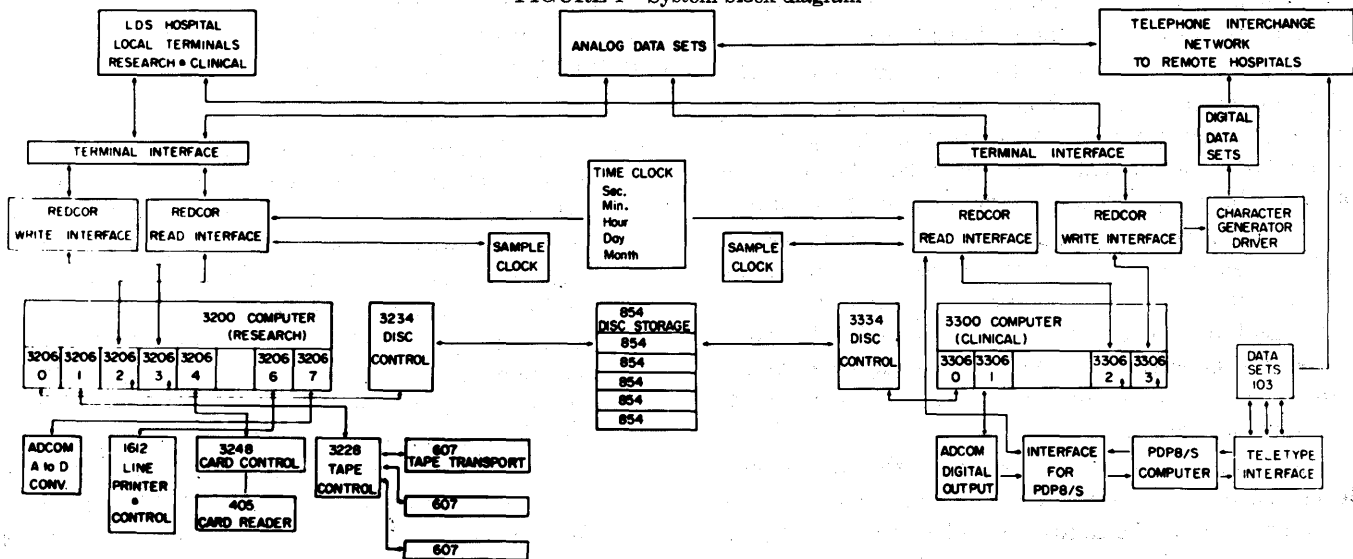
Figure 2 shows a photograph of a typical remote terminal through which an operator communicates with the computer. This remote terminal consists of a Tektronix 601 memory display unit, control and timing circuits for operation of this display unit, a decimal keyboard and two 12-bit octal thumbwheel switches for coding information into the computer. Also shown on the front panel are indicator lights which tell the operator the state of the computer, the state of his program

and various other indications.

In a typical operation, the user calls a program by dialing a code into the octal switches, then presses the CALL button which interrupts the computer. The computer reads the octal switch and displays instructions back to the operator on the face of the memory display unit. The display unit is capable of displaying 400 characters in a 25 column by 16 row pattern or graphical information with a capability of 512 horizontal and 512 vertical dots. In addition to its capacity as a remote computer display terminal, the terminal can also be used as a conventional three-channel memory oscilloscope by pressing a pushbutton switch on the front panel. This feature allows the operator to quickly check signal level qualities to be presented to the computer and insure that they are within range of the A-to-D converter and of adequate quality for the desired computer analysis. The display will revert from a conventional oscilloscope to a computer display terminal upon receiving an erase pulse from the computer; thus assuring that no computer generated information is lost while the operator is viewing waveforms.

The processing of analog signals is presently carried out independent of the display terminal. As a standard package, each laboratory or clinical area is assigned three analog channels. These three channels are used for multiple purposes. For example, the three channels could be carrying pressure information, electrocardiographic information, densitometry information, etc., depending on the requirement of the user. This three channel requirement was primarily determined by electrocardiographic analysis program where three simultaneous lead signals are necessary. A second reason for making three analog channels a standard configuration is that three channel data sets for telecommunication

FIGURE 1—System block diagram



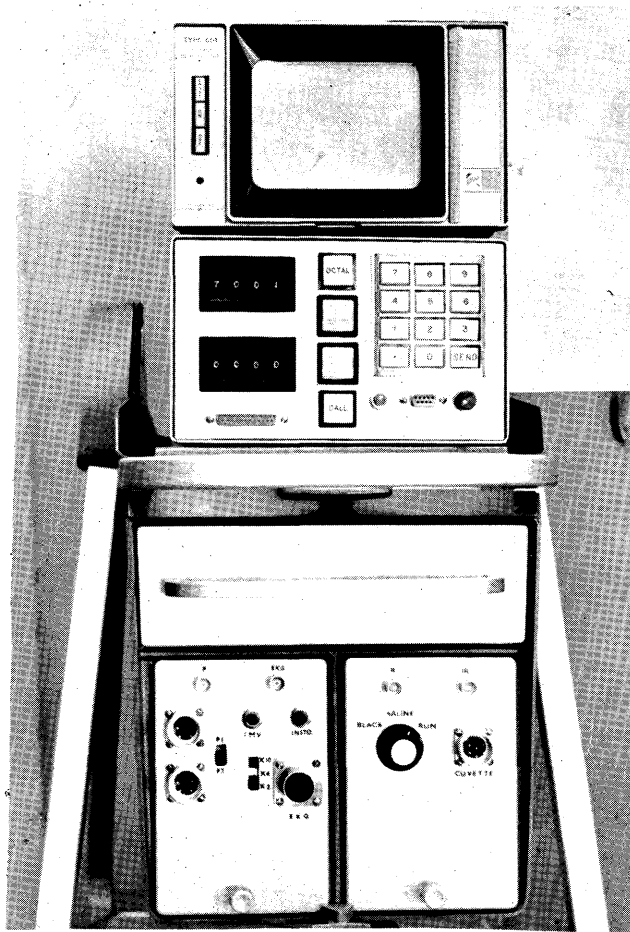


FIGURE 2—Remote terminal with instrumentation

to distant sites are available for processing of data from remote hospitals.

The remote terminals with instrumentation can be constructed for a cost of about \$3,000; a character generator added for operation of remote sites costs an additional \$1,200. With the capability of both alphanumeric and graphical functions, this terminal becomes an extremely flexible convenient module for use in both clinical and experimental applications.

Since most of the physiologic signals are analog, it was necessary to develop extensive front-end signal conditioning equipment for the computer operation. An objective in the development of the front end equipment was to provide extremely stable, highly reliable instrumentation such that a person with a minimum of instruction and training could use it. The lower part of Figure 2 shows the front panel of a typical instrumentation package. Note that there are no control knobs for adjusting gain or bias of the analog signals from the transducers and that there are a minimum number of control switches for operator use.

The analog front-end system is made up primarily

of integrated circuit operational amplifiers with all signals being amplified from their low level condition to a high level (± 10 volt) condition for transmission to the computer, either over hardware connection or a telecommunication link. In each case, signals are conditioned for optimum use by the computer and by the telecommunication link by amplifying and adjusting the offset for full scale capability of computer and communication link.

Experience has shown that minimizing the number of controls and adjustments makes the system easier to use, both by experienced and nonexperienced operators, and also increases the confidence of the operating personnel. As a typical example of an instrumentation application where this type approach has been used, consider the pressure transducer amplifier which amplifies signals from a balanced Wheatstone bridge strain gage. The gage itself can be balanced, the amplifier or amplifiers could each have a separate off-set control, the gage excitation could be varied, the amplifier gain could be varied, and so on. To minimize the problems in set up of a pressure transducer, only one control is provided and is made an integral part of the transducer system. The excitation voltage on the gage is fixed, the sensitivity of all the strain gages have been calibrated to a standard level and all pressure amplifiers are set up with the standard gains. A fixed off-set has been programmed into the amplifying system and the only adjustment that need be made by the user is gage balance which compensates for varying fluid levels of the patient. Therefore, a pressure system which is usually complicated and difficult to handle becomes a simple set up procedure which a nurse or inexperienced technician can adequately handle and get results that are technically adequate and, in fact, as good as an experienced operator can obtain.

As can be seen from the foregoing discussion, the computer system has been designed with both a research investigator and a clinical investigator in mind with standard packages designed and constructed which aid both. The system is also easily adaptable to special purpose experimentation with signal levels that can be conditioned with a great amount of flexibility for the occasional user who has special requirements for signal levels, sampling rates and timing.

Operation of terminals from remote hospitals is made over voice grade direct distance dial compatible telephone communication link. As far as the user is concerned, operation from a remote hospital is essentially the same as operating a local terminal. Figure 3 shows a block diagram of the communication link where the data sets shown are Bell system designations. As can be seen from this diagram, there are three types

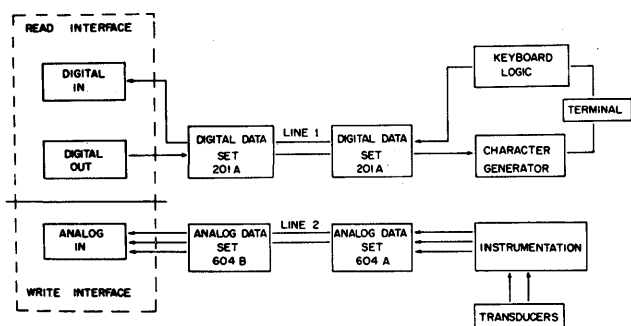


FIGURE 3—Block diagram of communication link used for remote hospitals

of data the computer system must handle for the remote terminal: (1) digital input and control information; (2) digital output used to drive a character generator and provide control outputs; and (3) analog input signals presently sent over a second telephone line.

Typical operation of the scheme might be as follows. Initially 201 data sets which connect the central processor and the remote hospital are both in the receive mode. Upon pressing a button at the remote terminal, the 201 data set at the terminal end goes into the transmit mode, after a settling time required by the data set, it transmits a serial 14-bit code to the receiver at the computer site. The data set at the computer site receives the serial data which is converted to a parallel 12-bit code and read into the computer. For the usual operation a keyboard entry is followed by an alphanumeric reply to the terminal. The data set at the computer end becomes a transmitter and transmits data back down the line to the remote console. Using a character generator scheme (developed jointly by the authors and Beehive Electrotech of Salt Lake City) alphanumeric and graphical data are presented on the remote storage display unit. The present transmission scheme uses a 14-bit word consisting of one sync bit, a parity bit and 12 data bits. The 12 data bits are broken down into nine bits of display data with three control bits which determine whether the character generator writes alphanumeric characters, using the ASCII code, plots graphical information, sets up control functions or outputs nine bits in parallel to remote control devices.

With the rate of 2,000 bits per second available with the Bell system's 201 data set, the character generator scheme will output characters at approximately 130 characters per second and plot graphs at approximately 130 points per second, being limited by the data set bit rate. With faster data sets correspondingly faster write-out rates could be obtained. After a transmission is completed both data sets return to the receive mode and are free for the operator to again enter some type of digital information or a piece of control equipment

outboard from the remote terminal to access the computer.

The basic remote terminal, which is not much larger than the 201 data set (Figure 2), is completely transportable and can be taken anywhere telephone communication facilities are available and used to communicate with the MEDLAB system. Applications for stand-alone communication terminals of this type are expected in tumor registry programs, radiation treatment planning and other areas where analog signals are not necessary. When analog signals are required, Bell system data sets 604A and 604B, which are analog transmitter, receiver respectively, are used to send three channels of analog information simultaneously. The analog data bandwidth of these FM multiplexed channels is DC to 100 cycles for each channel. Cross-talk and signal-to-noise characteristics and bandwidth of these channels is adequate for transmission of most clinical physiologic information. The requirements for three channels is dictated by the vectorcardiographic system which requires three simultaneous channels of ECG. With a slight modification of this system it is possible to use touch-tone telephone keyboard and a 604A=B, 401J data set configuration to transmit electrocardiograms and other physiological data from any patient room within a hospital by installing phone jacks in the rooms and using the internal television distribution system of the hospital to transmit instructions and results to the operator on the television receiver located in the patient room.

A "program-line" connection is operational between the neurophysiology research laboratory at a remote hospital. This type line is one which is commonly used by FM music stations and is conditioned to have "flat" frequency response from 50Hz to 8KHz making it ideally suited for transmission of action potentials.

Presently there are systems in four hospitals using the communications terminals and one additional hospital using a hardwire connection. Plans call for region-wide screening clinics to be conducted by State, local and private health organizations. With these terminals, health services can be provided to remote communities that heretofore were available only to patients at major hospitals.

Software

The software available on the 3200 (research) system and that on the 3300 (clinical) system are quite similar but some important differences exist. A major difference is the type of program which can run under either system. On the 3300 system, which is used for the clinical applications, there are 12 partitions within core memory, each partition being approximately 2,000 words in length. Only those programs which are designated

clinical real-time programs and have been written in assembly language are allowed to run within any of the 12 partitions. Since the user can have only 2,000 words of core at any time most programs are written as a series of overlays to be read in as needed into the same partition. These programs must have reached a high degree of reliability before being allowed to run on the clinical system. The clinical executive monitor contains a dictionary of the programs allowed and if a program is not in that dictionary, a message is written on the display unit at the terminal indicating that the program is not allowed. No debugging of programs is allowed on the clinical system. All debugging must be performed on the research system.

Within the 3200 (research system) there are only four partitions for real-time programs and a 16,000 word area of core set aside for background programs. These background programs consist mainly of compilations, statistical analysis programs, report generation programs and other nonreal-time programs. There is also available on the research system a program designated DEBUG which allows for on-line debugging of real-time programs from any remote terminals. Upon calling this program from a research terminal, the programmer may then use it to aid in debugging his program. This program allows the user, within his program, to execute instructions, change instructions, look at data in memory, etc., in an on-line situation.

Software-wise there is no interaction between the research system and the clinical system. The research system is unaware of whether the data generated on any disc is generated by a program being run on the research or the clinical system. All data which are generated by either machine and stored on disc are accessible by the research system for report generation on the line printer.

The time-sharing of the programs on either machine is accomplished by a series of tables which are stored in the executive monitor. These tables correspond to either external station interrupts or internal clock interrupts. As an example, assume that a user desires to sample a pressure waveform at the rate of 100 samples per second. Within his program he branches to a subroutine in the executive monitor with the rate he desires to sample and the location within his program where control is to be given at the time of his clock interrupt. This subroutine then searches through the tables to determine the minimum time for the next interrupt and sets the appropriate hardware interrupt registers to generate an interrupt at that particular time. On the occurrence of the clock interrupt, the system determines which station caused the interrupt and branches to that program via the clock interrupt

JUMP table which had been previously set by the program.

Similar action takes place with operator interaction from a remote terminal. Within his program the user will use the routines and tables within the monitor to set the addresses he wishes to branch to when he presses one of the interrupt buttons at his remote terminal. When an external interrupt is generated in this way, the system decodes it to determine which station has interrupted and branches to the appropriate table. If the code read from the terminal is a program call code, the system will load the user's program from disc and branch to the beginning of his program. Interaction continues with the program writing instructions on the storage display unit and waiting for a reply from the operator.

With the present hardware configuration on the clinical system there are 12 terminals connected to the 3300 and, thus, with the 12 partitions available within core there is no swapping of programs required. In the research system, however, this is not true since there are only four partitions available and 12 remote stations connected to the computer system. Algorithms have been developed to determine whether a station is inactive; that is, it is either not sampling any real-time data, it is not accessing the disc or performing some other I/O function. When these conditions exist, this program may then be transferred to a disc and another user's program brought into core at that location. When the user interrupts to initiate some action from the system, the monitor determines if his program is in core. If not, it checks to see if that partition of core is busy and then loads the program from disc or writes a message on the scope indicating that core is being used at that time.

Within the 3200 system there is available the larger area (16,000 words) in lower core for background programs. This program may, however, be linked to a real-time program in upper core. This is usually done when the program has need to sample analog data from some station. Fortran programs, which have been linked to one of the assembly language programs, are given the priority one whereas compilations and Fortran programs run through the card reader in a batch-processing manner are given a lower priority, priority two, and will be swapped out when request from a terminal is made for that area by a priority one Fortran program. Many of the present assembly language programs are developed using this capability; that is, writing the program, or at least portions of it initially in Fortran to get the basic logic developed. Then once the program has been debugged, it is converted to assembly language for use on the clinical machine.

Resident in core within both systems are a series of

re-entrant subroutines which may be accessed by any user's program. These subroutines are used to write messages on the display terminals, to store or retrieve data on the disc, to convert from binary to floating point or BCD, etc. Resident also are special clinical routines which are used by most of the users.

Since there is no memory protect on either machine, some software features have been developed, especially for the clinical system, to attempt to eliminate errors which might result in the running of any program. Since the basic cause of errors in running of the clinical programs at this facility results from generating external interrupts from the terminal at inappropriate times, measures have been placed within the executive monitor system to allow for external interrupts from a remote terminal only during those times when they may be serviced by the program without causing interference with other activities going on within that same program. Some of these times are specifically when the program is sampling data, using the disc or writing on a scope, etc. This does not, however, exclude other remote terminals from generating external interrupts and performing functions within their program but only excludes the user from interrupting himself at these times.

Applications

Patient screening

One of the newer projects using the system at this facility is a patient screening admission program. Every patient who is admitted to the L. D. S. Hospital, with the exception of maternity and emergency patients, are screened using this program. When the patient arrives at the hospital and registers in the Admitting Office he is given a hospital record number. This number is used by the computer system to generate a file of data for the patient. Once the patient has received his registration forms he is brought to an admitting laboratory where two samples of blood and a urine sample are taken for analysis in the Chemistry Laboratory. Upon leaving the admitting laboratory the patient is brought to the computer screening laboratory. A file is initiated on the patient by entering the patient's hospital number on the decimal keyboard at the terminal. A nurse measures the patient's blood pressure, temperature, respiration rate and enters these parameters along with age, height and weight in the patient's file.

Two on-line computer tests are then performed on the patient. The first is a maximum breathing test where the patient is required to take a deep breath and blow into a spirometer which measures both the total volume expired by the patient (Forced Vital Capacity), the

volume expired after one second and two flow rates during the maximum expiration. The analog signal generated by a potentiometer connected to the spirometer is sent directly to the computer. Corrections for temperature, barometric pressure and calibration factors are made by the computer and the results presented on the display unit within two seconds after the test. Once the patient has successfully performed this test, which usually requires blowing into the spirometer at least twice in order to obtain the best possible results, the patient is given a computerized electrocardiogram (ECG) with the computer sampling the output of the three vector signals from the ECG amplifier. This test requires a series of eight electrocardiographic leads to be connected to the patient. These leads are resolved by the amplifier into an orthogonal lead system used for the measuring of the electrical activity of the heart. The program performs a pattern recognition on the data collected and reports back to the screening technicians a classification of an ECG pattern. This information is also stored on the patient's file. Once the patient has completed his electrocardiogram he is taken to his room. Total time for these two tests is approximately five minutes with the computer being used for about one minute.

Other information entered into the patient's file includes the results of the urine analysis and the hematology analysis, and the blood chemistry tests run on a 12-channel autoanalyzer. The 12-channel autoanalyzer is operated as an on-line terminal which allows the computer to sample its output and store the results directly into the patient's file. The urinalysis, as well as the hematology results, are entered into the patient's file through the keyboard at a remote terminal.

At the end of the day the technicians generate a report from the patient's data by punching a card with the patient's name and hospital number. The report generated for each patient contains all the data which had been entered, either automatically by the computer or keyed in from one of the remote terminals. The program prints out the test results as well as a problem list; that is, a listing of all values which are outside of normal limits. The reports are then distributed to the nurses' stations and placed on the patient's charts. Subsequent data gathered on the patient during his stay in the hospital are also recorded on the patient's file by the computer. At time of discharge the file is taken from the active file, which is stored on one of the magnetic discs, and transferred to magnetic tape in the inactive file. At this time, or shortly after, a discharge diagnosis is placed on the patient's record. When the patient is readmitted to the hospital his record is retrieved and pertinent information returned to the disc in the active file. This screening procedure has been

performed on several hundred patients and although results are preliminary, significant numbers of patient abnormalities have been observed.

Intensive care ward

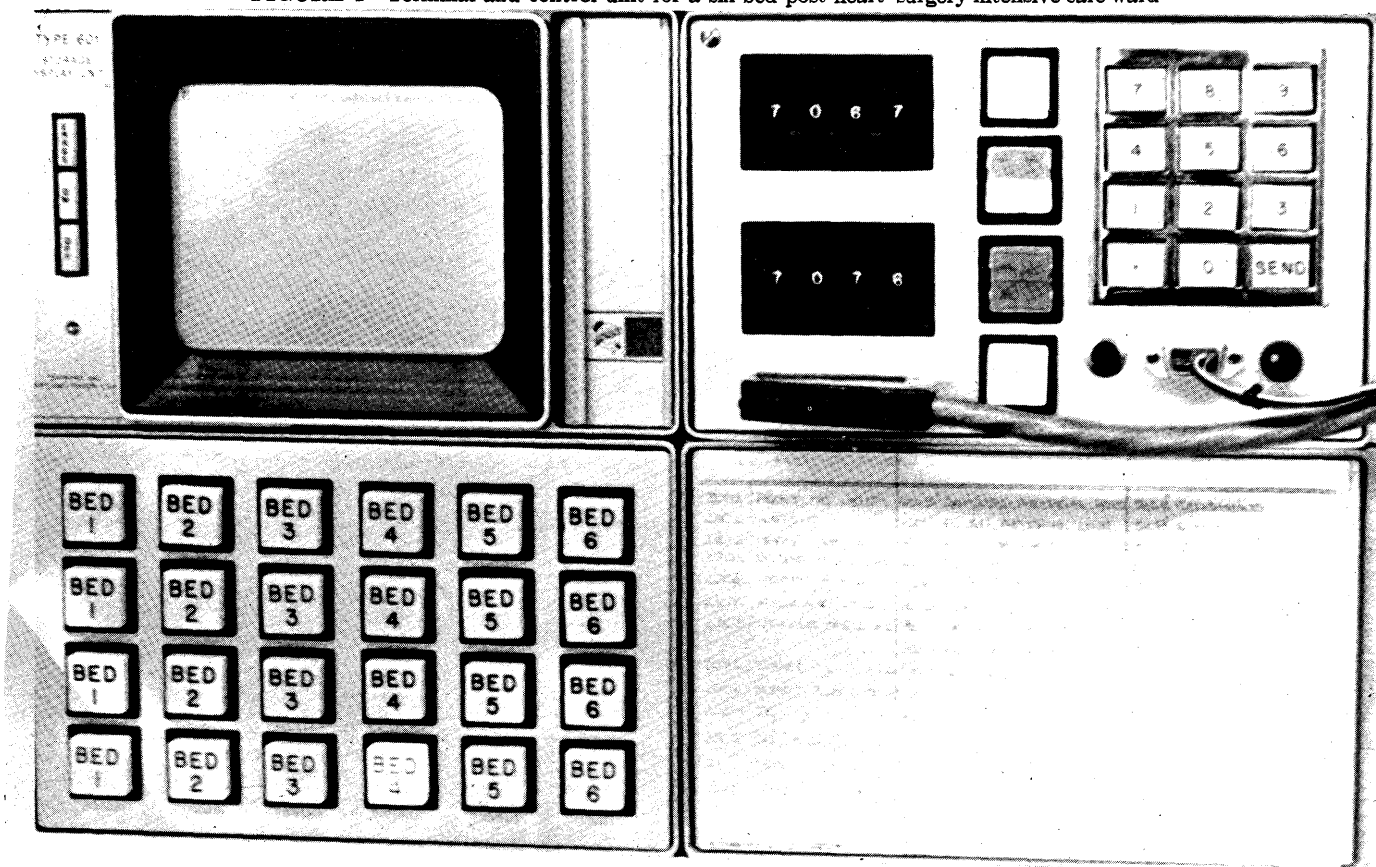
A six-bed intensive care ward for monitoring of patients who have had open-heart surgery has been in operation at the L. D. S. Hospital since March, 1966. The system is currently used routinely by the surgeons and nurses for monitoring cardiovascular function and entering nurses' notes for each patient on the intensive care ward. A picture of the central console used for control of all six beds is shown in Figure 4 and consists of a terminal similar to those discussed earlier as well as an input/output control box used to control switching of signals from each of the six beds in the ward.

The column of four lights for each bed in the intensive care ward are colored red, yellow, green and white and are controlled by the computer. A green light is turned on when the computer is actively sampling data from the respective bed. The red light indicates a change in patient status while the yellow light is an indicator to the nurse that some procedure is required on the patient, such as scheduled drug injections. The bottom row of lights are used to display the analog inputs from each bed on the memory display unit.

Pressing BED 1 on this row for example would cause the display unit to go into the sweep mode and simultaneously display arterial pressure waveform, electrocardiogram and venous pressure waveform. The display unit will erase at the end of each sweep and revert to a computer display anytime the computer begins to write out information.

Two modes of data collection are available. A "measure-once" option allows the nurse to call the program, indicate the bed number and initiate measurements on the patient. Sixteen heart beats are analyzed and the average value for each of ten variables (heart rate, stroke volume, cardiac output, peripheral vascular resistance, duration of contraction, maximum pressure, minimum pressure, mean venous pressure, respiratory rate and amplitude) is immediately displayed on the memory display unit and stored on magnetic disc by the computer, along with the time and date of the measurement. The second mode involves setting up a schedule of measurements. When the nurse chooses this option, measurements are made on 50 successive heartbeats and for each variable the mean and standard error of the mean are determined. This statistical description of the state of the patient forms a base from which subsequent measurements are evaluated thus eliminating the need for the nurse or doctor to decide

FIGURE 4—Terminal and control unit for a six-bed post-heart surgery intensive care ward



arbitrarily on upper or lower limits for each variable to initiate an alarm condition. Four minutes after the base line measurements are made on a patient, a scheduled measurement is initiated automatically and consists of determining a mean value for each of the ten variables over 16 heart beats. A red light is turned on if any one of the mean values exceeds its base value by more than three standard errors of the mean. To determine which variable has deviated from its base line, the nurse, at her earliest convenience, merely presses the red switch light which interrupts the computer and sends a code identifying the bed number and causes the computer to display a message indicating the variable furthest out of tolerance. The value of the last reading and the time of the reading are shown together with the mean, standard error of the mean and time of the base line measurement for comparison. At this point the nurse may choose one of several options; she may explain or verify the alarm indication, determine a new base line, or review the wave-forms. If the change in status detected by the computer represents the establishment of a new steady-state, the nurse establishes a new base line at this point. All subsequent measurements will then be referred to this new base.

The interval between scheduled measurements is under computer control and is dependent on the state of the patient. If the first measurement made four minutes after a base line is not statistically different from the base line, the next scheduled measurement will occur in eight minutes. If the red light is turned on, however, the next measurement is made in two minutes. Thus, the interval between measurements will vary between two minutes, when the patient is unstable, up to 16 minutes when successive readings coincide consistently with the base values.

Since the computer system is not continually sampling all six beds but is merely scanning from one bed to another depending on patient status, there is a possibility that a patient could get into trouble, say within the 16 minute interval. Two of the more common problems encountered are related to frequency of contraction of the heart. These are ventricular fibrillation (speeding up of the heart) and cardiac arrest (heart stopped). Since this could happen at any time and detection from the arterial pressure wave-form would require continuous monitoring, the electrocardiogram or arterial pressure signal are used to drive a rate meter.

Upper and lower heart rate limits are set for each patient. If the patient's heart rate goes beyond a limit, the red warning lamp for that bed begins to flash on and off and an interrupt and bed code are sent to the computer causing it to record the event and start making measurements of cardiovascular function. Remote hospitals also have similar intensive care wards with complex control features made possible by the communications system described above.

There are a variety of physiologic variables which would undoubtedly contribute additional useful information to patient care. Some of these, such as blood chemistry determinations, blood gas analysis, are measured periodically on these patients but at present the information is entered into the computer only semi-automatically from other laboratories in the hospital. The intensive care service is provided on a seven day a week, 24 hour a day basis and requires a maximum of computer reliability and up-time. To achieve the confidence of the doctors and nurses involved in patient's care, the system was designed with complete back-up of all the critical signal measuring and reduction systems. There is a possibility that a card reader or printer could fail, eliminating hard copy, but the information is still available to the user through his remote terminal. The hard copy could be printed later after repair time of the printer or card reader.

Results of the use of the computer system in the intensive care ward have been very encouraging. At present patients are automatically put on the monitoring system unless the attending physician requests that they not be monitored. The surgeons schedule their surgery around the computer system (i.e., if the computer system is not operational, surgery is rescheduled for a time when the computer service is available).

CONCLUSIONS

The dual system just described has been in operation since September, 1967, and presently services 20 terminals. These terminals are located in six separate medical areas at distances from a few miles to 50 miles from the central facility. The system is used in diverse areas as heart catheterization, nervous system studies, screening clinics and others. Experience to date has shown that the system performs both clinical and research functions well.

The use of computers to improve biomedical image quality

by ROBERT H. SELZER

California Institute of Technology
Pasadena, California

INTRODUCTION

Pictorial data have long been an important source of information in medicine and biology, but until quite recently, little use has been made of computers to process this data. Perhaps the major obstacle to such development was the difficulty of getting pictures into and out of the computer. The character recognition systems developed in the fifty's were in general not applicable to biomedical image problems because of the restrictive nature of the input devices. High resolution image scanners with the ability to detect multiple shades of grey were required and while such devices became available in the early sixties, it was difficult to find biomedical applications that would justify their cost which was frequently in the half million dollar range.

In the past two or three years, the cost of scanners has gone down and the need for image processing has gone up. As a result, the prospects are quite good that computers will be used extensively within the next two to five years both for image enhancement and for pictorial data-extraction. Some of the current and potential areas of application are described below.

When used to enhance existing pictures, the computer is potentially capable of producing an absolute improvement in image quality because processing methods are available that can retrieve information partially lost in the image system that generated the pictures. These techniques are applicable to X-ray films, photomicrographs or to virtually any type of photographic data.

The computer can also remove useless data or emphasize selected classes of features. Processing of this type might be used for mass screening of medical X-ray

film in the hope of reducing viewing time. Considering the fact that over 200 million X-ray films were made in the United States last year, it is clear that computer-aided analysis of radiographs is a useful objective.

A major factor in the current interest in computer image processing is the recent emergence of biomedical techniques that require objects on film to be counted or measured. Computer methods have already been applied quite extensively to the problem of counting and sorting human chromosomes (karyotyping). As an example, one film scanning system karyotypes a cell in 20 seconds—a job that requires approximately 15 minutes when accomplished manually by cutting and sorting the chromosomes from a print.¹

Work is also in progress in automatic white blood cell classification,² in autoradiographic grain counting and in numerous other areas.

Over the past three years, computer image enhancement research at the Jet Propulsion Laboratory has been applied on an experimental basis to biomedical pictures in general and particularly to medical radiographs.³ The initial objective was simply to see if the computing techniques developed to process spacecraft television pictures,⁴ could be extended to biomedical imagery in a useful way. Rather than immediately attempting to apply the computer to a specific clinical or research problem, we have taken the more general approach of developing processing techniques applicable to classes of pictures such as pictures with high noise levels, pictures of low contrast, pictures having poor resolution, etc.

The main purpose of this paper is to describe some of the enhancement methods that have developed. As outlined in the next section, the frequency-domain approach to image system analysis was chosen as the most suitable means to relate the computer methods to the physical imaging system* and to the subject. Particular attention is paid to the interpretation of the sys-

*In this paper, the phrase "imaging system" means all parts of the system between the subject and the image digitizer.

tem transfer function relative to the subject content. Following this, the most important computer enhancement technique, digital filtering is discussed and illustrated with numerical examples. Details of filter evaluation and synthesis are covered in the appendix. Following the section on filters, examples of enhanced pictures which portray the effects of several types of filters, both linear and non-linear, are discussed. In addition, some other commonly used enhancement methods including non-linear contrast enhancement and image subtraction are illustrated. Finally, an example of quantitative pictorial data analysis which utilizes the preceding enhancement techniques is described.

Frequency-response methods applied to imaging systems

The general objective of picture enhancement is to make selected features easier to see. This might require suppression of useless data such as random noise and background shading or perhaps amplification of fine detail. Background shading becomes a problem when it is super-imposed onto low contrast features. Usually, the subject itself is the source of the problem. For example, the small bones in the ear cannot generally be seen in a standard X-ray film because they absorb too little radiation relative to the larger surrounding bone mass. Random noise in an X-ray film results from the spatial fluctuations of the illuminating radiation. Similarly, film scanning systems inject noise into the image because of fluctuations in the light source and because of electrical noise in the output of the light sensing device. Fine detail is lost by diffraction effects in optical systems or by fluorescent intensifying screens in X-ray systems. Clearly the imaging system as well as the subject must be taken into account when selecting a computer enhancement procedure and to accomplish this, a common method for describing each part of the system must be used. In many cases, frequency response techniques based on the Fourier Transform can provide a common basis for relating the computer methods to the subject and to the imaging system.

Using this technique, the subject, or more accurately, the signal composed of the spatial distribution of photons that results from illuminating the subject, is described by its Fourier or frequency-domain transform. That is, it is well known that any absolutely integrable signal can be decomposed into pure sinusoids of varying frequency, amplitude and phase and the two representations, spatial and frequency, are precisely equivalent in the sense that each can theoretically be obtained from the other. For a picture system, frequency is measured as cycles per millimeter across the image.

Using frequency response methods, the imaging system is described by its optical transfer function (OTF) (also called more simply the transfer function) which is

defined as the relative amplitude and phase response of the system to sinusoidal inputs of varying frequency. The amplitude response is commonly called the modulation transfer function (MTF). In the absence of phase shift, which is the case with most parts of an imaging system, the OTF and the MTF are the same. If the imaging system is linear, which in this case means that the system output contains exactly the same frequencies as the input, then the signal spectrum of the output is equal to the product of the system transfer function and the input signal spectrum.†

Under suitable conditions,* the image digitizing process and many of the computer enhancement operations can be considered as linear extensions of the imaging system and transfer functions for these parts can be derived. This, in turn means that the entire system, from subject to computer enhancement can be related by frequency-response methods.

Before discussing the actual computer enhancement operations, it is useful to consider the effects of the imaging system on typical subjects in terms of the system frequency response.

For example, suppose two imaging systems, A and B with MTF as shown in Figure 1 are to be compared. Since B produces equal or less attenuation than A at all frequencies, B will clearly reproduce the input scene more accurately. In particular, B will reproduce fine

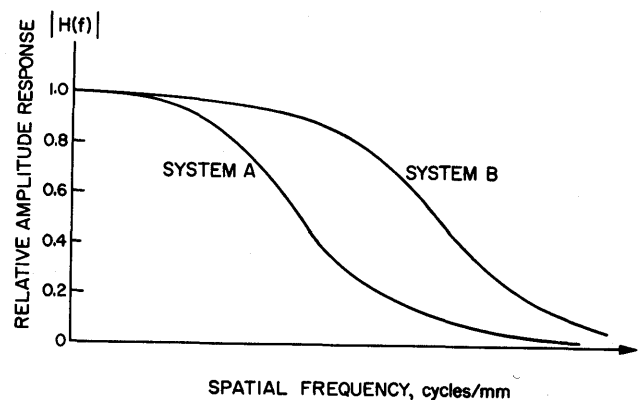


FIGURE 1—Typical modulation transfer functions for two imaging systems.

†Certain parts of the system, such as the film are not linear, but a linearizing correction can often be made when the image is processed with the computer. For film, the correction can be made if the optical density vs relative log exposure curve is known.

*The image must be sampled at a rate at least twice the highest frequency present. When this is the case, the sampled signal spectrum is a periodic version of the continuous signal spectrum and under most circumstances the periodic components can safely be ignored.

detail better than A since small features with sharp edges will usually result in an input signal whose spectrum has a greater proportional of its energy at high frequencies when compared to the signal from large smooth features. It does not follow, however, that B would be selected as the better system in every case. For example, if a low-contrast shadow in an X-ray film were being sought and background noise was high, a system like A that tends to reject high frequencies might produce a more suitable film.

The behavior of the MTF at low frequencies is sometimes as important as at high frequencies. To illustrate this, consider the problem of locating a suspected hairline fracture in an X-ray film as shown in the drawing of a bone in Figure 2a. A plot of film density across the bone on the line shown might appear as in Figure 2b. The fracture appears as a low amplitude narrow (i.e., high-frequency) density change superimposed on the large slowly changing low-frequency signal representing the bone shadow. As shown, the fracture shadow probably would not be visible on the film. This example demonstrates that sometimes it is desirable to reproduce high frequencies but *not* low frequencies. If a system were selected that has an MTF as shown in Figure 2c, the low frequencies would be rejected and the plot line through the film would look as shown by Figure 2d. At this point, an increase in contrast would bring out the fracture shadow.

As another example, consider the pictures generated by isotope scanners which frequently contain narrow bars of unexposed film between each recorded scan line. If the unwanted bars were spaced every T_1 millimeters down the image, an imaging system that rejected the bars would have a two-dimensional MTF like that shown in Figure 3. Designing an actual system with such a transfer function would be quite difficult in most cases, but a digital filter with zero gain near frequency $f_1 = 1/T_1$ and near unity gain at all other frequencies of interest could easily be obtained. Application of this filter to a picture from an isotope scanner with a normal MTF would produce an effective overall system MTF like that shown in Figure 3. Techniques for obtaining and applying filters of this sort are described in the next section.

Digital filters

Many, although not all computer image enhancement techniques are linear operations that can be analyzed by frequency response methods. Once a transfer function for a particular computer operation is determined, a new system MTF which includes the physical parts of the imaging system, the film digitizer and the computer operation can be determined.

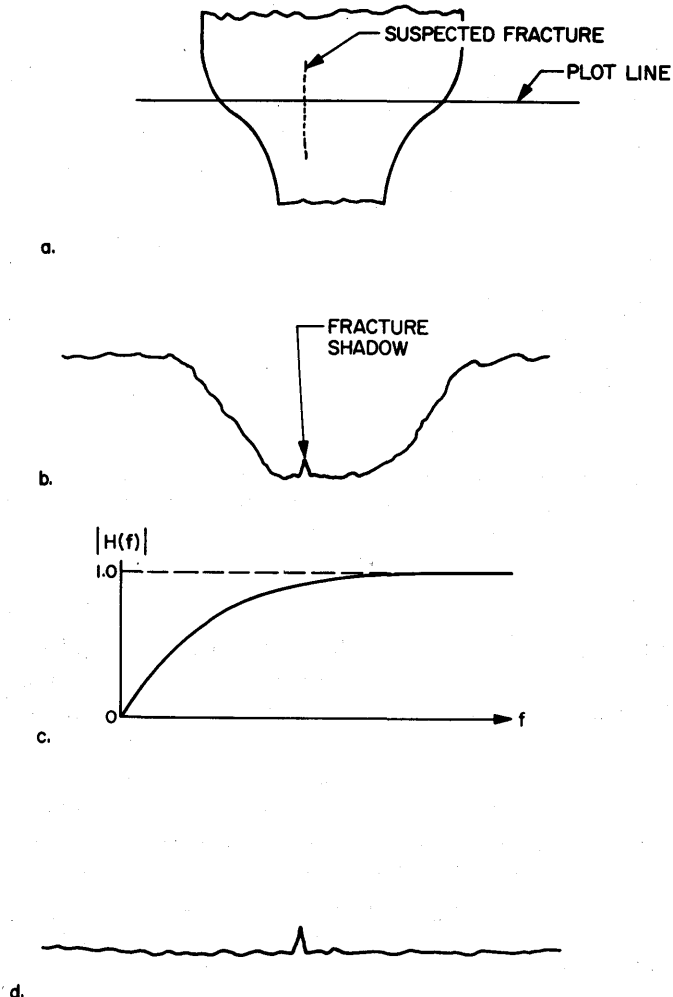


FIGURE 2—An illustration of the processing steps involved in removing background shading in order to enhance small low-contrast feature.

- (a) Representation of an X-ray image of a bone containing a suspected hairline fracture shown as a dotted line;
- (b) Typical density plot that might be observed along a single line across the film;
- (c) The MTF of a filter that could be used to remove the background;
- (d) Density plot after filtering, showing background removal.

One of the most important linear image enhancement techniques, filtering, is covered in this section. The basic definition of a digital filter in the spatial domain is given and elementary examples are used to classify filters into some general categories. Methods for calculating the modulation transfer function of a given filter and for obtaining a filter with a specified MTF are described in the appendix.

Definition of a digital filter

An electronic filter produces an output voltage that

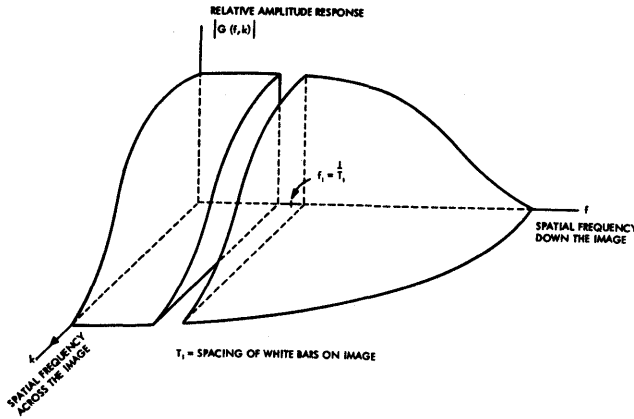


FIGURE 3—Modulation transfer function designed to reject vertical bars which typically appear on films generated by isotope scanners. The bars are assumed to appear every T_1 millimeters.

depends on weighting of the past signal. A digital filter operates in the same manner except both past and following signal is available and, of course, the signal is discrete.

Let (X_0, X_1, \dots, X_n) be a sequence of numbers derived by sampling and digitizing a continuous signal such as the optical density along a line on a film, and assume samples are taken every T millimeters along the line. Consider a simple three point filter that replaces the point x_n with the average of the points x_{n-1}, x_n, x_{n+1} . If y_n is the filtered point at position n , then

$$y_n = (1/3)(x_{n-1} + x_n + x_{n+1})$$

$$= 1/3 x_{n-1} + 1/3 x_n + 1/3 x_{n+1}$$

Similarly, the next filtered point y_{n+1} is obtained as

$$Y_{n+1} = 1/3 x_n + 1/3 x_{n+1} + 1/3 x_{n+2}$$

The values $1/3$ are the filter weights, which in general are not necessarily equal. The main filtering relation using $2K + 1$ weights

$$g = (g_{-K} \dots g_{-1}, g_0, g_1, \dots, g_K) \text{ is}$$

$$y_n = \sum_{k=-K}^K g_k x_{n-k} \tag{1}$$

Thus filtering is obtained by a weighted moving average.

Equation (1), the filter equation, is readily extended to two dimensions. Let the unfiltered signal be represented by an array of numbers $(x_{m,n}, m = 0, 1,$

$\dots, M, n = 0, 1, \dots, N)$ and the filter weights by the array

$$g = g_k, \ell, k = 0, \pm 1, \pm K; \ell = 0, \pm 1, \dots, \pm L).$$

Then if $y_{m,n}$ is a filtered point, the two-dimensional filter equation becomes

$$y_{m,n} = \sum_{k=-K}^K \sum_{\ell=-L}^L g_{k,\ell} x_{m-k, n-\ell}.$$

A two-dimensional averaging filter analogous to the 3 point one-dimensional averaging filter just discussed is one that averages the nearest nine points including the nearest three points on each line above and below the center point $x_{m,n}$ as well as $x_{m,n}$ and the points on either side of the same line. Most of the ideas that follow are easily extended from one to two dimensions so in the interest of keeping the notation simpler, only the one-dimensional filter will be presented in most cases.

Filter classes

To further illustrate the filtering operation, several classes of filters are defined and examples of each class are given.

The three point equal weight filter previously discussed is an example of a *low-pass filter* whose purpose, as the name implies, is to pass low frequency signal components and reject the high ones. A typical modulation transfer function of a low-pass filter is shown in Figure 4a.

A *high-pass filter* has the opposite function to remove low frequency signals and pass the high frequency signals. One obvious way to achieve such a filter is to subtract, point-by-point, a low-passed picture from the original. Rather than actually perform such a two step operation it is possible to derive weights that directly high-pass the picture. Suppose x_n is the unfiltered input, y_n the output from a low-pass filter and y_n' the output from a high-pass filter. Let $g = (g_{-K}, \dots, g_K)$ be the low-pass weights and $g' = (g'_{-K}, \dots, g'_K)$ be the high-pass weights. If the low-pass picture is subtracted from the unfiltered picture, point-by-point,

$$y_n' = x_n - y_n,$$

and then expressions for y_n' and y_n according to the filter equation (1) are substituted above

$$\sum_{k=-K}^K g_k x_{n-k} = x_n - \sum_{k=-K}^K g_k x_{n-k}$$

Equating coefficients of x_{n-k} on either side of the

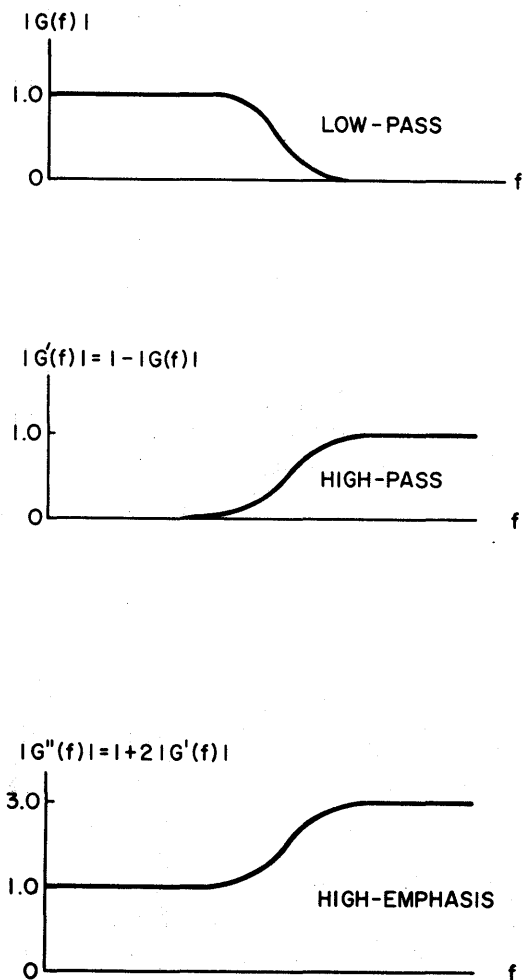


FIGURE 4—Typical filter transfer functions:

- (a) Low-pass;
 - (b) High-pass;
 - (c) High-frequency restoration;
- Filters (b) and (c) are derived from filter (a).

equality, it follows that

$$g'_k = \delta(k) - g_k$$

where

$$\begin{aligned} \delta(k) &= 1 \text{ for } k = 0 \\ &= 0 \text{ otherwise} \end{aligned}$$

As shown in the appendix, if the low-pass filter has MTF $G(f)$ as shown in Figure 4a, then the high-pass filter derived above has MTF $G'(f) = 1 - G(f)$ as shown in Figure 4b.

Consider another type of filter that might be called a high-emphasis or high-frequency restoration filter. This filter passes low-frequency signal unchanged and am-

plifies high-frequency signal. Pre-emphasis networks in hi-fi systems are filters of this type. Applied to a picture this type of filter sharpens edges and generally magnifies small detail. A high-pass filter and a high-emphasis filter are similar except the high-pass filter removes the low frequencies while the high-emphasis does not. In fact, a high-emphasis filter can be derived from a high pass filter by multiplying all the high-pass weights g'_k by a constant and then adding one to the center weight. That is, if g''_k is a high-emphasis filter weight, and A is a constant,

$$g''_k = \delta(k) + Ag'_k.$$

Multiplication of every weight by a constant is equivalent to multiplying every value of the MTF by the same constant. It also can be interpreted as stretching the contrast of the picture by this constant. Also, adding one to the central weight adds one to the MTF at every point. Thus, for $A = 2$, the high emphasis filter derived from the high-pass filter of Figure 4b would appear as shown in Figure 4c.

Instead of smoothing by simple averaging, higher degrees of polynomial smoothing may be applied. For example, suppose a smoothed point is obtained by evaluating a third degree least square polynomial fit to the nearest five points. It can be shown that such an operation is equivalent to applying the filtering equation, (1), using the following weights:

$$g = (-3/35, 12/35, 17/35, 12/35, -3/35).$$

In order to determine how this filter differs in its effect on the signal from the equal weight filter or from any other filter, the modulation transfer function can be computed.

As described in the appendix, the transfer-function $G(f)$ of a filter represented by weights $g = (g_{-K}, \dots, g_0, \dots, g_K)$ is obtained as the Fourier Transform of the weights. The three point equal weight filter, for example, defined by

$$g = (1/3, 1/3, 1/3),$$

has a transfer function obtained as

$$G(f) = 1/3 + 1/3 \cos 2\pi f,$$

where a sampling period of one is assumed, so f represents cycles per sample.

A plot of $|G(f)|$ is shown in Figure 5a. The dotted lines indicate negative values of $G(f)$ which represents a 180° phase shift. That is, if the input to the filter is a sine wave whose frequency is in this range, the output would

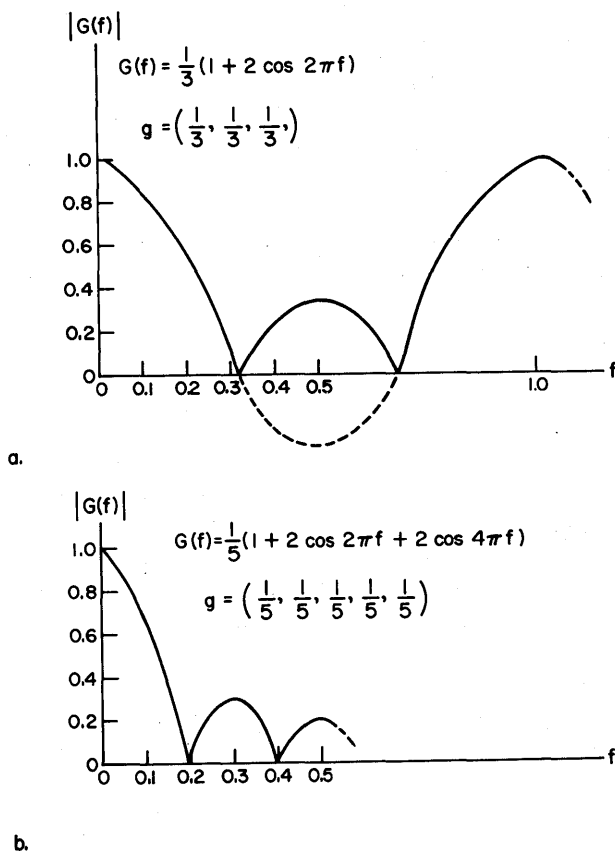


FIGURE 5—Transfer functions for equal weight low pass filters:
 (a) Three point;
 (b) Five point.

be the negative of the input. Figure 5b shows the MTF for the five point equal-weight filter. As would be anticipated, averaging five points results in greater attenuation of high frequencies than for three point averaging. In comparison, the MTF of the five point third degree filter shown in Figure 6 produces less attenuation than either equal weight filter.

For all of the filters discussed above, the weights were selected first and then the effective filter MTF was calculated. To obtain a filter with a given MTF, the weights are calculated by taking the inverse Fourier Transform of the specified MTF (see Appendix C).

Specific applications of filtering and of other image enhancement techniques are presented next.

Computer image enhancement applications

Before-and-after examples of computer enhanced pictures are described in this section which illustrate four classes of two-dimensional linear filters, non-linear filters including non-linear contrast enhancers and image subtraction. The types of images processed in-

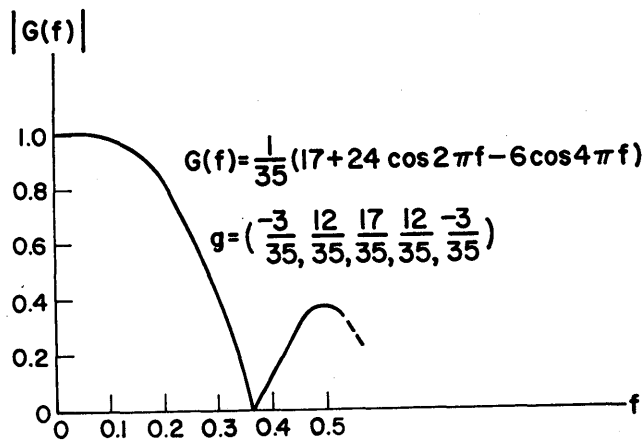


FIGURE 6—Transfer function for a five point third degree least-square filter.

clude x-ray films, isotope scanner films and photomicrographs.

All of the pictures were scanned with a cathode-ray tube flying spot scanner and digitized to six bits (0-63 counts). The system provides a maximum of 1200 scan lines per inch but only accepts film about 2 inches square. Thus larger film such as X-ray films are first photographically reduced and then scanned. Considerable degradation results from this reduction process and in addition some random noise is injected into the image by the film scanner. Photographic reduction can be eliminated by directly scanning the original full-size film with high-quality image dissector television systems that are available today. Scanning noise can easily be minimized either by leaving the scanning spot on each picture point for a long period and then integrating the resulting signal from the photosensing device or by scanning a picture several times at a normal rate and then averaging the frames in the computer. Although this latter technique has recently been adopted at JPL with very good results, most of the pictures shown in this section were obtained with a single fast scan and thus have unnecessarily high noise levels. In order to demonstrate the effect of the computer processing independently from the scanning losses, the enhanced pictures below are compared with the unenhanced but digitized versions rather than with the original films.

Low pass filter applications

The removal of high-frequency components from a picture may be desirable in a variety of situations. The most common application is to pictures containing excessive random noise which makes large low-contrast features difficult to see clearly. Low-pass filtering of

noisy pictures is frequently necessary when the computer is used for pattern recognition or measurement purposes. There are also occasions where it is useful to remove non-random high-frequency structure such as sharp edges that are not important and make the rest of the picture difficult to view.

As an example of the application of a low-pass filter to a picture, consider the tomograph section of the ear shown in Figure 7. This picture is composed of 1000 lines and 1000 elements per line. The sample and line spacing on the original film were 25 microns. The feature to be enhanced is the cochlea, the light spiral in the upper half of the picture. In this case, the noise-like structure results from the spatial fluctuations of the exposing radiation which is characteristic of tomographic imaging systems. The filter selected was an averaging filter consisting of equal weights in a 31 X 31 array. The size was chosen large enough to remove the noise but not so large as to remove the image of the cochlea which varied in size from about 30 to 50 samples across.

A second example of the use of a low-pass filter is shown in Figure 8. This figure shows an unprocessed picture of the calcaneus on the left and a filtered ver-

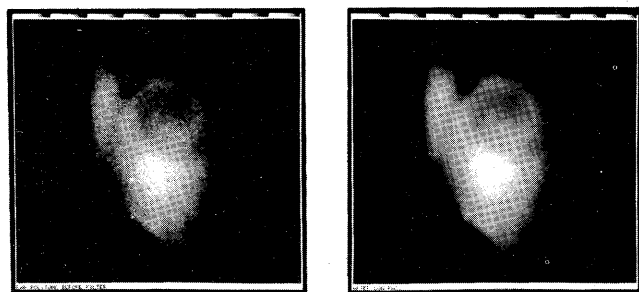


FIGURE 7—X-ray tomograph of the cochlea (light spiral) (left) unprocessed (right) after 31 X 31 equal weight low-pass filter.

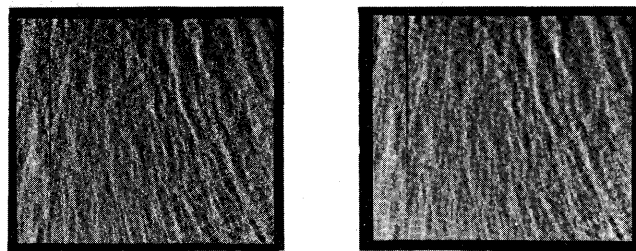


FIGURE 8—Radiograph of the calcaneus: (left) unprocessed (right) after 7 X 7 equal weight low-pass filter.

sion on the right that was obtained by applying a 7×7 equal weight filter. In this particular picture, the viewer does not have great difficulty following the trabecular shadows (the vertical light streaks) but the computer was subsequently required to detect and measure the width of each shadow, a job made extremely difficult by the high-frequency noise. An example of a picture in which disturbing non-random structure appears that can be removed by filtering is shown in Figure 9 (top). This chest film generated by an isotope scanner characteristically has square-like picture elements and white bars between each line. Useful information in the film is obscured because the eye is attracted to the sharp edges and lines. The lines were first removed by the computer, as shown in Figure 9 (middle), (this process will be described later), and then, as shown in Figure 9 (bottom), an 11×11 equal weight low-pass filter was applied to eliminate the square-like structure of the original individual picture elements.

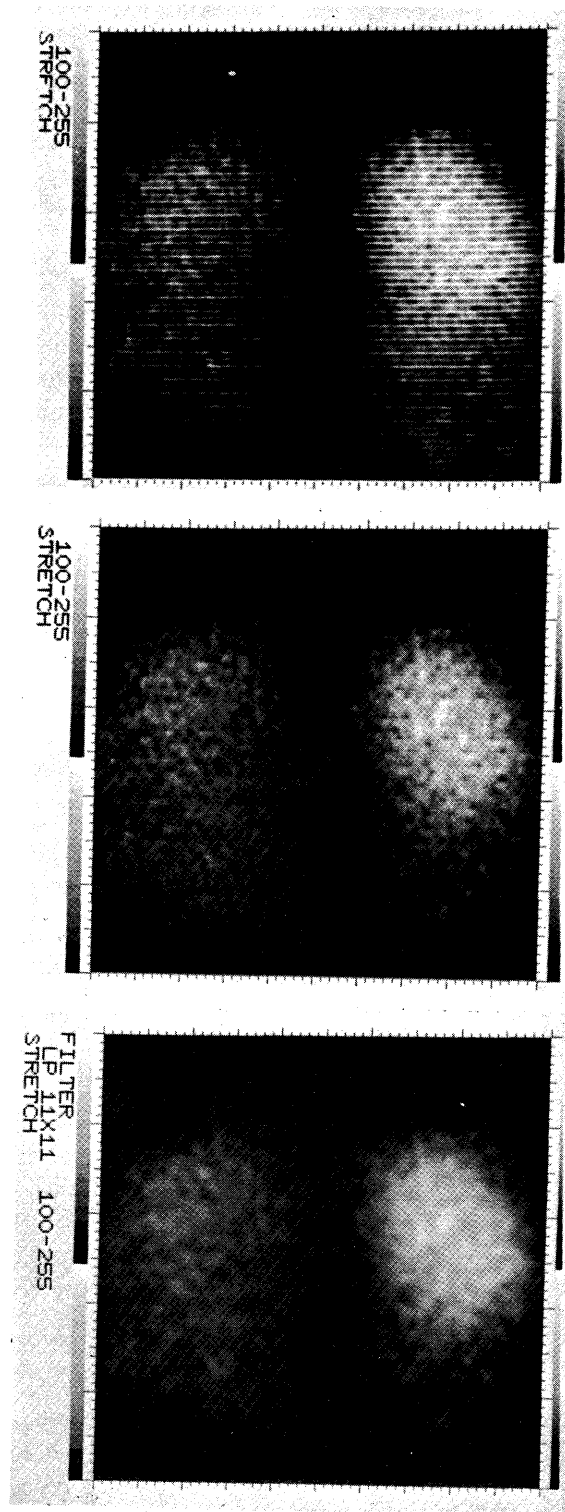
Radiograph noise resulting from quantum fluctuations in the source is not concentrated at the high-frequency end of the spectrum as might be assumed from the salt-and-pepper effect produced. Rather, the noise is somewhat evenly distributed over all frequencies passed by the system. Consequently, a simple low-pass filter cannot remove noise in many cases and may even result in a poorer picture than the original by producing a clumping or mottled effect such as that shown in the radiograph of the spine in Figure 10. This effect is less noticeable in Figures 7 and 8 because considerably more of the high-frequency signal was removed from these pictures.

High pass filter applications

A common problem in X-ray photography is the visualization of small low-contrast features when they are superimposed onto a very dark or very light background. Direct contrast enhancement will not improve the image because the film or the print will saturate. The solution is to apply a high-pass filter that removes the background by converting constant or very slowly changing dark or light areas to grey. The smaller low-contrast superimposed feature also moves to grey but now contrast enhancement may be applied without saturation occurring.

The basic shape of a high-pass filter is shown in Figure 4b. Removing the constant or dc component of an electrical signal generally means centering the signal about zero. For a picture system of the type discussed here where maximum white is represented by the number zero and the maximum black by the number 63, removing the dc component means centering the picture about mid grey, represented by the number 31.

FIGURE 9—Radioisotope scanner chest film;
(top) unprocessed image;
(middle) after removal of scan lines;
(bottom) after 11×11 equal weight low-pass filter applied
to picture with the scan lines removed.



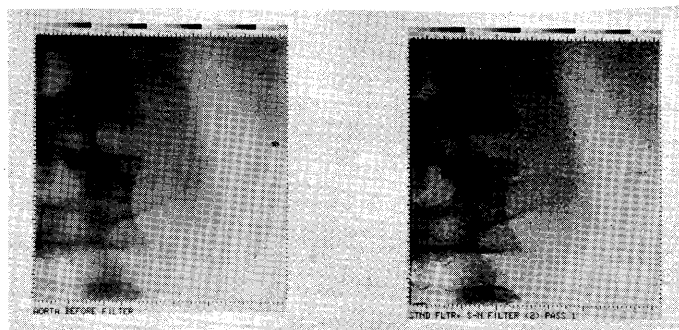


FIGURE 10—Radiograph of the spine showing the mottling caused by removing only high-frequency signal components when random noise is present at all frequencies; (left) unprocessed; (right) after low-pass filter.

Thus the filtering equation 1 becomes

$$y_n = 31 + \sum_{k=-K}^K g_k x_{n-k}$$

An example of before and after high pass filtering is shown in Figure 11. The left figure is a radiograph of a bone in which an opaque dye has been injected into the blood. By varying viewing illumination on the original film the blood vessels alongside the bone and those in line with the bone could be seen. Similarly, a darker print would bring out those alongside the bone, while probably rendering the bone itself solid black. The problem is simply that the range of film density is too large. The high-pass filter narrows that range by removing background which forces the data to more satisfactorily match the film characteristics. The information that is thrown away (the background shading) is irrelevant. Figure 11 shows the result of first applying a high-pass filter and then increasing con-

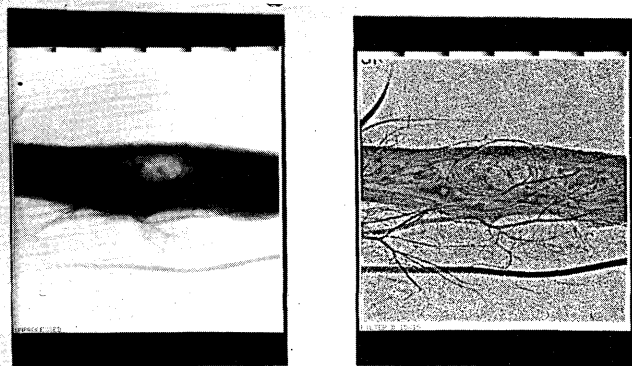


FIGURE 11—Angiogram showing background removal; (left) unprocessed; (right) after high-pass filter and contrast enhancement by a factor of four.

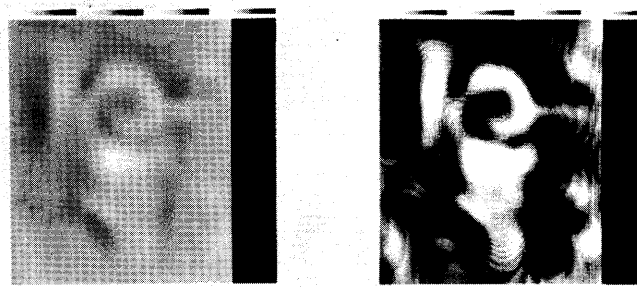


FIGURE 12—Tomograph of the ear; (left) after high-pass filter (right) after high-pass filter and contrast enhancement

trast by a factor of four. Since increasing contrast by four is equivalent to applying a filter that uniformly amplifies all signal frequencies by four, a substantial amount of noise amplification takes place as is clearly evident in this example.

A high-pass filter applied to the ear tomograph of Figure 7 is shown in Figure 12 on the left. The result of increasing the contrast is shown on the right.

The filter weights for this example are defined as

$$g_{k,\ell} = 1 - \frac{1}{(2K + 1)(2L + 1)} \text{ for } k = 0, \ell = 0$$

$$= - \frac{1}{(2K + 1)(2L + 1)} \text{ otherwise}$$

Application of these weights using equation 1 is exactly equivalent to subtracting the average of the $(2K + 1) \cdot (2L + 1)$ points surrounding x_n from x_n itself. Then as previously indicated, this difference is added to 31 to produce the filtered point y_n . With this interpretation in mind, it becomes clear that the size of the filter weight matrix must be substantially larger than the largest feature that is to be left in the picture. Otherwise, the feature contributes too heavily on the local average and may as a result be subtracted from the picture. For the ear tomograph a filter size of 55×55 was selected. Since the high-pass filter was applied to a picture that had previously been low-passed, the overall effect is that of a band-pass filter that removes both high and low frequencies. This same result could as easily be obtained from a single band-pass filter applied to the unprocessed picture.

A last example of high-pass filtering is shown in Figure 13 which shows pictures of the calcaneus before and after filtering. In this case, the filtering was not done for enhancement but to prepare the picture signal for a computer quantitation program. The shading shown in Figure 13 on the left was removed with a one-dimensional filter that removed the average calculated over

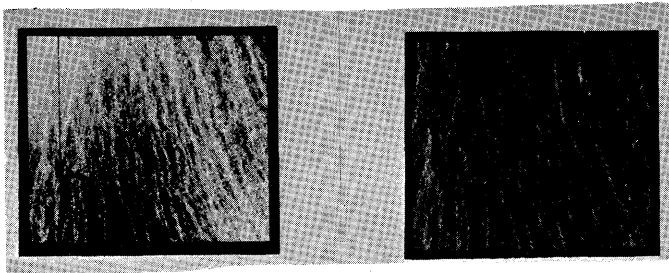


FIGURE 13—Radiograph of the calcaneus showing background removal; (left) unprocessed; (right) after 101 weight one-dimensional high-pass filter.

101 samples. Since the shading did not change abruptly from line to line, a one-dimensional filter was assumed sufficient. Although the shading is removed some of the horizontal trabecular structure also is removed,* an effect that was avoided on subsequent film by using a two-dimensional filter.

High-frequency restoration filters

Consider an imaging system with a modulation transfer function $H(f)$, indicated by Figure 14a, showing the usual dropoff in response at high frequencies. Restoration of this loss could be accomplished by a filter with an MTF, $G(f)$, equal to $1/H(f)$, as shown in Figure 14b. Thus the overall MTF of the system including the filter becomes

$$M(f) = G(f) \cdot H(f)$$

as shown in Figure 14c. The filter must level off at some point instead of continuing as the inverse of $H(f)$ in order to avoid excessive amplification of system noise.

Considerable work has been devoted to measuring the MTF of imaging systems. For example, the MTF of fluorescent intensifying screen and film combinations used in X-ray systems has been obtained by Rossman^{5,6} and Morgan⁷ as well as the MTF of the X-ray focal spot as a function of its size and geometry by Doi.⁸ It does not follow, however, that if the X-ray system MTF were known, the optimum high-frequency restoration filter would be the one whose transfer function was the inverse of the system MTF. The reason for this, aside from the problem of noise amplification, is that a distorted system response is as likely to enhance *important* information as a flat system response. The high-pass

*This effect could not be seen visually, but was detected by subsequent computer measurement of the horizontal shadow widths.

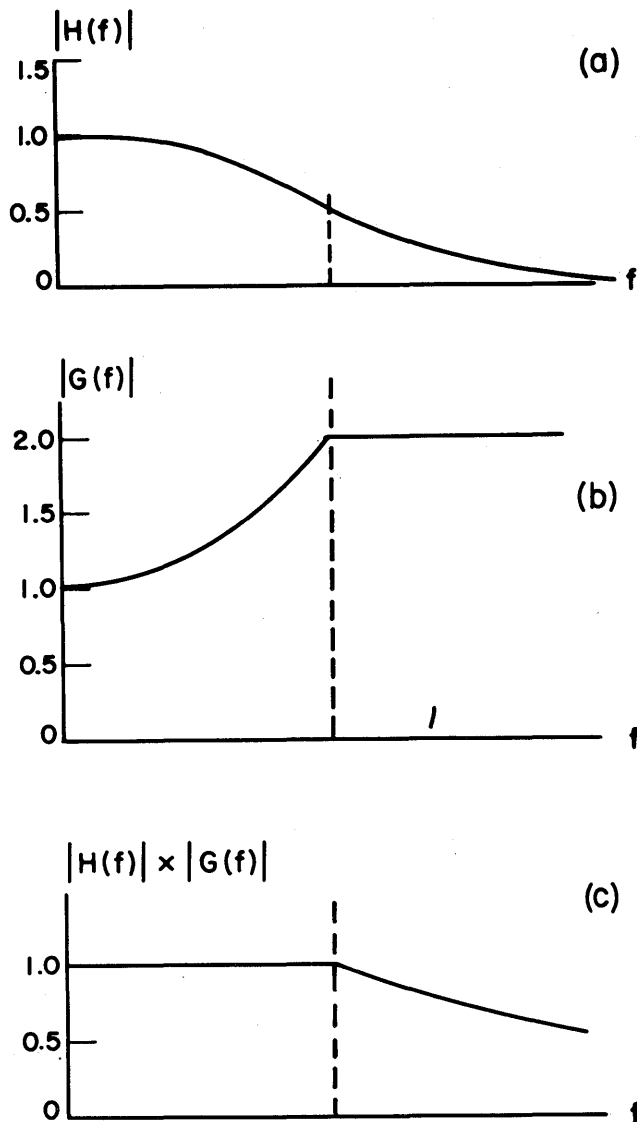


FIGURE 14—Diagram showing effect of high-frequency restoration; (a) Typical system MTF; (b) Filter MTF designed as the inverse of the system MTF; (c) MTF of overall system including filter.

filter examples demonstrated this. The important point is that optimization depends not only on the imaging system, but also on the subject, the (medical) purpose of the film and the perceptual response of the viewer. These areas are currently under investigation by Rossman,⁹ Tuddenham¹⁰ and others.

For systems with very high signal-to-noise ratio, high frequency restoration filters are quite effective. Figures 15 and 16 show examples of this type of filtering applied to pictures of the lunar surface televised by the Surveyor spacecraft. It might be noted that the

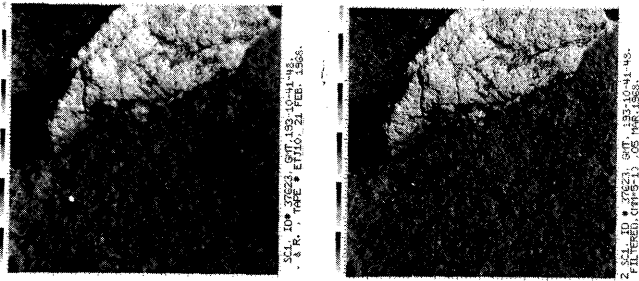


FIGURE 15—Surveyor I picture of the lunar surface; (left) unprocessed (right) after high-frequency restoration filter

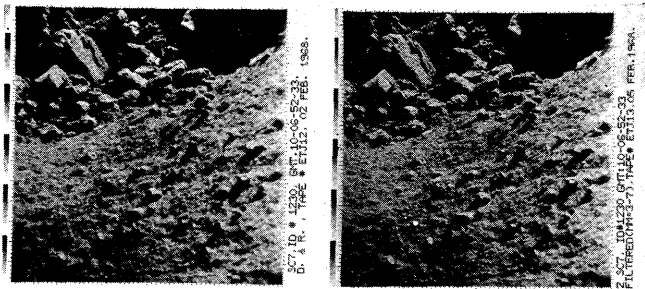


FIGURE 16—Surveyor 7 picture of the lunar surface; (left) unprocessed; (right) after high-frequency restoration filter.

received video signals were directly sampled and digitized without intermediate film scanning.

Applying high-frequency restoration to photomicrographs also provides some degree of detail sharpening, as seen in the chromosome pictures in Figure 17.

Unfortunately, the noise level in x-ray films is so high that this type of filtering is relatively less effective. An example of a high-frequency restoration filter applied to a noisy x-ray picture is shown in Figure 18. Some improvement in resolution of the intermediate-size blood vessels can be seen (see arrows) but the very

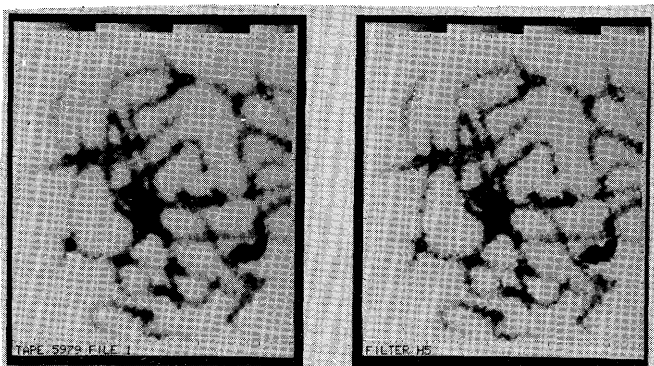


FIGURE 17—Photomicrograph of human chromosomes; (left) unprocessed; (right) after high frequency restoration filter.

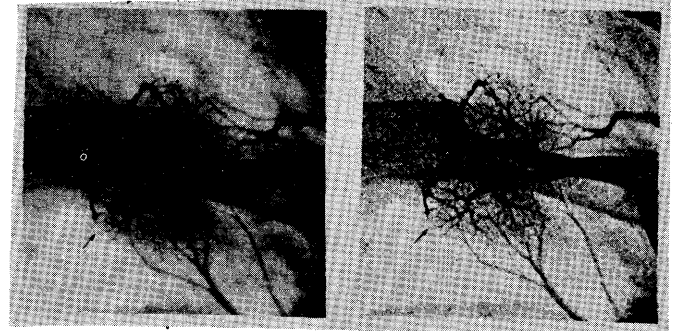


FIGURE 18—Pulmonary angiogram showing effect of high-frequency restoration; (left) unprocessed (right) after high-frequency restoration. Arrow indicates small blood vessel sharpened by filtering.

fine blood vessels are still lost in the noise. This does not mean that further improvement in resolution cannot be obtained, however. It does mean that the filters must be tailored more to image content than to the imaging system. Filters of this type are described next.

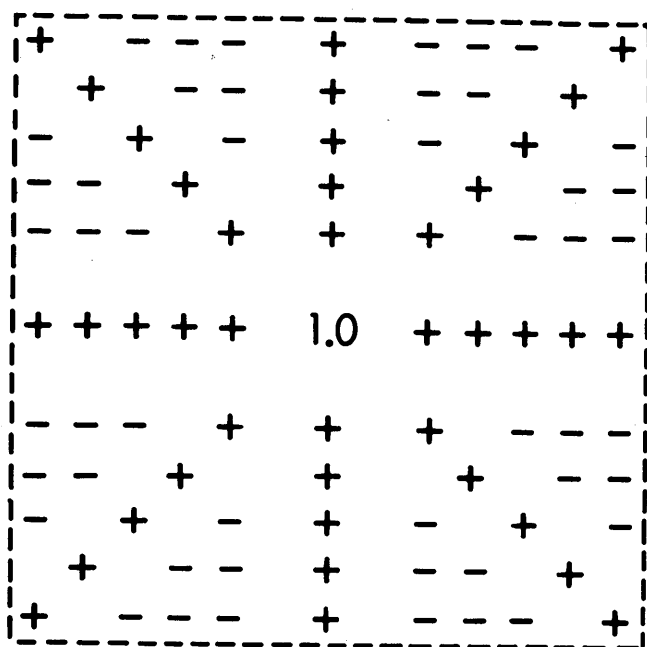
Feature-selective filters

Feature-selective filters operate by cross-correlating the picture with a matrix of weights that geometrically resembles the feature to be enhanced. If, for example, vertical lines are to be enhanced, the filter weights are made positive along a vertical line in the center of the matrix and negative otherwise so that the output of the filter is a maximum when the filter is centered on a vertical line in the picture. The weights shown in Figure 19 were selected to enhance the blood vessel shadows in the angiogram shown in Figure 20. The arrow points to some blood vessels faintly visible in the unprocessed film and better defined in the filtered version. The two lower pictures represent further enhancement using subtraction methods to be discussed later.

Filters of this type can enhance data or as easily be used to remove selected features. A radiograph of a thin section of bone from an excised vertebra is shown in Figure 21 (upper left). A subsequent computing requirement to measure the width of the horizontal trabecular shadows made it desirable to first remove all non-horizontal trabeculae from the image. The results of applying filters designed respectively to remove vertical trabeculae and trabeculae lying at minus forty-five degrees and plus forty-five degrees from vertical are also shown in Figure 21.

Non-linear filtering and contrast enhancement

Non-linearity in digital filters occurs when the filter output, as defined by equation 1, is truncated, either deliberately to produce a specialized effect or unavoid-



+ = 0.0250

- = -0.0207

ALL REMAINING POINTS ARE ZERO

SUM = 1.0

FIGURE 19—Feature-selective filter used to enhance lines that are linear or nearly-linear over a short distance.

ably when the output exceeds the allowable grey-scale range (i.e., saturation). The filtering procedure used to remove the white lines from the isotope scanner image of Figure 9 is an example of intentional non-linearity. Filtering was accomplished in two steps. First, equation 1 was applied using the following weights:

$$g = (1/6, 1/6, 1/6, 0, 0, 0, 1/6, 1/6, 1/6).$$

Next, the filtered point y_n was compared with the corresponding input x_n . If the filtered point was larger than the input, as would likely be the case if the input point was part of a line to be removed (white is represented by a low number), the filtered point was accepted as is. Otherwise, the input point was substituted. The effect of this procedure was to leave the picture unchanged whenever the filter was not centered on a line.

Simple contrast enhancement of the type shown in Figure 22a in which input points having values between A and B are linearly stretched to full scale is an example of filter non-linearity resulting from saturation. A more interesting procedure is to divide the input into several intervals and stretch each interval simultaneously. As shown in Figure 22b, the input range is di-

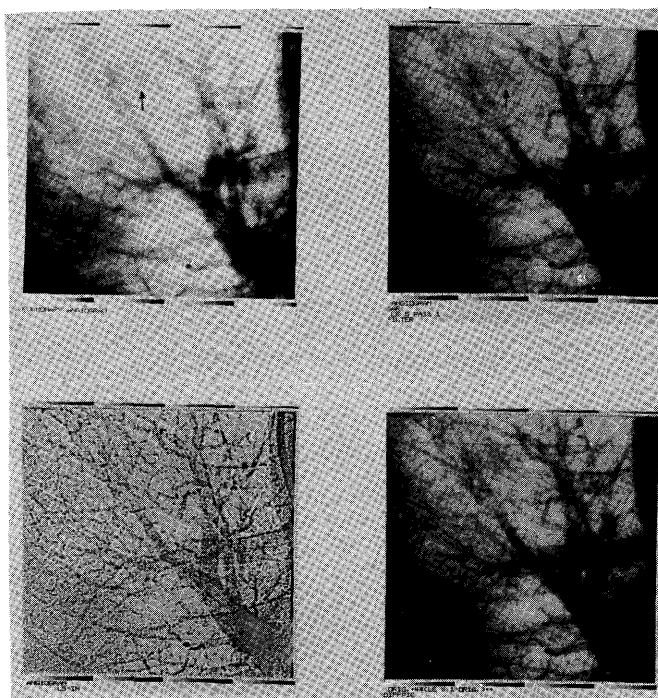


FIGURE 20—Pulmonary angiogram showing the result of applying a filter designed to enhance straight lines.

(upper left) unprocessed picture;

(upper right) after application of the filter shown in Figure 19. The arrow indicates blood vessels faintly visible on the unprocessed picture but more clearly defined on the filtered picture;

(lower left) The result of subtracting the unprocessed picture from the filtered picture in order to emphasize changes caused by the filter. Only positive differences were retained and contrast was greatly increased;

(lower right) The result of adding four times the positive difference picture to the unprocessed picture.

vided into four intervals and each interval is expanded to full scale. The effect of such a procedure on a picture is to display everything in the picture at a high contrast. An example of a four and eight cycle expansion applied to a photograph of the retina is shown in Figure 23 on the bottom. In particular, note the improved definition of blood vessel wall. The original picture is shown in the upper left and an enlarged version of the area of interest is shown in the upper right. There is, of course, a substantial amount of artifact created by this procedure, so caution must be exercised in the interpretation of these pictures.

Picture subtraction

Subtraction of radiographic images is a useful method for amplifying differences between two pictures. In particular, for an angiographic series, subtraction of the pre-dye picture from pictures which include the dye is

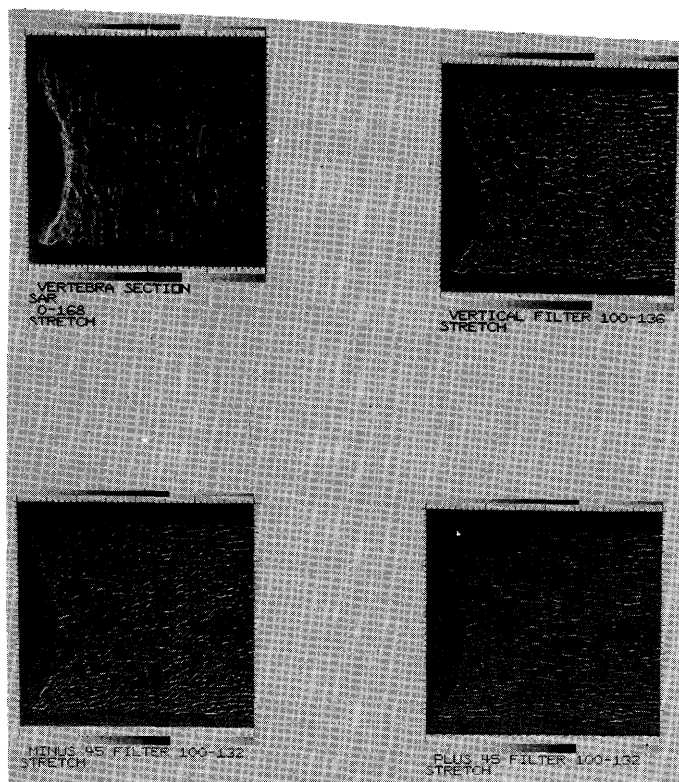


FIGURE 21—Radiograph of a thin section of a vertebra showing the effect of filters designed to remove non-horizontal trabecular shadows.

(upper left) unprocessed picture;
 (upper right) after filter to remove vertical shadows;
 (lower left) after filter to remove minus 45 degree shadows;
 (lower right) after filter to remove plus 45 degree shadows.

an effective means for removing bone shadows from the picture in order to see the fine blood vessels structure more clearly. Optical subtraction methods are in clinical use today for cases involving rigid structures such as the skull.¹¹ For non-rigid structures such as the chest, optical subtraction can still be performed, but difficulties in matching large areas usually restrict the subtraction to a small area.

Image subtraction can also be performed by computer and in the case of mismatching pictures such as described above, the computer can be used to geometrically distort one picture to match the second. An example of such an operation on a pair of chest films is described in reference 3. An advantage of computer subtraction as opposed to optical subtraction is that the difference picture is immediately available for further enhancement such as high-frequency restoration. Also, using cross-correlation techniques, the best match between pictures could be automatically accomplished by the computer. This "best" match would probably be superior to a match obtained manually by physically superimposing two films.

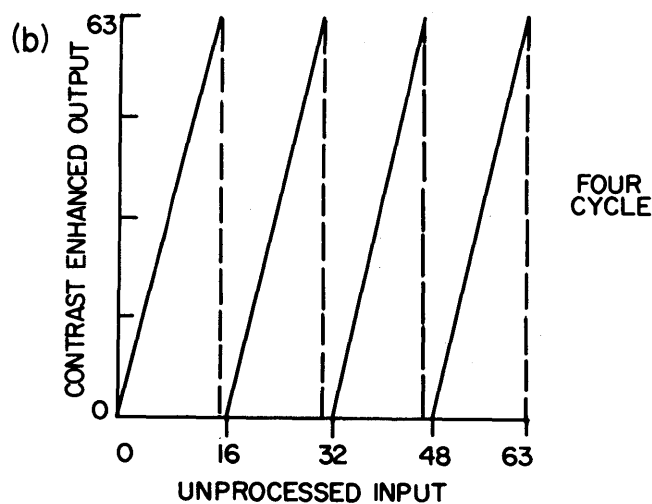
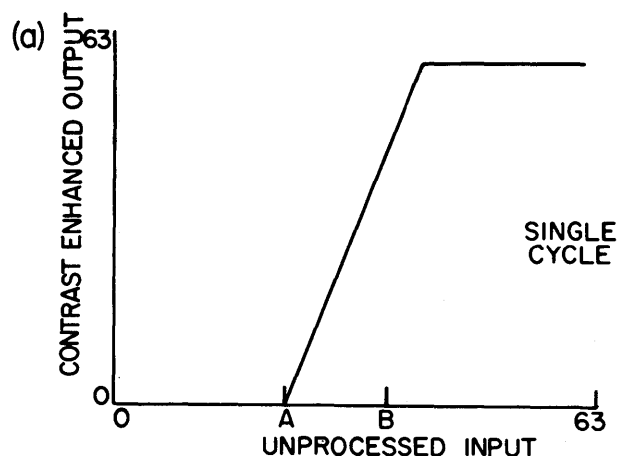


FIGURE 22—Transfer characteristic relating the picture input and output for contrast enhancement;

(a) simple enhancement;
 (b) four cycle enhancement.

Subtraction of an unprocessed picture from a filtered picture can also be a useful way to evaluate the filter itself when changes produced by the filter are rather subtle. The result of subtracting two pictures in this manner is shown in Figure 20 (lower left). Figure 20 (lower right) was obtained multiplying the positive part of the difference picture by four and adding this difference to the unprocessed picture.

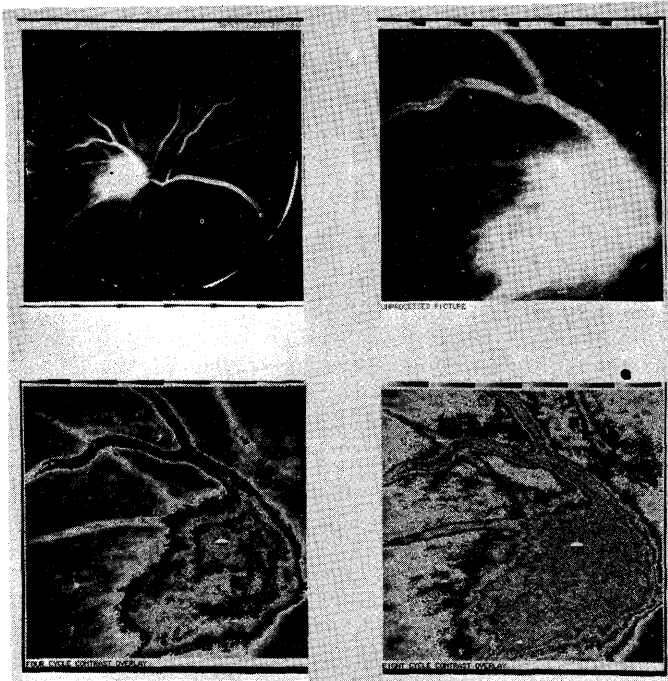


FIGURE 23—Retina photograph showing multiple-cycle contrast enhancement;
 (upper left) unprocessed picture;
 (upper right) enlargement of section to be enhanced;
 (lower left) result of four cycle overlapping contrast enhancement as indicated in Figure 22b;
 (lower right) result of eight cycle overlapping contrast enhancement.

The use of computers to make quantitative measurements on x-ray film

The use of a computer to make numerical measurements on x-ray film is a relatively new application that promises to be of great value.

In one of the earliest applications, by Becker et. al.,¹² a computer and flying-spot scanner were used to measure the maximum transverse diameter of the heart shadow and the maximum transverse diameter of the rib cage shadow. Their method involved summing each column of a digitized x-ray film of the chest and then using the sums, plotted as a function of distance across the chest to detect the location of the required features.

There is also a large amount of current research on the problem of measuring heart volume using X-ray cineangiography.¹³ Some of the techniques measure volume from bi-plane films by measuring total heart outline as revealed by the dye. Others only measure maximum heart length and width. Single plane methods utilize dye concentration to indicate heart depth. In each case, a computer is used for part or all of the total measurement.

In another application, a computer is used to locate and measure the width of trabecular bone shadows for

the purpose on determining bone mineralization and strength which is reflected in the trabecular pattern.¹⁴ In this application, the scanned film is first preprocessed by filtering to remove high-frequency noise and background shading. A line-by-line measurement is then made by the computer to determine mean trabecular width, spacing and frequency. The detection algorithm is of the peak-connecting type with minimum and maximum allowable width conditions imposed. Figure 24 shows an example of three lumbar vertebra films showing progressive demineralization (top row). The bottom row gives a pictorial representation of the computer trabecular shadow detection program. That is, each point of the top picture that the computer decides is part of a trabecula shadow is displayed in the bottom picture as maximum white. The rest is shown black. The main function of this display is to verify the accuracy of the detection process. However, it also represents a type of non-linear enhancement.

The originator of this technique, Dr. S. David Rockoff of Washington University in St. Louis, initially made these measurements with a mechanical film scanner and an analog computer. Approximately ten minutes per line were required to scan the film and hand compute the required parameters from analog plots. Consequently it was not practical to take more than a few scans per film. On the other hand, the computer prefiltered

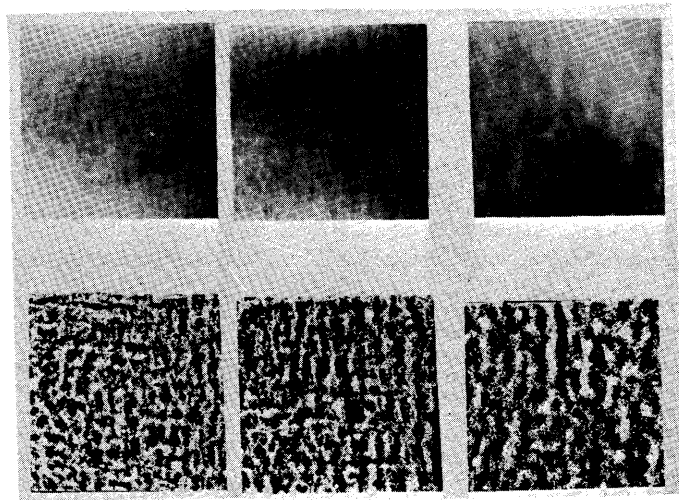


FIGURE 24—Lumbar vetebra radiographs and display of trabecular shadow detection.

(top row) three examples showing progressive demineralization from left to right. The two left-hand examples are *in vitro* radiographs while the right-hand example is an *in situ* radiograph;

(bottom row) pictorial representation of computer program to detect trabecular shadows. Each bottom row point is printed white if the corresponding point in the top row falls within what the computer decides is a trabecular shadow. Otherwise the point is printed black.

and measured 400 scan lines for the films in Figure 24 in four minutes, or about 0.6 seconds per line.

CONCLUSIONS

The goal of computer enhancement research on X-ray films is to produce medically useful pictures. While the initial results have been interesting and even encouraging, there is no question but that the goal has not yet been reached. It may be of interest to discuss some of the reasons for this.

First of all, image losses from film scanning are still very severe, although as previously discussed they can be decreased substantially with better scanners, particularly with the type that view the original film instead of a reduced version. Without these losses, an improvement in resolution can undoubtedly be obtained with the computer enhancement. In addition, since an enhanced picture in which selected features are more easily seen than on the original could help the radiologist screen films more rapidly or more reliably, resolution improvement need not be achieved for enhanced pictures to be useful.

A major obstacle to the application of computer enhancement at the clinical level, even aside from problems of image quality, is processing speed. Actual computer time may be small, varying from 20 or 30 seconds to 30 minutes or more, depending on the type of processing and the computer. However, the time required to photographically reduce a film (if that is necessary), scan it, process it in the computer and then reconstruct the computer output into a photograph is likely to be one or two days under even optimum conditions. The use of computer-controlled film scanners and on-line picture displays from the computer could reduce processing time by a large factor. However, systems of this sort are very expensive.

Perhaps the most medically promising application of computer image processing is in the area of quantitative measurement. The quantitation of trabecular patterns in bone as described above, for example, would be virtually impossible without either a digital or analog computer. The measurement of heart volume, which requires two or three weeks per patient when hand methods are used can be accomplished by computer in an hour or less. It seems likely that applications of this sort will increase very rapidly as film scanners become more available.

REFERENCES

- 1 R S LEDLEY et al
Optical and electro-optical information processing
MIT Press, Cambridge Massachusetts and London England
1965 Chapter 33 p 591
- 2 J M S PREWITT
The use of computers in radiology
University of Missouri April 1968 p A2
- 3 R H SELZER
The use of computers in radiology
University of Missouri April 1968 p A-84
- 4 R NATHAN
Pictorial pattern recognition
Thompson Book Company Washington DC 1968 p 239
- 5 K ROSSMAN
Modulation transfer function of radiographic systems using fluorescent screens
Journal of the Optical Society of America 52 774 1962
- 6 K ROSSMAN G LUBBERTS
Some characteristics of the line spread-function and modulation transfer function of medical radiographic films and screen-film systems
Radiology 86 235 1966
- 7 R H MORGAN L M BATES U V GOPALARAO A MARINARO
The frequency response characteristics of x-ray films and screens
Am J of Roent Rad Ther and Nuc Med 92 426 1964
- 8 K DOI
Optical transfer functions of the focal spot of x-ray tubes
Am J of Roent Rad Therapy and Nuc Med 94 712 1965
- 9 K ROSSMAN R D MOSELEY
Measurement of the input to radiographic imaging systems
Investigative Radiology to be published in the last half of 1968
- 10 W J TUDDENHAM
Visual search image organization and reader error in roentgen diagnosis
Radiology 78 694 1962
- 11 W HANAFEE P STOUT
Subtraction technique
Radiology 70 658 1962
- 12 H C BECKER W J NETTLETON P H MEYERS J W SWEENEY C M NICE
Digital computer determination of a medical diagnostic index directly from chest x-ray images
IEEE Transactions on Biomedical Engineering MBE 11 67 1964
- 13 Conference on Measurement of Heart Chamber Volumes and Dimensions Denver Colorado July 24-27 1967
Summary of proceedings published by the American Heart Association Inc
- 14 S D ROCKOFF R H SELZER
Radiographic trabecular quantitation of human lumbar vertebrae In Situ
Proceedings of the Conference on Progress in Methods of Bone Mineral Measurement Bethesda Maryland February 15-17 1968 Proceedings to be published as a special issue of Clinical Orthopedics and Related Research
- 15 J ORMSBY
Design of numerical filters with applications to missile data processing
Journal Society Indust Appl Math 9 440 1961
- 16 R H SELZER
An evaluation of three digital low-pass filters
MS Thesis UCLA Department of Engineering January 1962
- 17 G M JENKINS
General considerations in the estimation of spectra
Technometrics 3 138 1961

APPENDIX

Digital filter representation in the frequency domain

Let x_n , $n = 0, \pm 1, \dots$ be the discrete series obtained by sampling film every T millimeters and y_n , $n = 0, \pm 1, \dots$ be the output of a digital filter applied to x_n . If the filter weights are given by $g = (g_{-K}, \dots, g_K)$, the filter equation is given by

$$y_n = \sum_{k=-K}^K g_k x_{n-k} \quad (\text{A1})$$

$$n = 0, \pm 1, \dots$$

Let the discrete Fourier Transform of y_n be defined as

$$Y(f) = \sum_{n=-\infty}^{\infty} y_n e^{-j2\pi f n T} \quad (\text{A2})$$

Substituting A1 into A2,

$$Y(f) = \sum_{n=-\infty}^{\infty} \left[\sum_{k=-K}^K g_k x_{n-k} \right] e^{-j2\pi f n T}$$

and after rearranging terms,

$$Y(f) = \sum_{k=-K}^K g_k \sum_{n=-\infty}^{\infty} x_{n-k} e^{-j2\pi f n T}$$

After making the change of variable $m = n - k$,

$$\begin{aligned} Y(f) &= \sum_{k=-K}^K g_k \sum_{m=-\infty}^{\infty} x_m e^{-j2\pi f (m+k) T} \\ &= \sum_{k=-K}^K g_k e^{-j2\pi f k T} \sum_{m=-\infty}^{\infty} x_m e^{-j2\pi f m T} \\ &= G(f) \cdot X(f) \end{aligned}$$

where we define $g_k = 0$ for $|k| > K$.

Thus the input and output of a digital filter are related in the frequency domain by $G(f)$, the filter transfer function that is found to be the discrete Fourier Transform of the filter weights.

Calculation of digital filter modulation transfer functions

Now consider the MTF of some of the filters discussed earlier in the paper. For example, the three point low-pass filter was defined by the weights.

$$g = (1/3, 1/3, 1/3)$$

Thus,

$$\begin{aligned} G(f) &= \sum_{k=-1}^1 (1/3) e^{-j2\pi f k T} \\ &= 1/3 + 2/3 \cos(2\pi f T) \end{aligned}$$

It is convenient at this point to express the period of the sinusoids in units of samples rather than in millimeters. Thus the unit of frequency becomes cycles per sample and the sample interval T is equal to one as is the sample frequency f_s . Thus a frequency of 0.1 indicates a sinusoid whose period is 10 samples. Conversion of the period to millimeters is accomplished by multiplying by the actual sample spacing.

Thus, with f in cycles per sample, the MTF for the three point averaging filter becomes

$$G(f) = 1/3 + 2/3 \cos(2\pi f)$$

A plot of $|G(f)|$ is shown in Figure 5a.

A basic property of Fourier Transforms is that the transform of an even function is real-valued. Thus all filters symmetric about the center weight, have real valued transfer functions. Figure 5b also shows the MTF for the five point averaging filter, $g = (1/5, 1/5, 1/5, 1/5, 1/5)$ which is calculated as

$$G(f) = 1/5 [1 + 2 \cos(2\pi f) + 2 \cos(4\pi f)]$$

As would be anticipated, the five point filter falls off faster and thus removes more of the high-frequency signal.

In comparison, the five point third degree least squares filter has MTF.

$$G(f) = 1/35 [17 + 24 \cos(2\pi f) - 6 \cos(4\pi f)]$$

and as shown in Figure 6 has a flatter low-frequency response than either of the equal weight filters.

Earlier it was shown that if g_k are weights for a low-pass filter with MTF $G(f)$, then a high pass filter is defined as

$$g'_k = \delta(k) - g_k \quad k = 0, \pm 1, \dots, \pm K \quad (\text{B1})$$

Further, it was stated that the MTF $G'(f)$ of this filter is given by:

$$G'(f) = 1 - G(f).$$

To show this is true, three properties of Fourier transforms are required. Let $F[\]$ indicate the Fourier trans-

form of the bracketed quantity. Then if $g_1(t)$ and $g_2(t)$ are two spatial functions, the following properties are true:

- (a) $F[g_1(t) + g_2(t)] = F[g_1(t)] + F[g_2(t)]$
- (b) If $A = \text{constant}$, $F[Ag_1(t)] = AF[g_1(t)]$
- (c) $F[\delta(t)] = 1$, where $\delta(t)$ is an impulse at the origin

The desired result follows by taking the transform of both sides of equation (B1).

Digital filter synthesis

Frequently it is desirable to generate a digital filter with a particular given MTF by obtaining the weights from the inverse Fourier transform of the given MTF. Suppose $G^*(f)$ is the given MTF and the sample interval is one. Then the inverse transform is

$$g_k^* = \int_{f=-\infty}^{\infty} G^*(f) e^{i2\pi f k} df. \quad (C1)$$

Suppose a truncation function W_k is defined as

$$W_k = 1 \text{ for } -K \leq k \leq K \\ 0 = \text{otherwise}$$

and we define a truncated sequence of non-zero weights as

$$g_k = g_k^* W_k. \quad (C2)$$

This truncated weight sequence gives an approximation to $G^*(f)$. The MTF is calculated in the usual manner as

$$G(f) = \sum_{k=-K}^K g_k e^{-i2\pi f k}$$

Let the mean square error difference between $G^*(f)$ and $G(f)$ be defined as

$$E = \int_{-0.5}^{0.5} [G(f) - G^*(f)]^2 df$$

It is known that the filter defined by equation C1 has the smallest E among all possible $2K + 1$ weight filters.¹⁵ However, it is also a fact that a discontinuity in $G^*(f)$ causes overshoot in $G(f)$ which does not decrease to zero as K goes to infinity.¹⁶

This overshoot can be avoided by using a different

truncation function W_K . For example, the function

$$W_K = 1 - \frac{|k|}{K} \text{ for } -K \leq k \leq K \\ = 0 \text{ otherwise}$$

when applied to g_k^* produces a filter whose MTF uniformly converges to $G^*(f)$ but at a slower rate than the filter derived from the uniform truncation function.¹⁷

There are a considerable number of possible truncation functions whose use depends in general upon the shape of the desired filter and the type of errors that can be tolerated.

Two-dimensional filter synthesis

Given the two-dimensional modulation transfer function $G^*(u, v)$, where u and v are spatial frequencies in the horizontal and vertical direction, respectively, the two-dimensional Fourier transform is found as

$$g_{k,\ell}^* = \iint G^*(u,v) e^{i2\pi(uk+v\ell)} du dv \\ k = 0, \pm 1, \dots \ell = 0, \pm 1, \dots \quad (D1)$$

Filter weights $g_{k,\ell}$ are obtained by truncating $g_{k,\ell}^*$. That is,

$$g_{k,\ell} = g_{k,\ell}^* W_{k,\ell}$$

where $W_{k,\ell} = 0$ for $|k| > K$, $|\ell| > L$. The MTF of this filter is obtained as

$$G(u,v) = \sum_k \sum_{\ell} g_{k,\ell} e^{-i2\pi(uk+v\ell)}$$

Typically, the three-dimensional surface $G^*(u, v)$ is known only along the horizontal and vertical axis. This necessitates generation of a surface to fit these known functions. A common assumption is that all equi-response contours of $G^*(u, v)$ are ellipses. This assumption plus the condition that the horizontal and vertical responses, $G(u, 0)$ and $G(0, v)$ are monotone functions of frequency allows generation of $G(u, v)$ by iterative solution of the equations,

$$u/u_0 + v/v_0 = 1 \text{ and } G(u_0, 0) = G(0, v_0) \\ = G(u, v)$$

In the above, as shown in Figure 25, u and v are given, u_0 is chosen arbitrarily, v_0 determined from the first equation and then the quantity

$$|G(u_0, 0) - G(0, v_0)|$$

is calculated. If this difference is too large, the procedure is repeated with a new u_0 chosen in a direction to decrease the difference and the procedure repeated until sufficient accuracy is obtained.

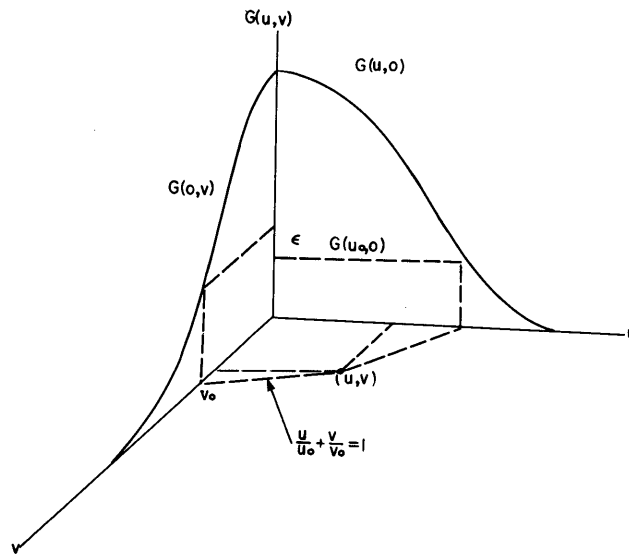
If $G^*(u, v)$ has circular symmetry, then so does $g^*(u, v)$ and equation D1 reduces to the one-dimensional Hankel transform. That is if $q^2 = u^2 + v^2$, $r^2 = k^2 + \ell^2$ and $G^*(u, v) = g^*(q)$, then the Hankel transform to generate filter weights for a symmetric MTF is

$$g(r) = 2\pi \int_{q=0}^{\infty} G^*(q) J_0(2\pi qr) q dq$$

where $J_0(2\pi qr)$ is a zero order Bessel function of the first kind,

$$J_0(2\pi qr) = 1/2\pi \int_{\phi=0}^{2\pi} e^{i2\pi q r \cos\phi} d\phi$$

FIGURE 25—Diagram showing the method used to obtain the surface $G(u,v)$ from the projections $G(u,0)$ and $G(0,v)$. $G(u,v)$ is taken equal to $G(u_0,0)$ when ϵ is less than some predetermined value.



A computer system designer's view of large scale integration

by M. E. CONWAY* and L. M. SPANDORFER

Consultant
Boxford, Massachusetts

Sperry Rand Corporation
Philadelphia, Pennsylvania

INTRODUCTION

Russell has defined *faith* as the willingness to adhere to a belief in the face of evidence to the contrary. According to this definition it is quite appropriate to describe as faith the forecasts being made by many industry optimists about the imminent future of Large Scale Integration (LSI) of semiconductor logic in digital computers. This paper will attempt to cast a realistic light on the promises and possibilities of mainframe LSI. Two major topics will be discussed:

- A. Semiconductor technology: What major improvements are possible and under development, and what difficulties must be overcome before these possibilities materialize.
- B. Computer design: In what ways can the special characteristics of LSI enhance the design of general-purpose computers, and why are some proposals not as promising as they might first seem.

The reader will be left to draw his own conclusions about future possibilities; our purpose is not to predict the future but to provide a basis for evaluating the many predictions already available. The subject is a complex one and covers several technical disciplines; a proper evaluation of the potential influence of semiconductor technology on computer design in the next decade will require a generous share of faith, accompanied by a probing skepticism.

*Dr. Conway's contributions to this paper were made when he was with Univac Division of Sperry Rand Corporation.

A technology appraisal

The original impetus for LSI arose out of (a) observations of the potential yield of large area chips on silicon wafers coupled with the possibility of improved tolerances by nearly an order of magnitude, and (b) the desire to reduce the proliferation of solder-type joints. The early economic picture was not clear, although hope existed for an eventual cost reduction in escalating from small chips to the higher level of batch fabrication. Now, after several years and considerable development, there is little reason to doubt that semiconductor technology will be able to process wafers with large complex chips containing hundreds of gates. Although much progress has been made at the wafer level, major unsolved problems still exist in (a) fabricating multilayer connections on the chip, (b) providing connections to a multi-pad chip, and (c) heat dissipation from the chip. Economics has not become clarified, but instead has grown to be a problem of the first magnitude.

The status and long-range forecasts for bipolar and MOS LSI have been covered by semiconductor technologists in excellent review papers.^{1,2} Briefly, current third generation computer integrated circuits are characterized by 1500-3600 square mil chips; production wafers range up to 2 inches in diameter. The larger wafers provide up to 1000 chips, each containing about 2-6 gates and nominally 14 connection pads. Yield figures, although highly proprietary, can exceed 50% at the wafer level if circuits are carefully designed with moderate tolerances and specifications. Fault data on current wafers, in agreement with proposed defect models, show a low but useful

~ (5%) potential yield for chips in the 6000-12000 square mil range. Tolerances down to 0.25 mils, commensurate with those used in higher performance devices in existing systems, currently provide component densities sufficient to realize 50 bipolar gates or several hundred MOS shift register stages in a chip less than 10,000 square mils. Long-range forecasts indicate the possibility of three-inch wafers, chip sizes up to 60,000 square mils, and 0.1 mil tolerances which will enable a factor of 3 increase in gate density, thus providing the basis for up to 1000 bipolar gates on the larger chips. Device and circuit innovations can be used to achieve even higher densities; an exploratory design has been reported which provides a density of 10^5 gates per square inch.⁸

For purposes of calibrating the present whereabouts of LSI from a usage standpoint, it should be noted that although many development programs are currently under way, as of mid-1968 no major LSI hardware is believed to be doing production work for ultimate equipment users. Known exceptions are a number of systems containing scratchpad memories employing 16-bit bipolar chips. The time required for new technology to evolve at the system level is substantial; it may be further noted that despite the obvious theoretical advantages of LSI in high-speed systems, none of the large scale high performance commercial computers (i.e., average monthly rental in excess of \$50,000) delivered by early 1968 is even based upon monolithic integrated circuits (excluding control memory and special subsystems). Several currently announced systems are expected to cross the small chip integrated circuit hurdle.

The large number of gates per chip cited above reflect only the geometrical limitations incurred in wafer processing and do not necessarily take into account important limiting factors such as heat dissipation (for bipolar circuits), wiring space, and chip input-output connections. Heat dissipation does not appear to be a fundamental limitation at the higher gate count for circuits operating slower than about 30 nanoseconds. Circuits in the one nanosecond range present as yet unsolved chip thermal problems in the 50-100 gate range, and almost an order of magnitude improvement in circuit power-delay product and/or further development of advanced cooling techniques will be required before LSI can make a serious dent in line delay losses and provide an average in-use

delay of less than 2 nanoseconds per gate. Current mainframe production is limited to an average in-use gate delay of about 4 or 5 nanoseconds, a value which appears to have only been attained with discrete components; about half the delay is attributable to line propagation and reflection effects. The state of the early production art in high-speed monolithic chips is around 1.5 nanoseconds unloaded and has been realized with current mode circuits operating at 50 milliwatts. It appears that circuit power in the 1-2 nanosecond range may be reduced by at least a factor of two with only a slight loss of speed, providing the load is on the same chip as the driver. Nevertheless, total dissipation for a 50 gate chip still approaches 2 watts, taking into account the need for off-chip drivers. Although several watts is comparable to or less than the power density of a commercial power transistor, it has not yet been demonstrated that the packaging problems at this level will result in LSI reliability.

The present design trend in LSI offers no panacea for reducing the number of wiring layers. Circuits on the chip are purposely close-spaced to minimize silicon area requirements, thus reducing the area available for wiring and creating the need in general for more than two layers. In the limit, the space available for signal wire routing to a first approximation is a function of the ratio of the length of the circuit edge to the wire width, although in any given case the space depends on various parameters such as via diameter, etc. For example, using the long-range forecasts of 50 square mil circuits and interconnect widths of about 0.2-0.25 mils, the resultant ratio of 35 or less is not significantly different from that found in various examples of current close-spaced technology. This suggests that an LSI chip will require about as many signal wiring layers as found on IC cards of comparable logic power. IC technology has the advantage that circuit density can be tailored to match the wiring space; the LSI chip is area-constrained due to yield. Multilayers have proved to be difficult in production at even the two-layer level; recourse to off-chip wiring would negate some of the attractiveness of LSI. Paradoxically, LSI incurs this potential limitation on internal interconnects at about the integration level where it becomes attractive from a circuit-to-I/O pad ratio standpoint. A reduction in circuit packing density by a factor of 2-3 below current practice at the 100 gate level may be required to offset the need for more than two metal levels.

LSI memories show considerable potential in the range of several hundred nanoseconds down to several tens of nanoseconds. In contrast with logic, LSI memory is ideally suited to exploit the advantages and liabilities of large chips: partitioning is straightforward and flexible, a high circuit density can be obtained with a manageable number of input-output pads, and the major economic barriers of part numbers and volume which confront LSI logic are considerably lower. Small-scale memory cell chips have already superseded film memories in the fast scratchpad arena; the depth of penetration into the mainframe is the major unresolved question.

Assuming conservative yields, simple calculations indicate the eventual cost of memory chips in the 256 bit range can go well below one cent per bit at the wafer level. Peripheral circuits, packaging, test and the usual other factors will raise the subsystem cost to a level which, although currently speculative, appears highly competitive with core and plated wire at the several cents per bit level. The number of connections in a mainframe LSI memory would be substantially greater than in a magnetic memory but on the other hand not vastly different from the count in existing small chip medium-scale processors. The array in an 8K word module, for example, with 256 bit chips and a performance level which would permit on-chip decoding requires on the order of 15K connections; a high performance module without on-chip decoding would require an additional factor of about three. Volatility should not be a serious issue since recovery can be managed by customary restart procedures. In the low-performance small I/O buffer area where circuit costs heavily outweigh stack costs, core still dominates and LSI has yet made no inroads. The situation in this volume market is interesting in view of the apparent simplicity of small all-semiconductor buffers; the observation can serve as an additional calibration point for the status of LSI.

Economic considerations

The major economic hurdles for LSI arise in connection with a series of closely related issues inherent to the very essence of LSI, namely the increased level of circuit integration. The issues are denoted as the part number, volume, design, test, and change problems.

The part number and volume problems can be

illustrated with reference to processor requirements. A typical small third generation processor, exclusive of I/O control, might utilize about two thousand logic gates or under 700 chips, assuming a three-gate median chip. The chips might be subdivided into a half dozen part numbers, each with varying fan-in and gate count. A fully loaded card might carry about 50 gates; 50 printed circuit cards would characteristically be used and, in the worst case, each would be unique. Assuming for simplicity that one LSI chip will be used to replace the circuits on a card, a total of 50 unique chips are required. In terms of the customary high-volume, low unit-cost practices of semiconductor manufacturing, two important parameters have moved in unfavorable directions: Part numbers have increased and are in essence equal to the card part numbers, and parts per part number have decreased to a value approaching unity. An even less desirable situation exists in large scale processors where, say, a 20,000 gate unit using 40 gate chips might, without favorable partitioning, require on the order of 50-80 part numbers when all wiring differences are taken into account.

As for total volume, the existing market for the low performance machines of any given third generation family appears to be numbered in the thousands, or low tens of thousands; the potential for the high performance systems is numbered in the hundreds. Studies of growth potential suggest a factor of 2.5 overall increase in general purpose digital computer shipments by 1975.⁴ The potentially large volume situation in memory and the volume requirements for low cost, short word length, instrumentation-type computers and other classes of equipment are more encouraging. Terminal equipment is undergoing a very rapid growth; comparatively few part numbers might be required with LSI.⁵ Part numbers for I/O controllers may be as high as for a small processor but the part volume is growing steadily. It is of interest to note that the venerable processor may soon surrender its long-time role as pacesetter for new technology.

Using present yields and manufacturing costs as a basis, it is plausible that LSI arrays will be available in several years at a manufacturing cost of around 3 cents per gate. To this figure must be added costs related to engineering design and test procedure determination, manufacturing final test, packaging, and other elements of mark-up, and profit. Whereas the latter are more specula-

tive at this time than the cost of gate fabrication at the wafer level, it is reasonable to believe that the total price per gate will evolve in 3-5 years to be 15 cents and less in arrays at the volume level of several tens of thousands cited for the small processor. On the other hand, the order of magnitude lower volume situation in large scale processors, coupled with the technology hurdles, requires the simultaneous realization of the more optimistic forecasts in market size, partitioning, and engineering costs before a favorable comparison can be made between nanosecond LSI and projections in small chip technology.

The increase in the level of engineering design implied by the growth of semiconductor part numbers does not confront the system manufacturer with an altogether new situation since he is already accustomed to providing partitioning, assignment, layout, and simulation routines in support of current card manufacture. The truly new aspect is that major improvements in test capability, design accuracy and turn-around are essential. The problem of specifying chip test is considerably more complex than in the test philosophy used in earlier equipment where all price parts are individually tested prior to assembly. Although important theoretical formulations of the problem have been reported, practical results in determining test procedures are only slowly appearing, and the early work suggests that the determination procedure may in many cases exceed the gate cost in moderate production runs. A major increase in logic simulation with particular attention to dynamic conditions will be required at the chip level to insure correctness of design and minimize the need for chip redesign.⁶ The change problem is particularly perplexing and is lacking in good proposals for its solution. One is reviewed in a later section. If a chip change is required sufficiently early in an LSI machine project, redesigned costs will be incurred although slippage may not occur. During machine test, however, where practice is to install a fix within minutes if possible, slippage as well as redesign costs will accrue; a string of repeated change cycles would be intolerable.

One technique which should contribute in some measure to the solution of the above problems lies in the use of master chips which are identical in all stages of fabrication except for the final metallization step in which the required logic function is defined. As with most general purpose

schemes there is a loss; in this case, it is estimated that a master chip has about one-half the area efficiency of a highly tailored chip at the 50-100 gate level, with efficiency decreasing at higher levels of integration.⁷ Alternately, it is possible to conserve master chip area at the expense of additional wiring layers. In view of the current yield at two layers, it appears that inefficient area usage is the better course of action for the near future. The master chip concept tends to become less effective in high-performance systems (where it is needed most) because of the difficulty in devising an efficient standard allocation of silicon area between the large line drivers and small lower dissipation gates internal to the chip.

Direct replacement

The concept of direct replacement of circuits on cards of late third generation systems with LSI raises a number of interesting if not altogether defensible issues. (Modest improvements in partitioning and mechanical packaging are assumed to be incorporated wherever possible.) The rationale for such an approach might be as follows: (a) Introduction of LSI into an existing and proved logic structure would achieve an increase in raw performance and eliminate many elements of cost discussed previously (as well as many unanticipated ones); (b) Maximizing gates per pad has or will have already been considered in third generation printed circuit card packaging; although improvements can undoubtedly be made, it is not at all clear that one more attempt at partitioning (approximately the same logic) will produce vastly better results for LSI; (c) A large number of existing or anticipated third generation cards have on the order of 50-60 pads or less and approximately a 1:1 gate-to-pad ratio; although the former is beyond current practice, it is not out of range of existing technology; (d) The pad spacing requirements on the periphery of a chip with a technology such as beam-leads is not incommensurate with the silicon area needed to contain about 50 gates.

The rationale is obviously not flawless. The pressure of continuing evolution in system enhancements may not permit a major hardware change without organizational improvements. Perhaps a far superior partitioning exists with a major logic reconfiguration. There is a good chance that the restructuring would result in more gates rather than fewer pads. If so, would 200 gates

and 80 pins, say, really be an improvement? How many years longer would it take to achieve such a chip and fully engineer a system? Proposed schemes for revising logic structure for the sole purpose of accommodating the unique properties of LSI should be carefully examined to determine precisely what tradeoffs are involved. It is important to assess not only the increase in silicon area, dissipation, and reliability in evaluating such proposals, but also the affect on throughput in high-performance systems which may be lowered due to a possible increase in the number of gates in the signal path and an increase in fan requirements.

Even if so-called direct replacement were the best strategy, it still may not be economically justifiable. If this proves true, it is tempting to believe that the timing of the move to LSI logic will be considerably delayed. It would then appear reasonable first to go through a generation of LSI memory in view of its comparative simplicity and distinction of being well-matched to the application and competitive economics. During this period, the technology could be shaken down, and the small chip alternatives to LSI exploited.

System enhancement applications

There are two categories into which the first major wave of LSI applications to computer system design might fall: Quasi-replacement applications, where LSI essentially replaces structures appearing in third-generation equipment (discussed earlier) and enhancement applications, where the special properties of LSI are exploited to build structures which would permit general-purpose computers to have greater capability than what exists in the third generation.

Let us now discuss the enhancement applications to general-purpose computer systems, particularly to their processor-memory subsystems. In order to evaluate the proposals which have been made, we shall enumerate some of those proposals which seem to be possible in the time frame under consideration, and then we shall evaluate those proposals according to specific user-oriented criteria which we shall make explicit. Here are the four broad classes of enhancement applications which we shall consider.

- A. Design processors around large scratchpad memories. There are two functionally distinct cases here, differing in the way the scratchpad is addressed.

- A1. The scratchpad is conventionally addressed in the processor instruction.
- A2. The content-addressed scratchpad is not available to the user but enhances the apparent speed of the working ("main-frame") store by intercepting frequently occurring working-store references.
- B. Design a processor-memory system around a backing store built of high-density semiconductor shift registers.⁸
- C. Use microprogramming techniques based on LSI both in the logic-control ("read-only") store and in the memory-driven logic. Two different modes of use have been proposed.
 - C1. The content of the logic-control store is fixed, as far as the user is concerned.
 - C2. The logic-control store can be written on command, thus permitting the dynamic definition of system operations by the user.
- D. Any assumed cost advantage of LSI can be used not in reducing system cost but in providing more logic for the same or somewhat more cost.^{6,9}
 - D1. Features now found in software can be built into hardware, hopefully improving total performance and reducing user programming cost.
 - D2. Total performance and reliability can be enhanced by replication of subsystems and concurrency of operation.

Evaluation criteria

The consistent evaluation of these proposals requires some criteria related to the economic interests of the user. There are three principal criteria enumerated below in their historical order of discovery. All three criteria are important in almost every system, but with different weights depending on its intended applications.

- A. Throughput—the rate of flow of useful information through the system.
- B. Inquiry response time—the elapsed time from a well-defined request to an adequate reply to that request.
- C. User's manpower efficiency—the response time and rate of productivity of the user's programming and analysis personnel in putting new applications on the system and modifying existing applications.

These criteria are not well defined, and of them-

selves they cannot be used to evaluate directly a particular technological proposal. They do lead to the more specific criteria, enumerated below, which can be used to evaluate the LSI proposals given above.

- A1. Processor instruction or data-manipulation speed.
- A2. Input-output rates and the amount of concurrency among input-output channels and between input-output and processing.
- B. Multiprogram switch time, i.e., the elapsed time, after a decision is made to switch a processor from instruction stream (or program) X to instruction stream (or program) Y, between the making of this decision and the first executed instruction of Y.
- C. Efficiency of translation, diagnosis, and execution of programs written in the common high-level languages.

These criteria are still not well defined in an absolute sense, but they are sufficient for qualitative comparison of proposed schemes to each other and to present techniques. Again their weights are different in different applications, but since we are concerned with the design of equipment to serve in many environments, we shall reject those proposals which improve performance according to one criterion only at the obvious expense of another.

Now, how do the new capabilities allegedly made possible by LSI fare according to these criteria?

LSI scratchpads

Conventionally addressed scratchpad stores force a tradeoff between instruction execution time and multiprogram switch time, the two major components of total processor time. For a given application and processor type there is a minimum number S_m of scratchpad words (index and arithmetic items) such that if fewer are used, processor speed is severely degraded. Experience suggests that this number is between one and sixteen and is often less than eight. Experience also suggests that above S_m the increase in instruction speed due to increments in scratchpad usage rapidly falls to near zero. On the other hand, that part of a conventionally addressed scratchpad associated with the user program must be fully dumped to working store and then reloaded every time the processor switches between

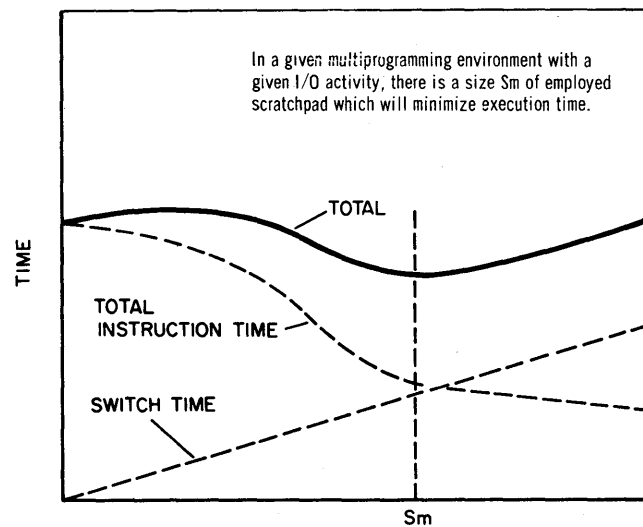


FIGURE 1—Instruction time vs. switch time as a function of scratchpad size

two user programs. (The need to dump all registers whether or not they were altered arises because the executive system has limited knowledge about scratchpad usage and is generally forced to make a worst-case assumption—namely that every scratchpad register is in use.) Figure 1 shows the instruction time vs. switch time tradeoff as a function of scratchpad size.

Now, it may be argued that LSI offers the opportunity to build a scratchpad large enough to hold simultaneously S_m words for every program in working store, thus reducing the dump-reload time to zero. This argument is valid only if it can be assured that the scratchpad is large enough to handle the worst-case maximum number of user programs in working store. If this cannot be assured, then there is a cost of making a decision (whether to perform the swap) which must be added to the switch time whether or not the swap occurs. In the design of systems for general use such assurance is difficult to provide (a consequence of Murphy's Law: "If anything can go wrong it will.") and the whole rationale for a large conventionally addressed scratchpad falls of its own weight. (This argument does not hold if the machine is designed for a specific family of applications, a case we are not considering.)

On the other hand, a "transparent" scratchpad (one whose addressing is not available to the programmer) appears to offer a genuine advantage. If the scratchpad is content-addressable by virtual address so that it serves automatically to intercept references to working-store cells whose con-

tents are duplicated in the scratchpad, and if there is built into the logic of this scratchpad a purge strategy which assures that its content is approximately related to the set of working store addresses currently in greatest use, then there is no necessity to make any worst-case assumptions about scratchpad usage, and the dump-reload cost of a program switch will adapt to actual need, independent of scratchpad size. (This is a resurrection of a concept which appeared in the Atlas computer.¹⁰ Furthermore, if this scratchpad is modularly expandable, decisions about the size to buy are principally related only to performance desired as a function of switching frequency, and do not run afoul of Murphy's Law. Thus, if LSI makes a transparent scratchpad possible, this appears to offer a substantial opportunity to systems designers.

The above comments about Murphy's Law are not to be taken lightly. The decision-making process in the design of general-purpose systems frequently requires the making of many worst-case assumptions with full knowledge that in almost all cases they are excessive. Every time a worst-case assumption can be avoided an opportunity exists to better match a general-purpose system to any given application.

LSI backing store

It was shown as long ago as UNIVAC® * I that data transfer of the full contents of two synchronized delay lines of equal size can take place with zero latency. This principle is equally valid to data transfers between tapped shift registers and random-access memories, and suggests a system organization based on the interesting properties of a large array of semiconductor shift registers employed as a backing store. Such an organization may be quite promising in transaction-oriented systems, in which the interface between working and backing store is a classical bottleneck.

Microprogramming

Microprogramming has already proven its value in the computer marketplace as a way of selling equipment to users with an investment in programs not designed for that equipment. It is fallacious, however, to base upon this observation

the conclusion that the deliberate use of microprogramming to expand the user's set of languages is good for the user. Quite to the contrary. The establishment of programming standards is a matter of extreme importance to users; language and data standardization is, the claims of zealous language designers notwithstanding, the single most important technique available for improving the efficiency of user manpower. Even the present commercial use of microprogramming is in the service of ultimate standardization, not diversity, even though the short-term consequences appear to be to the contrary.

Those proposals for alterable microprogramming, wherein the processor instruction repertoire is dynamically redefinable, may sound appealing, but the auditor of such proposals should ask himself: "What advantage is claimed?" It appears that no functional advantages have been proposed over what is available with today's better software, but some hope is held out that common processes will run faster when microprogrammed.

As soon as the user is given the opportunity to alter his logic-control store, Murphy's Law requires the general-purpose system designer to assume that each user will do so . . . differently. In a multiprogram environment, then, the program switch time must account for the cost of dumping and loading (or, alternatively, disengaging and re-engaging) logic-control stores appropriate to the old and new programs. All the arguments cited above with respect to swapping conventionally addressed scratchpad apply in this case, also, but the situation is aggravated since many more bits of storage are involved in microprogramming than in the conventional use of scratchpad registers for indexing and arithmetic.

The problems associated with designing and debugging a large, complex chip have already been discussed here. The design of computers is also characterized by the need for frequent logic revisions. Clearly, the introduction of LSI technology into the computer design cycle can be expected to aggravate seriously the normal problems of the procurement relationship between the semiconductor supplier and the computer designer. Microprogramming, when viewed as a technique aimed principally at simplifying this relationship, may be a genuine boon, because it isolates the most probable logic changes into a very regular, (hopefully) easily redesigned component: the logic-control store. Furthermore, the processor which

* Registered trademark of Sperry Rand Corporation

executes the microprogram is more suitable for LSI than a conventional processor because it is a more regular structure due to its unusually simple control logic. The computer user won't be concerned with this application of microprogramming, but nevertheless it bears an important relationship to LSI. Note, however, that LSI is not clearly destined to replace the magnetic or capacitive stack in the logic control store. A truly non-volatile read-only LSI storage array based on existing technology presently has an inordinate turnaround time for alterations, and there has been no economic incentive as yet to provide on-site facilities for the computer manufacturer to implement quick changes. Recently reported discoveries of new non-volatile storage mechanisms in MOS-like devices may provide an eventual basis for the implementation of alterable read-only stores with semiconductors.¹¹

Replicated subsystems

The use of replicated subsystems to improve performance falls into two classes. The "network processor" of the ILLIAC IV type¹² acquires, for a given task, a topological configuration in which the position of a processing subsystem in the topological structure bears a definite relation to the subfunction to be performed. Network processors are interesting, possibly even promising, but are beyond the scope of this paper because they are not expected to offer broad enough utility in the time period under consideration to be called general-purpose.

The other approach to replication may be called the "multiprocessor."^{13,14} In the multiprocessor replicated subsystems of the same type are interchangeably available for assignment to subfunctions. Thus a multiprocessor with replication of every distinct type of subsystems has the potential for greater system reliability than is conventionally possible.

LSI may contribute to multiprocessor design where structure is periodic at a low level. This may occur in two places: working store modules (which must be smaller than is considered economical today to permit many concurrent accesses to the total working store), and switching matrix modules. The multiprocessor approach to system organization cannot normally be justified on the basis of cost/performance, but it can on the basis of total system performance or reliability. To the extent that the multiprocessor organization is jus-

tified, the application of LSI to this organization is promising.

The multiprocessor switching-matrix application of LSI has some additional interest because the matrix contains within it a distributed content-addressable memory. The use of an associative store in this way is, in fact, a generalization of the proposed use of associative storage in memory paging.¹⁴ This appears to be the only defensible application of associative storage in a general-purpose computer which has so far been proposed, and it forms an excellent match to the unique capabilities of LSI.

LSI has an inherent functional advantage over magnetics in associative applications, namely that fast bit-parallel searches can be achieved. The main drawback of magnetic associative memories, even in those applications which require relatively simple match-logic per word, is that imperfect cancellation of analog sense signals and other noise effects give rise to a low signal-to-noise ratio and thereby limit the technology to essentially bit-serial operation. Thus, the more nearly binary signals available from semiconductor associative devices seem to provide a unique advantage over magnetics which is not strongly evident in comparisons of the two technologies over other categories of memory.

Hardware-software tradeoffs

The incorporation of software capabilities into hardware is a subject difficult to discuss specifically because of the shortage of concrete proposals, but a few general statements can be made. When someone proposes a hardware-software tradeoff, he usually has one of three things in mind.

- A. The direct execution of high-level languages by processors.^{15,16,17}
- B. The handling of many small frequently executed tasks which are now done by software but which are obvious candidates for hardware, for example, parallel limit checking of subscripts and stack and list pointers, locating the source of an interrupt, priority task allocation, and processing of certain interrupts.¹⁸
- C. The incorporation of a major portion of software (usually the system executive) into hardware.

First, let us observe that none of these ideas is particularly well matched to the features of LSI.

It is likely that if any of these ideas has validity, it could have been justified in third generation equipment; if this has not happened the reasons may be more related to organizational inertia than any merits of the idea.¹⁹ Let us briefly attempt, however, to examine these merits.

Direct execution machines are interesting because they apparently eliminate a translation stage which costs machine time. It is doubtful, however, that this property alone offers any important advantage over state-of-the-art software on conventional hardware. The important characteristics of a high-level language which offer cost/performance advantages to a processor which is modeled after the language are related not to the superficial characteristics of the language (e.g., alphanumeric representation) but to its deeper, structural properties, such as the way it separates data description from procedure description, the way it permits segmentation of programs, and how long it permits delaying the assignment of values to certain variables, such as array size. In the United States, with the notable exception of one manufacturer, most people who have attempted to achieve a cost performance gain by modeling a machine after a language have been scrutinizing the wrong end of the horse.

There is still, however, a good reason for modeling machines after the more superficial characteristics of languages, and because it affects the user's manpower costs it may turn out to be a very important reason. This is the simplifying effect which executing a program in its original form has on the human processes of program modification and documentation. It can be expected that a commercial system which is explicitly designed to simplify the processes of program checkout, integration, documentation, and modification will be a very attractive commodity.

As for the numerous small software tasks which might go into hardware, it is likely that some of them offer performance advantages, possibly justifying their cost even with today's hardware. The point here is that these proposals, and others, can be considered on their own merits and should not be confused with the question of what to do with LSI.

The incorporation of a major portion of system software into LSI hardware has very little to recommend it. The principal objection to the idea is that it aggravates one of the greatest weaknesses of LSI: the high cost of design changes. A

major piece of system software is a more complicated object than most computers, and it is usually not debugged before its host computer becomes obsolete. It has been estimated by a software executive of a major computer manufacturer that as much money is spent on "maintaining" a major piece of systems software after it is initially released as is spent in its development before that point. The re-creation of a physical program realization is not entirely straightforward even when the change involves only the rewriting of a magnetic tape; it would be intolerable if each new revision level implied layout and manufacture of one or more distinct semiconductor arrays.

It has been suggested that this objection can be overcome by putting the software in a form of read-only store. Such a store might not even be a semiconductor array, as was remarked above in connection with microprogramming. This arrangement might speed up the execution of software, although the benefit is questionable on both complex processors (which already have enough concurrency that instruction fetching is not a speed limitation) and simple processors (where the elimination of instruction fetching might save some fraction of execution time but the cost of this approach might be unacceptable). If the program in the read-only store were a microprogram²⁰ instead of the ordinary "machine language" of today's software, a speed improvement is more likely, but the problems of increased cost of ownership (see below) are aggravated.

Executing software in conventional machine language from a rewritable working store has more to recommend it than can be accounted for purely by historical accident. There is a cost associated with owning and storing the realization of a large program, particularly when it is continually undergoing change, and the virtue of storing the parts of a quiescent program on a relatively low-cost magnetic surface and writing them into working store only for execution is that the relatively higher cost of the working store can be shared with other programs. This sharing is not possible if the entire software must always reside in the high-speed store from which it is to be executed.

Note that the no-write character of a read-only store does not necessarily substantially reduce its cost with respect to read-write stores, and there is also to be considered the cost of replacing the storage medium at each revision. The algorithms

of software can be more compactly expressed in conventional machine language than in a logic-control microlanguage, so the cost of ownership of software in such a microlanguage would be even higher. There is a tradeoff, then, between the increased ownership cost and the possibly decreased execution time of microprogrammed software in read-only store; we know of no serious attempt to investigate this tradeoff. It appears that there may exist a useful compromise in which only the frequently used functions of a compiler object program are microprogrammed.²¹

SUMMARY

An appraisal of the technological, economic, and system application aspects of Large Scale Integration has been offered. In logic, the chief competition for the large complex chip is the small simple chip. In memory, large chips are better than small chips but not yet superior to magnetics in the volume areas. The change problem has no solution short of attainment of a level of design automation far above that now existent. Approximate replacement of existing stable and proved third generation circuit partitions with large chips would be the simplest way of introducing LSI; the procedure, however, lacks a clear economic basis at the moment. The attempt to achieve major processor restructuring to obtain more favorable partitions with higher gate/pad ratios is a current activity of significance. If restructuring entails large chips, the entry of LSI into processors will be considerably delayed; if it provides nothing more than a mere accommodation for LSI, inertia and economics will rise in opposition. Initial system enhancement applications are limited in number. Transparent scratchpads appear to be advantageous. *Fixed* microprogramming, rather than as an aid in proliferating languages, can be used to alleviate the change problem and further serve as the basis for a restructuring favorable to LSI partitioning requirements. Multiprocessor working store switching matrices constitute an important application area, and the size and performance characteristics of the working store module itself are moving in a direction in which LSI appears inherently competitive. The use of LSI to build large scratchpads may prove to be economically justified on a replacement basis but not on an architectural enhancement basis. The notion of putting more software into hardware is generally found to be ill-defined; this may be

a major area of opportunity but it is only weakly related to any advantage LSI might offer. Clearly the hurdles for LSI are high. Perhaps, as one sage has said, it will be used "only because it's there."

REFERENCES

- 1 R L PETRITZ
Technological foundations and future directions of large scale integrated electronics
Proc AFIPS Fall Joint Conf pp 65-87 1966
- 2 R L PETRITZ
Current status of large scale integration technology
Proc AFIPS Fall Joint Computer Conf pp 65-85 1967
- 3 B T MURPHY G J GLINSKI
Transistor-transistor logic with high packing density and optimum performance at high inverse gain
Proc Int'l Solid-State Circuits Conf p 38 1968
- 4 P J McGOVERN editor
EDP Industry Report January 26 1968
- 5 P R LOW
Large scale integration and small computers
IEEE Int'l Convention Digest p 81 1968
- 6 M G SMITH W A NOTZ
Large scale integration from the users point of view
Proc AFIPS Fall Joint Computer Conf pp 87-94 1967
- 7 W A NOTZ et al
Large scale integration: Benefiting the system designer
Electronics pp 130-141 February 1967
- 8 C G THORNTON
LSI subsystems assembled by the silicon wafer-chip technique
Proc Int'l Solid-State Circuits Conf pp 42-43 1968
- 9 R RICE
Impact of arrays on digital systems
Proc Int'l Solid-State Circuits Conf 1967
- 10 C H DEVONALD J A FOTHERINGHAM
The atlas computer
Datamation Vol 7 5 pp 23-27 May 1961
- 11 H A R WEGENER et al
The variable threshold transistor, a new electrically alterable non-destructive read-only storage device
IEEE Int'l Electron Dev Conf p 70 1967
- 12 D L SLOTNICK
Achieving large computing capabilities through an array computer
AFIPS Conf Proc Vol 30 April 1967
- 13 J P ANDERSON et al
D825—A multiple computer system for command and control
AFIPS Conf Proc Vol 22 December 1962
- 14 M E CONWAY
A multiprocessor system design
AFIPS Conf Proc Vol 24 November 1963
- 15 J P ANDERSON
Better processing through better architecture
Datamation Vol 13 8 August 1967
- 16 T R BASHKOW et al
System design of a FORTRAN machine
IEEE Transactions on Electronic Computers Vol EC-16 4 August 1967
- 17 A J MELBOURNE J M PUGMIRE
A small computer for the direct processing of FORTRAN statements
The Computer Journal Vol 8 pp 24-28 1965
- 18 I FLORES

Hardware features that aid programming
Computer Design Vol 6 1 pp 48-54 January 1967
19 M E CONWAY
How do committees invent?
Datamation Vol 14 4 pp 28-31 April 1968
20 A OPLER

Fourth-generation software
Datamation Vol 13 1 pp 22-24 January 1967
21 R BARBIERI J MORRISSEY
Computer compiler organization studies
Office of Aerospace Research Report No AFCRL-67-0483
pp 83-86 May 15 1967 Clearinghouse Number AD-658196

High speed modular multiplier and digital filter for LSI development

by DON F. CALHOUN

Hughes Aircraft Company
Culver City, California

INTRODUCTION

In order to realize the increased economy and reliability of systems implemented in large chip or full wafer LSI, five requirements must be satisfied:

1. Systems must be organized and partitioned to obtain a high gate-to-logic pin ratio in order to maximize the use of wafer components.
2. Efficient use must be made of standard logic cells more complex than current IC chips.
3. Logic cells must be defined to both facilitate automated routing and to allow automated testing with a restricted number of test points.
4. Discretionary interconnect of logic elements must be eliminated or minimized.
5. Sufficient redundancy must be used to insure reliability, facilitate testing, and allow economical interconnect in view of non-100 percent yields.

An efficient and original scheme for modularizing a general purpose high speed digital filter is described in this paper. The prime components of this filter are the Modular Carry Advance Multiplier and Adder which meet the first three requirements above extremely well. In a future paper, a technique will be presented which offers a unique means of economically obtaining a specified standard pattern of good circuits on either an LSI chip or an entire wafer. The use of this technique for LSI non-discretionary signal routing helps to meet requirements (4) and (5) by standardizing all

signal interconnect and facilitating automated testing.

High speed modular multiplication for LSI development

The Modular Carry Advance Multiplier uses an original algorithm and implementation to provide very high speed multiplication for any wordlength operands. The technique is most important to any digital processing requirement such as Fast Fourier transforms, pulse compression, digital filtering, or even general purpose computers which require very high speed multiplication. The highly efficient modular logic characteristics and the use of a single circuit type (i.e., a gated full adder, NAND gate, or NOR gate) make it a system very applicable to efficient LSI implementation. Thus, the Modular Multiplier is both an excellent vehicle for the development of LSI processing, testing, and packaging technologies, and is, in itself, an important instrument with which to meet the current and future high speed digital processing requirements.

Emphasis in the description of the multiplier is placed on the modular design techniques which provide the very important characteristics for LSI development of efficient standard modules that: (1) obtain a very high gate-to-pin ratio of 43 for an 8-bit multiplication, (2) require only one circuit type, (3) are readily diagnosable, and (4) allow a one-time logic design that does not require temporary storage, control, or shift logic for any wordlength. Finally, the organization of an entire digital filter is described which meets the LSI

criteria and uses the Modular Multiplier as the key component in its high speed design.

Design of the modular carry advance multiplier

There are two original concepts in the design of the Modular Carry Advance Multiplier. First, the most general multiplication technique of summing the columns of a matrix formed by ANDing and shifting the multiplier-multiplicand bit pairs (as shown in Figure 1 for an 8-bit multiplication) has been analyzed to determine independent and modular blocks of logic which allow partial products to be determined in parallel and then summed to form the final product. The most efficient division of the multiplication matrix into modular logic blocks is accomplished by forming geometrically similar blocks within the matrix as shown by the dashed lines where an 8-bit multiplication is partitioned into four identical 4-bit multiplier blocks (MB1-MB4). Since these four logic blocks are independent, each can function in parallel to determine 8-bit partial products by the column summation of the bits in its portion of the large products in a three-input adder is the final product.

Since the four segments of the multiplication matrix are reduced at once and since the multiplication matrix itself has pre-determined the shifts required, no control, temporary storage, or shift logic is required. By this simplification of the column summations into shorter, parallel operations without control logic, a significant speed advantage is obtained. In addition, a modular multiplier block has been defined which can be used to build any longer wordlength multiplication by paralleling modules and summing their outputs.

The bit length of the multiplier module can be determined by the yields of different complexity LSI chips and by the efficiency obtained for various multiplication wordlengths. In any case, the modular multiplier reduces the maximum multiplication delay from N full wordlength additions and shifts for the classical one-bit-at-a-time technique to that of only one double-wordlength addition. Since most of this delay occurs in the adder which sums the partial products, look-ahead or other speed-up techniques in the adder will further improve the speed beyond that reported herein.

The second original technique in the modular multiplier is the asynchronous carry advance philosophy that is used to implement with a single circuit type the multi-operand addition required both in the multiplier modules and in the final adder. The carry advance implementation performs the simultaneous addition of multiple operands with any carries being asynchronously advanced toward the output rather than being "saved" or propagated in a ripple-carry fashion. The logic signals with the shortest delay paths are reduced first, thus adding less delay to the most critical paths.

In general, if K-bit modular building blocks are used, $(N/K)^2$ of the blocks allow the multiplication of two N-bit (magnitude) operands, where N is unrestricted. These operate in parallel to form partial products which are then reduced to the final product by a $(2N-K)$ bit adder. In general, an (N by N) bit multiplication with register storage of the product will require 6 SUM and $(2N-3)$ CARRY delays. A hybrid chip-and-wire MSI multiplier module (Figure 2) and an integrated circuit breadboard (Figure 3) which form 8-bit products have been built and tests (see Figure 4) show a 12-MHz multiplication rate using 12 of the relatively slow SN5480 gated full adders. The hybrid MSI multiplier was fabricated on a 1-inch square ceramic substrate. The introduction of a 6-bit latch register would allow the 12-MHz rate to be nearly doubled via pipelining.

As an example of paralleling identical multiplier blocks to build longer wordlength multiplications, consider the multiplication of two 8-bit operands as shown in Figure 5. As for the general case in Figure 1, the four 4-bit multipliers (MB1-MB4) are defined by forming the smaller, geometrically similar matrices, each of which determines the product of four of the multiplier bits

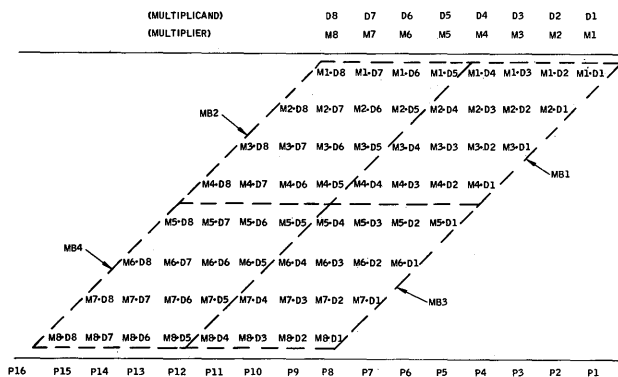


FIGURE 1—Four modular 4-bit multipliers for the multiplication of two 8-bit operands

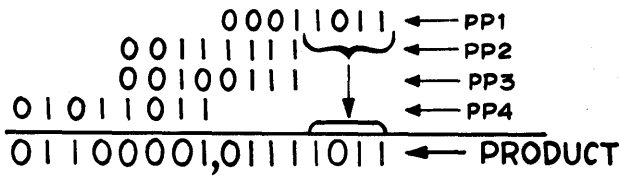


Figure 7—Adder inputs for multiplication of Figure 6

gered in this fashion, the adder need only be a three-input 12-bit adder in order to determine a 16-bit product. This can be reduced to a 10-bit delay by doing a simple look-ahead in the last three bits which does not require additional circuits if gated full adders are used. And, since the multiplier block delay is equivalent to a 6-bit addition, the longest delay to multiply two 8-bit numbers is equivalent to a double wordlength (i.e., 16-bit) addition.

Figure 8 shows the logic required in each of the identical 4-bit multiplier blocks with the inputs being ANDed multiplier-multiplicand bits if non-gated full adders are used. The longest delay is from the SUM (S) output of the first FA3 or from the CARRY (C) output of FA2 to the S output of FA7. If gated full adders are used as the logic blocks, only the four multiplier and four multiplicand bits are necessary at the left as inputs. That is, the gated full adders can generate the required "AND" of the multiplier-multiplicand bit pairs as well as determine the SUM and CARRY outputs. Otherwise, the ANDing function of the multiplier-multiplicand bit pairs must be generated external to the full adder blocks. The design that has been breadboarded and tested requires only 12 SN5480 gated full adders per 4-bit multiplier block. Thus, to multiply two 8-bit operands, 48 full adders are needed for the modular multiplier blocks, 19 for the adder, 1 as a sign determining circuit, and 34 gated full adders to implement a 17-bit product register. This totals only 102 gated full adder cir-

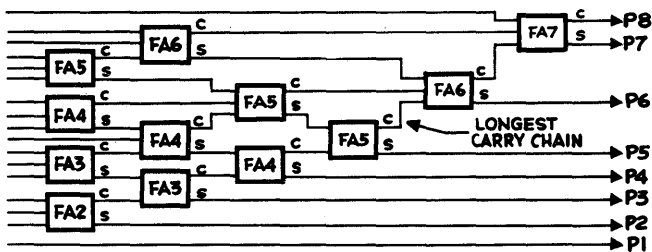


FIGURE 8—Design of 4-bit carry advance multiplier

cuits and is well within the goals of full-wafer LSI technology. Only 38 external connections are required which obtains a very high gate to pin ratio of 43 since each gated full adder is logically equivalent to 16 two-input NAND gates. If 16 additional input pins and 12 full adder circuits are added, a 16-bit number can be added to the product which represents all the arithmetic processing necessary to accumulate the finite sum of products required in linear digital filtering.

Use of an "Integrated Electronic Component" as the basic logic block

All logic functions in the multiplier, adder, and register can be accomplished with NAND gates, NOR gates, or with gated full adders (e.g., the Texas Instruments SN5480 "Integrated Electronic Component"). This gated full adder chip (shown logically in Figure 9) is used efficiently in different modes to obtain the functions of a full adder, half adder, NAND gate, single-input inverter, and flip-flop (by cross-coupling the input gates). In addition, conversion to one's or two's complement notation can be accomplished in the adder by appropriate use of full adder input gates. The use of this single circuit type as a logic building block and the modular characteristics of the multiplier are especially important since they provide a sys-

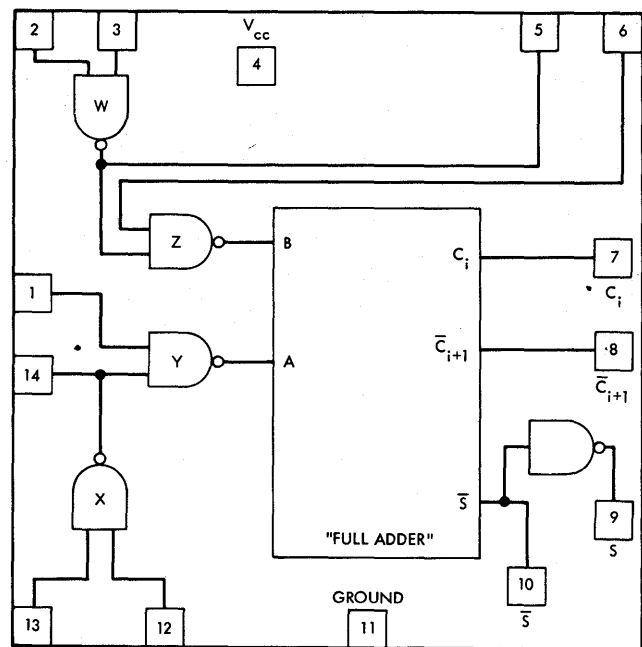


FIGURE 9—SN 5480 full adder with gating and interconnection pads shown

tem very applicable to LSI development which can greatly reduce the cost and volume per circuit of this high-speed design. A further advantage to choosing a gated full adder as the single circuit type is that its logic is equivalent to 16 preconnected NAND gates that greatly reduce the number of circuit chips and interconnections required between circuits. The simplification of interconnect by using more complex logic chips as building blocks in an LSI implementation is especially important.

The design of the modular multiplier with the product register is about 85 percent as efficient (in number of IC chips) as an optimum design using gates and full adders, but at that high an efficiency it is far more desirable for LSI processing techniques than a mixed circuit design since it offers extreme advantages in both the wafer processing and circuit interconnect. Furthermore, the area of an SN5480 circuit is 3600mil² for 16 gates while the area of an SN5400 quad two-input NAND gate is 3000 mil². Thus, the SN5480 has 333 percent more circuits per area than the SN 5400 quad two-input NAND gates. For any application that makes much use of full adder functions, such as the design presented here, a gated full adder is seen to be a much more efficient circuit type. Because it is a complete logic building block and offers the great logic flexibility described above, it may be found to be quite important for other LSI systems.

Summary of modular carry advance multiplication characteristics

The Modular Carry Advance Multiplier has been described having the following characteristics:

1. An efficient, high speed design (30 MHz with MECL II for 8-bit products) is used. For example, a four-times speed increase is obtained over two-bit-at-a-time ternary multiplication using a look-ahead adder.
2. The modular multiplier is implemented with a single circuit type (a gated full adder) of high logic capability which minimizes the unit volume and interconnection.
3. The modular logic design allows any wordlength multiplication to be built by paralleling more identical building blocks.
4. No control, shift, or storage logic is required, thus permitting a maximum gate-to-

pin ratio (43 for an 8-bit multiplier).

5. The multiplier is easily tested from inputs and outputs alone by using counters for the inputs and monitoring the outputs of two or more units with a simple comparator. Total test time for entire 1600 gate wafers is about 200 μ s.
6. By using Q additional full adders, a Q-bit operand can be added to the product while it is being formed with only one additional SUM delay. That is, $(AxB)+P$ could be generated with only one additional SUM delay beyond the time required to generate (AxB) . This has a most important application in radar data processing as described later since it entails all the arithmetic processing required in digital filtering, pulse compression, or correlation calculations.
7. Pipelining of a Modular Carry Advance Multiplier can be used to both gain speed and save circuits in long wordlength applications. That is, successive groups of the multiplier and multiplicand bits can be sequenced through a single modular multiplier which forms a partial product that is summed with the sum of previous partial products until all multiplier-multiplicand groups have been multiplied. For example, Figure 10 shows the multiplication of two 32-bit operands by sequencing 8-bit groups of the multiplier and multiplicand through the multiplier 16 times. Thus, sixteen 12-bit additions are required and at 20 ns per stage, $(16 \times 12 \times 20)$ ns = 3.84 μ s is the total delay to form the 64-bit product.
8. The combination of efficient modularity, very high gate to pin ratio, and the fault

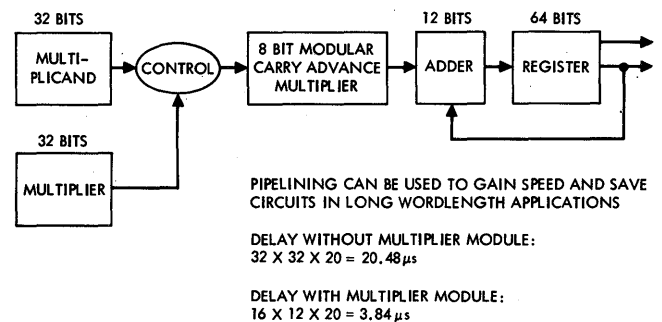


FIGURE 10—Multiplication of two 32-bit operands by sequencing 8-bit groups of multiplier and multiplicand through multiplier 16 times

isolation characteristics allow the Modular Carry Advance Multiplier to obtain the increased reliability and economy of LSI implementation.

Automated diagnosis of an LSI modular multiplier and digital filter

The 8-bit plus sign modular multiplier that has been discussed consists of four 4-bit multiplier modules, a 12-bit adder, and a 17-bit product register. The adder and register can be combined into one 4-bit module type allowing four of these 4-bit modules to perform the partial product addition and storage. Thus, two modular unit types can be defined on an LSI wafer which together comprise the multiplier that develops a 17-bit sign and magnitude product. If the partial products developed by the multiplier blocks are available at test points external to the wafer, access can then be gained to each of the eight modules to test and isolate any failure to a single module. Since only combinatorial logic is used, and there are only 2^8 combinations of inputs to each module, complete, exhaustive testing of the modules is both fast and straightforward. Figure 11 shows the simple test-bed of two 4-bit counters, two 8-bit registers, and an 8-bit comparator that can be used to automatically test the modules of two LSI wafers exhaustively. Each successful comparison clocks the multiplier to its next state and a multiplier carry-out clocks the multiplicand into its next state, thus successively generating all 2^8 combinations of inputs. If a comparison is not made, the FAIL signal is given and the counters

are left in their last state so that a logical diagnosis can, if desired, isolate the fault to one of the 12 to 14 full adders in the bad module. This simple automated test procedure can proceed at speeds of $200 \mu s$ per LSI wafer. Because the Modular Multiplier allows an LSI wafer to be efficiently partitioned into smaller cells that can be readily diagnosed, it is an excellent vehicle with which to develop the full wafer LSI technology with multi-layer interconnect. A fully automated test bed for 4-bit multiplier modules fabricated on a single LSI wafer chip is currently being built at Hughes. Figure 12 shows, in the bordered regions, three of these 4-bit multiplier chips on a $1\frac{1}{2}$ -inch wafer of SN5480 circuits. The twelve circled circuits within each bordered area specify the gated full adders that will be interconnected with two levels of metal to implement the multiplication. These 4-bit multipliers of 200 gate complexity having a quarter of a square inch of wafer area will be separately tested and packaged to provide an important step toward processing a full wafer of 1600 gate complexity that will develop and store in a register a 17-bit product.

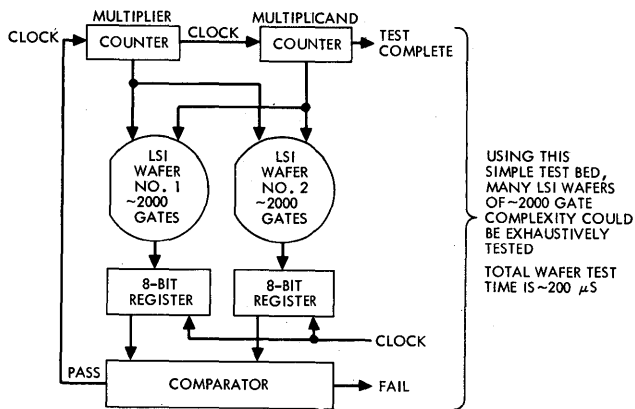


FIGURE 11—Simple test-bed of two 4-bit counters, two 8-bit registers, and 8-bit comparator for automatic testing of modules of two LSI wafers

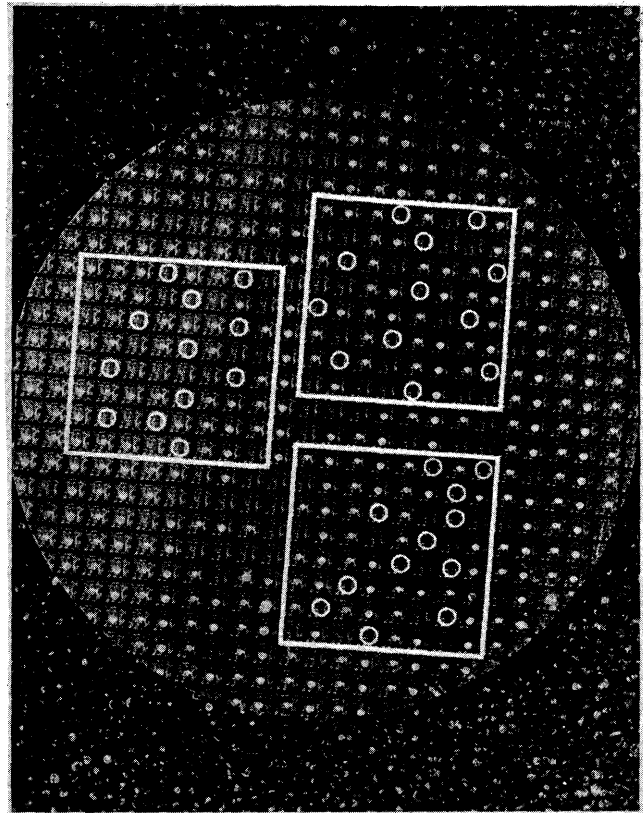


FIGURE 12—Three half-inch square chips of SN 5480's that will be interconnected and packaged as 4-bit modular multipliers

Slight modification of the testing approach will allow the exhaustive testing of an entire digital filter. Figure 13 shows how two registers (A and B) can simulate the input data and two computers (labeled SINE and COSINE) can be used to generate successively the phasor multiplier coefficients for A and B. By clocking the SINE and COSINE counters and the shift register memory, the successive outputs of the shift register can be compared with the expected results until a failure is detected or all the phasor multipliers have been used. In the latter case, another set of A and B inputs can then be used for another test sequence. Since these tests can be made as a free-running and self-stopping comparison (as described for Figure 11), the total exhaustive digital filter test time will be only a few seconds or less.

A high-speed general-purpose digital filter of modular design

As already described, the Modular Carry Advance Multiplier can readily be expanded in word-length by paralleling more of the modular multiplier-adder blocks. The total multiplication delay remains equal to the addition time of two numbers the length of the product, and the multiplication rate can be set simply by the rate at which the product is clocked from the multiplier (up to the maximum multiply rate as determined by the particular circuit type used). The preceding Summary mentioned how, in addition to forming the product of $(A \times B)$, another number could very efficiently be added to this product to form

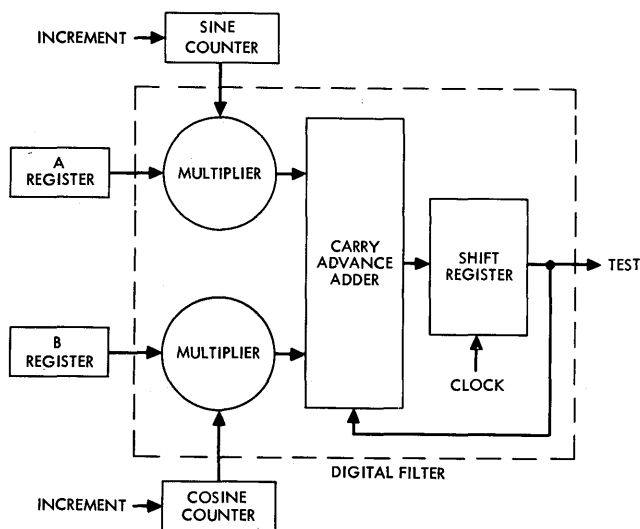


FIGURE 13—Diagnosis of entire digital filter

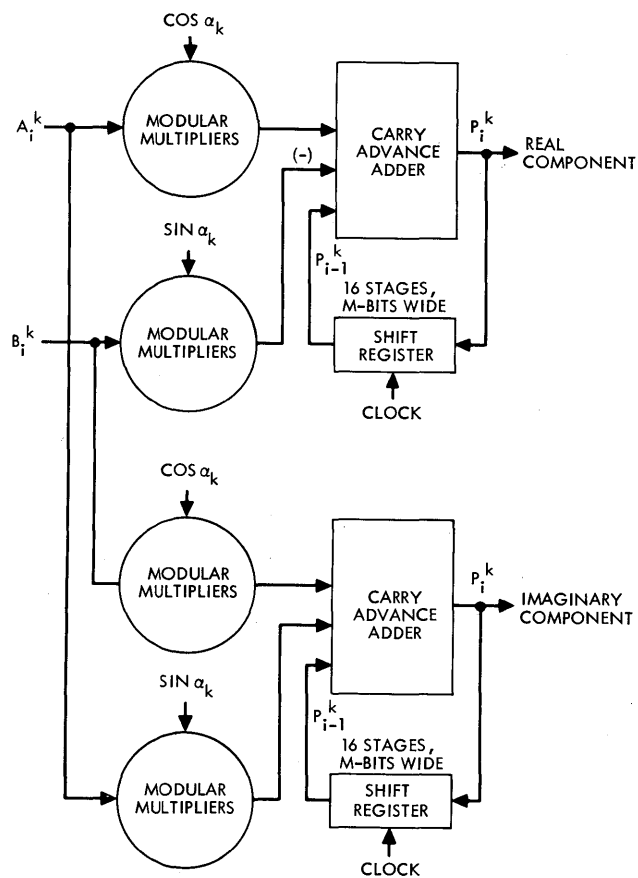


FIGURE 14—Unbuffered digital filter implemented with standard multiplier, adder, and shift register modules

$((A \times B) + P)$. This unique capability to accumulate a sum of products in a highly efficient modular and expandable fashion is most important to the design of digital filter banks which can readily be configured from these standard modules to meet the processing requirements of a wide range of systems.

As an example application, consider a radar filter bank having 16 filters and 32 8-bit pulse returns per filter dump. Figure 14 uses an unbuffered filter bank as an example of how the real and imaginary components of each pulse return can be multiplied by the appropriate filter phase shifts using the modular multiplier blocks. The partial products formed are summed with the previous sum of products for that filter by adding an additional input to the carry advance adder and bringing the previous sum to the adder from a shift register memory. Thus, the adder output is

$$P_i^k = ((A_i \times B_i) + P_{i-1}^k) \tag{1}$$

where

k = the filter number

i = the number of the pulse return

A_i = the i^{th} pulse return (in-phase and quadrature components)

B^k = the complex phasor coefficient associated with the k^{th} filter which determines the digital filter's characteristics of shape and tuning.

The filtering is accomplished by successively incrementing k from 1 through 16 and then incrementing i until $i = 32$ at which time the filtered outputs are dumped.

The use of only a clock pulse to control the multiplication and addition rate allows the filter processing rate to be easily synchronized with the input data. Thus, as long as the maximum processing rate is not exceeded, any further system processing requirements can be accommodated with minimum changes in the current processor. The only changes would possibly be in the clock rate, additional shift register modules, and/or filter modules. More important, however, is the fact that many very different system configurations can use the same standard parts that can greatly decrease the design, fabrication, and maintenance costs of these units. Table I presents a detailed list of the simple differences in configuration of these modules which allows them to meet a wide range of processing requirements.

TABLE I—Modifications necessary to meet variations in processor requirements

Processor Requirement	Modifications to Modular Filter				
	Use Higher Clock Rate	Use Faster Logic Circuits	Add Necessary Multiplier Modules in Parallel	Attach Additional Inputs to the Adder Modules	Lengthen or Widen the Shift Register Accumulator
Additional Filters	Yes	Only if maximum multiply rate is now exceeded	No	No	Lengthen
Additional Range Bins	Yes	Only if maximum multiply rate is now exceeded	No	No	No
Higher Pulse or Input Data Rate	Yes	Only if maximum multiply rate is now exceeded	No	No	No
Increased Wordlength	No	Only if maximum multiply rate is now exceeded	Yes	Yes	Widen

The use of the Modular Carry Advance technique to form sums of products as required in a wide range of digital filtering applications uses, at most, three types of standard modules which can efficiently be configured to satisfy the widely varying requirements of different systems. The same modules are as efficient in systems whose wordlength, input pulse or data rate, number of filters, and number of range bins vary significantly. The three modules that are used to meet these various system requirements are the Modular 4-bit Multiplier, the Carry Advance Adder, and the High-Speed Shift Register. The Modular 4-bit Multiplier and the multi-operand Carry Advance Addition technique have been described in detail above. The Carry Advance Adder modules, however, may be modified with look-ahead logic for higher speed and with additional inputs to allow for expansion in wordlength with minimum system modification. The shift register modules will be 4, 8, or 16-bit serial increments compatible to the processor logic and allowing simple modification in wordlength and/or number of filters.

A wide range of processing rates is determined by various circuit implementations of the Multiplier-Adder modules shown in Table II. If very high speed processing is required, the High Speed Modular Filter implemented with high speed circuits (e.g., MECL II or III) and using look-ahead addition with latch registers for pipelining can provide *complex* multiplication rates in excess of 30 MHz for forming 17-bit sign-and-magnitude products. This would be most important for applications requiring high data pulse rates or a large number of range bins or filters. If these requirements did not exist, a 6 MHz complex multiply rate (equivalent to a 24 MHz "simple" multiply rate) could be obtained by doing serially the four simple multiplies required in a complex multiplication. This use of a single set of modular multiplier blocks (rather than four sets) would allow the package count to drop by at least two to one. The modular filter design allows this clear trade-off between number of modules and processing speed while requiring no new module types when modification is necessary in the wordlength or processing speed (i.e., data pulse rate, number of range bins, or number of filters). Only the Modular Carry Advance Multiplier-Adder algorithm of forming sums of products coupled with a set of shift register modules is known to allow such a wide range of digital filter requirements to be met

efficiently by varying the configuration of only three standard modules. And yet, the definition of the standard modules actually allows a maximum processing efficiency for the number of circuits required since there is a high level of parallelism both in the multiplier and the adder modules.

TABLE II—Maximum delay to form the 16-bit product of two 8-bit numbers using the modular carry advance multiplier

Circuit Type	Sum Delay	Carry Delay	Multiplier Delay, ns	Multiplier Rate, MHz
SN 5480 (II)	65	13	290	3.3
SUHL (Sylvania)	22	10	156	6.4
MECL II (Motorola)	8	8	95	10.5
MECL III (Motorola)	4	4		
SUHL (with speed-up)	(Using Look-ahead and pipelining)	(Using Look-ahead and pipelining)	~50	~20
MECL II (with speed-up)	(Using Look-ahead and pipelining)	(Using Look-ahead and pipelining)	~40	~25
MECL III (with speed-up)	(Using Look-ahead and pipelining)	(Using Look-ahead and pipelining)	~30	~33

Comparison of the modular filter with a current filter processor

Table III presents a comparison of a current digital radar signal processing system with two LSI versions using the Modular Filter. Version I of the Modular Filter system uses 3/4-inch square flat-packs to package chips similar to those of Figure 12 and having typically 200 gates of logic. Version 2 is a full-wafer 1600 gate complexity LSI

implementation requiring only three 2-inch square modules. Even Version I of the Modular Filter decreases the inter-package connections from 10,000 to 424 while reducing the number of packages from 1000 to 24 and the volume from 170 cubic inches to only 9. With this, ninefold gain in processing capabilities is obtained by the combined ability to process over twice as many range bins and to do 16 (rather than 4) unique filter phase shifts. Version 2 provides the potential lower cost and increased reliability of a full-wafer LSI system while obtaining further advantages in speed, volume, and interconnect. The modular partitioning techniques which efficiently define the standard filter modules permit these extreme advantages in physical characteristics, increased processing capability, and standardization, to be realized from the single-chip and full-wafer LSI technologies.

TABLE III—Comparison of a current processor and two versions of the modular filter

	Range Bin Capability	Multiplication Rate	Flat Packs	Volume (with Mounting)	Filter Characteristic	Connections To External Packages	Is Mpy Delay Linear with Wordlength
Current digital filter processor	384	2 MHz	1000	Brass-board 3200 cu in. 170 cu in. final size	4 wired-in filter phase shifts	10,000	No
Version 1 of Modular Filter (Wafer Chip LSI)	800 1040	25 MHz 33 MHz	(MECL III) (MECL III) 24 flat packs	9 cu in.	16 Variable filter phase shifts	424	Yes
Version 2 of Modular Filter (Full-wafer LSI)	1100-1300	35-40 MHz	(MECL III) 3 flat packs	5 cu in.	16 Variable filter phase shifts	145	Yes

ACKNOWLEDGMENTS

The author sincerely wishes to thank Dr. Ira Terris and Messrs. R. F. Stewart, J. M. Block, C. F. Edge, and J. S. Steiner, all of Hughes Aircraft Company, for their helpful counsel and encouragement.

Efficient partitioning for the batch-fabricated fourth generation computer

by N. CSERHALMI, O. LOWENSCHUSS and
B. SCHEFF

Raytheon Company
Bedford, Massachusetts

INTRODUCTION

The computer industry is on the verge of an upheaval, due to drastically new hardware and modern computational concepts. Read-only memories which operate near 0.1 microseconds are available, while large-scale integration (LSI) offers the promise of inexpensive, batch fabricated processing of logic and storage elements. The problem which confronts the computer designer is how to use these elements in an efficient manner to take full advantage of their speed and flexibility. Many approaches have been proposed, but none have shown a clear solution to the problem. The concept presented in this paper is the result of extensive development activities.

The partitioning problem

A study of both general and special purpose digital hardware indicates that certain basic arithmetic and Boolean functions are repeated in various combinations throughout a given computer system. Presently available complex logic arrays such as adders, shift registers, and counters perform many of these tasks, and reduce logic cost, compared to the use of simple gates. At the same time they create a serious logistics problem to the hardware product engineer, because stock and maintenance parts increase. In a rapidly growing technology, where labor is so costly, the design, procurement, fabrication, and test cycles are all lengthened by the needs for many different types of parts. By combining several related operations into a single array, significant improvements in logistics, interconnect ratios, power-speed merit figures, and component logic efficiency are realized.

An array can be fabricated with discretionary wiring methods (slice technology) or with the "cell approach" (chip technology), or with hybrid technology. From the component point of view, bipolar or MOS devices may be used.

The size and the complexity of the array is determined by the logic partitioning. The proper selection of the components and interconnection method is dictated by packaging, power consumption, performance, and economic considerations.

How can one tell whether the partitioning is right? Two numeric criteria provide a measure for any given partitioning design;

- a) *Maximum Gate/Pin Ratio*—This ratio is generally between 0.1 to 0.6 in present integrated circuit (IC) systems. A ratio near 1 is considered outstanding. With the LSI building blocks described below, a ratio of 3 to 10 can be achieved. The requirement for increased gate/pin ratio is a basic reason for the existence of LSI arrays, because interconnections on the microelectronics level are less expensive and more reliable than at the package level.
- b) *Minimum Number of Array Types (Part-numbers)*—This is the best criterion of the logic partitioning, and demonstrates the level of coordination between semiconductor, logic and system designers. Most IC systems employ from 50 to 100 types of printed-circuit modules. Minimum "part-numbers" is one of the primary requirements in all military systems, and is an economics requirement for commercial and low-volume production sys-

tems. Several digital IC systems (including general purpose computers) have been built that contain 3 to 15 types of printed-circuit plug-in boards at 40 or more gates per board.

The penalty paid for maximum gate-pin ratio and minimum part numbers is an inefficiency in the use of silicon (gates) and hence of power. Thus, in "tight" applications such as space vehicles, the designer might prefer to forego the economic advantages of this type of partitioning in favor of minimum power.

Directions in LSI computer architecture

Current developments

With LSI technology progressing toward maturity, more useful documents are being published in the field of LSI system architecture. Most assume the unlimited success of the semiconductor process, but only a few cite documented hardware development. LSI development projects, either completed or near completion, differ in their partitioning concepts; all are designed to replace third generation computers in whole or in part. Some of these development projects are briefly described to supply a background to this paper.

- 1) The sole producer of discretionary wiring (or slice technology), Texas Instruments, has designed and built an LSI airborne computer, the Model 2502. This general-purpose 16-bit word computer consists of a Central Processing Unit (CPU) and a 3-channel Input/Output Unit. The single address CPU is controlled by a 2 MHz clock, and the 36-instruction set is executed through four general purpose registers. (Reference 1)

From the partitioning point of view, the most significant feature is that the CPU is implemented with 16 identical arrays, which include all arithmetic and register functions except for the control logic. The elimination of "special-purpose" registers (index, accumulator, etc.) must also be emphasized. While the CPU is partitioned on a bit basis, the three I/O channels are implemented by six arrays of another type, partitioned on a functional basis. A gate-to-pin ratio of two was achieved on both arrays.

Although automated design was used to assist in the implementation of the control logic, the use of ten different arrays with low gate-to-pin ratios was required. Another disadvantage is that the industry-wide claims of LSI speed improvements do not appear; the computer is three to five times slower than is attainable using currently mass-produced TTL circuits.

- 2) A larger and more powerful general-purpose aerospace computer has been designed (hardware near completion) at Raytheon Company. This parallel 32-bit word computer offers 94 instructions, indefinite chaining of Index and Indirect Addressing, and Multiprocessor capability. The seven full-length registers of the CPU are partitioned into eight identical arrays. Each array contains four stages of all seven registers, including the transfer gate structure. These eight LSI Arrays of one type contain 56 percent of the entire CPU logic.

The control logic is temporarily implemented with 14-lead TTL circuits. Some further work is being done to replace the majority of control with a Read-Only Memory (ROM), leaving only less than five percent non-LSI hardware.

- 3) In another Raytheon development program, a CPU is under construction which utilizes 25 "Raytheon AS-80" arrays. This multi-purpose 4-bit Counter/Register is used as a building block to form the eleven working registers which are 4, 8, or 16 bits long. The block diagram of this ar-

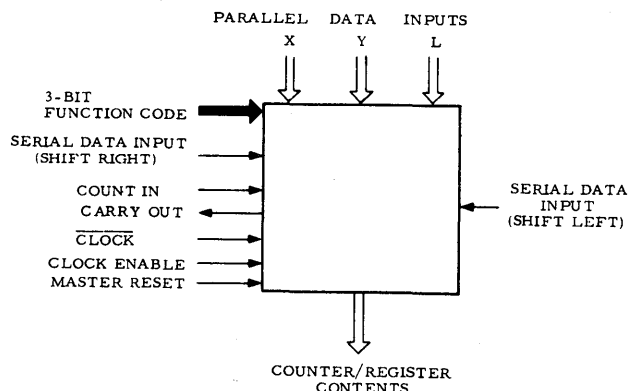


FIGURE 1—Multi-purpose counter/register

ray is shown in Figure 1; it will be described in detail in a later part of this paper.

The processor is a 16-bit, parallel, single-address, high-speed unit. Operating with a solid-state, read/write memory, it can execute an add instruction in less than $\frac{1}{2}$ μ sec, and a 16-bit multiply in less than 2 μ sec average.

From a partitioning point of view, the CPU is organized as follows: 80 percent of the logic is implemented by two types of arrays (Raytheon AS-80 and Signetics 8260) and the remaining 20 percent is implemented by high-speed TTL circuits. The majority of the control logic will be eventually replaced with ROM.

- 4) A classical example of functional partitioning is the LIMAC (Large Integrated Monolithic Array Computer) concept being implemented by RCA. The hardware of this 16-bit machine is grouped into functional execution units, each of which contain the control and the operand registers. A simple centralized control directs the data flow, governed by micro-instructions. (Reference 2)

This concept favors large arrays, but it is not restricted to any specific size. As semiconductor technology improves, more circuits can be included in a package which leads to increased gate-to-pin ratio and decreased unique parts in this particular system organization; e.g., with 100 gate blocks, a gate-to-pin ratio of 2.5 can be achieved; with 1000 gate blocks, a gate-to-pin ratio of better than seven can be realized.

Efficient partitioning with building blocks

In the following paragraphs two building block approaches will be presented. The first utilizes an 80 gate array with 3 to 1 gate-to-pin ratio, the second concept is based on a more complex array with gate-to-pin ratio of ten, and Read-Only Memories.

- 1) *AS-80 Processor*—A basic register array with built-in microprogrammable functions can be fabricated with 4, 8, 16 or

even 32-bit elements. A 4-bit array was designed and fabricated to prove the feasibility of this approach. The block diagram of this Raytheon AS-80 (Patent pending) is shown in Figure 1. Simultaneously, a processing unit was designed around 25 of the AS-80 arrays to demonstrate its usefulness.

The Raytheon AS-80 is an example of a high-speed, programmable, 4-bit register contained in a single 28-lead flat-pack. As shown in Figure 2, eighty NAND gates are interconnected to provide a programmable counter/register, capable of operating in any one of the following eight mutually exclusive modes by entering a 3-bit function code:

- Clear—Also enter data into selected bit positions
- Shift-left—
- Shift-right—
- Load-Enter 4 bit parallel data
- Hold
- Complement
- Count-down—straight-binary counter, decrementing
- Count-up—straight-binary counter, incrementing

The Raytheon Register comprises four flip-flops, each consisting of six NAND gates. Additional logic circuits are included to enable storage elements to be programmed and used as either a storage register, binary counter (up or down), shift register (left or right) and to be complemented (change state).

TABLE 1

	AS-80 Array	Equivalent Logic Built of Standard TTL IC's
Number of Packs	1	28
PC Board Connections	28	292
Wire Bonds	56	584
Clock Interval and Power	60 ns 0.75W	125 ns 1.4W
Speed. Power Product	45	175

Several Boolean operations are performed without the addition of external logic, by combining coded commands with the "L" data input. For example, the execution of both hold and load (L)

yields the logic "OR" function (A+L).

Table 1 compares the AS-80 array with equivalent logic, constructed by means of present-day standard TTL integrated circuits.

Interface with the AS-80 is at standard Transistor-Transistor-Logic (TTL) levels. Input and output characteristics are identical to those published for high level TTL circuits. Gate outputs designed to be used internally on the chip can be connected together to form wired OR functions. The AS-80 array is currently fabricated by Sylvania on a single monolithic "chip" with three levels of metallization for the internal connections. Figure 3 is the microphotograph of an area of a silicon slice where an AS-80 is formed. On the periphery the bonding pads are also visible.

Arithmetic operations require the use of another logic element. A highly complex four-bit array was designed which performs most arithmetic and logic combinations of two operands, and can be implemented with 70 gates and 24 pins. However, it functionally overlaps the AS-80 register array, and its usage is limited; therefore, its building block usefulness is questionable.

Examination of the AS-80 array shows that many of the functions of a general arithmetic unit are included in the array (shift, complements, count, etc.). All that is necessary in the arithmetic unit are the pure add (and therefore subtract) operations associated with a central processor. Selection gating provided at one of the operand inputs of the register array enables convenient implementation of subtraction. In addition to the register and adder arrays, control logic is required to provide synchronization, gating signals, and a sequence of logic instructions in order to be operated as a homogeneous system.

2) Register Array Concept for the Fourth Generation (RACON-4)

Every system previously described had to carry the burden of third generation computer operation, such as specialized hardware, high application programming cost, and excessive processing time for internal scheduling (Reference 3).

Users of third generation systems have found that in the case of leased equipment, the cost of hardware is only about five

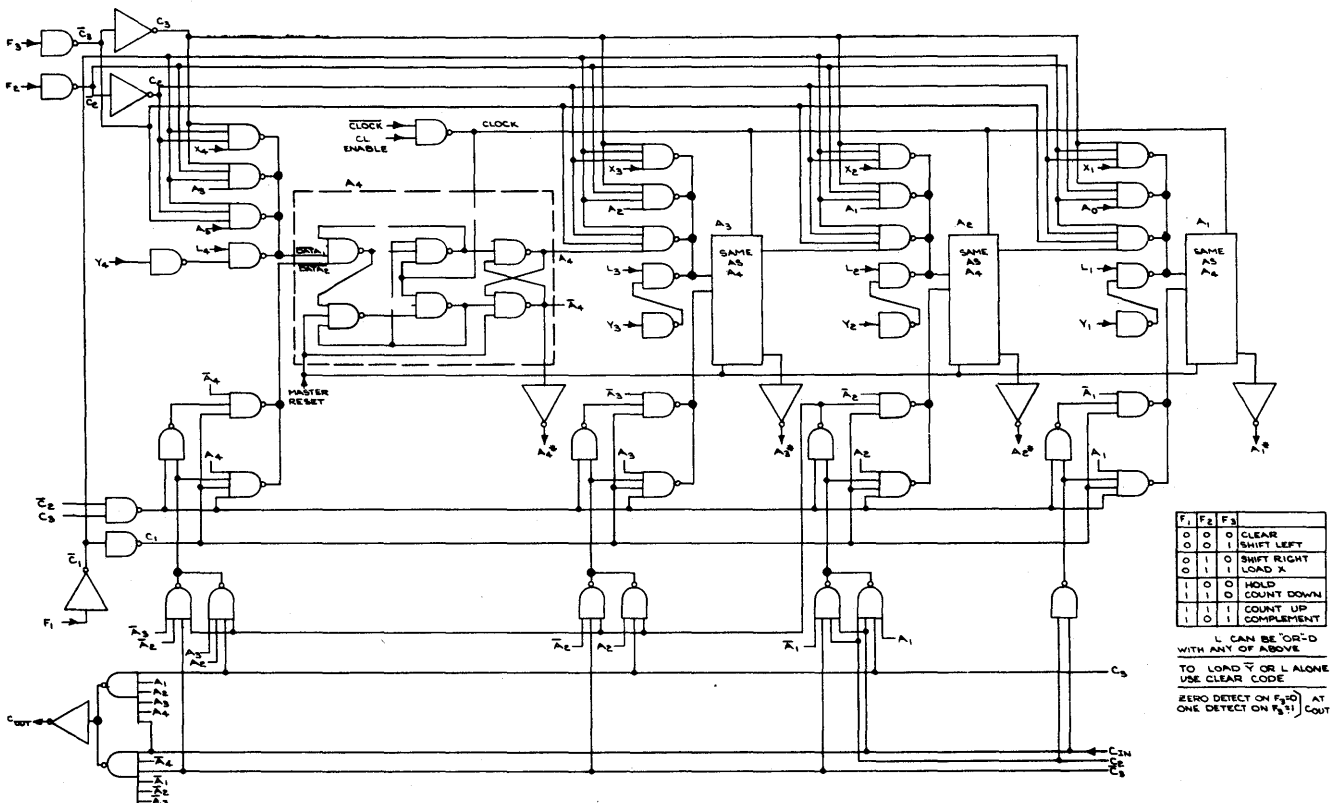


FIGURE 2—Logic schematic of AS-80

percent of the total operating cost. The system electronic circuitry represents roughly one-third of that five percent. Of the remaining 95 percent of operating cost, approximately 35 percent is absorbed by software; 30 percent is required by special environmental facilities, power, and operating costs; and the balance (30 percent) is made up of manufacturer's costs, sales fees, maintenance, and profit. This means that even if LSI technology makes the entire electronics available at no cost it does not represent any significant economies for the user.

In less than one year, the industry will have

mastered a new generation of hardware. However, if this new generation of hardware is used only as a replacement for the hardware of a third generation system, it will have a doubtful future. Only systems that can exhibit reduction of the other 95 percent of cost can be considered advanced systems and candidates for the fourth generation family.

Thinking further ahead, such advanced systems will lead to a new market situation where the cost of the hardware becomes significant and a major factor in a Buy/No-Buy decision.

If we assume that a small number of general-purpose building blocks can be developed to perform all the computer functions, the problems of

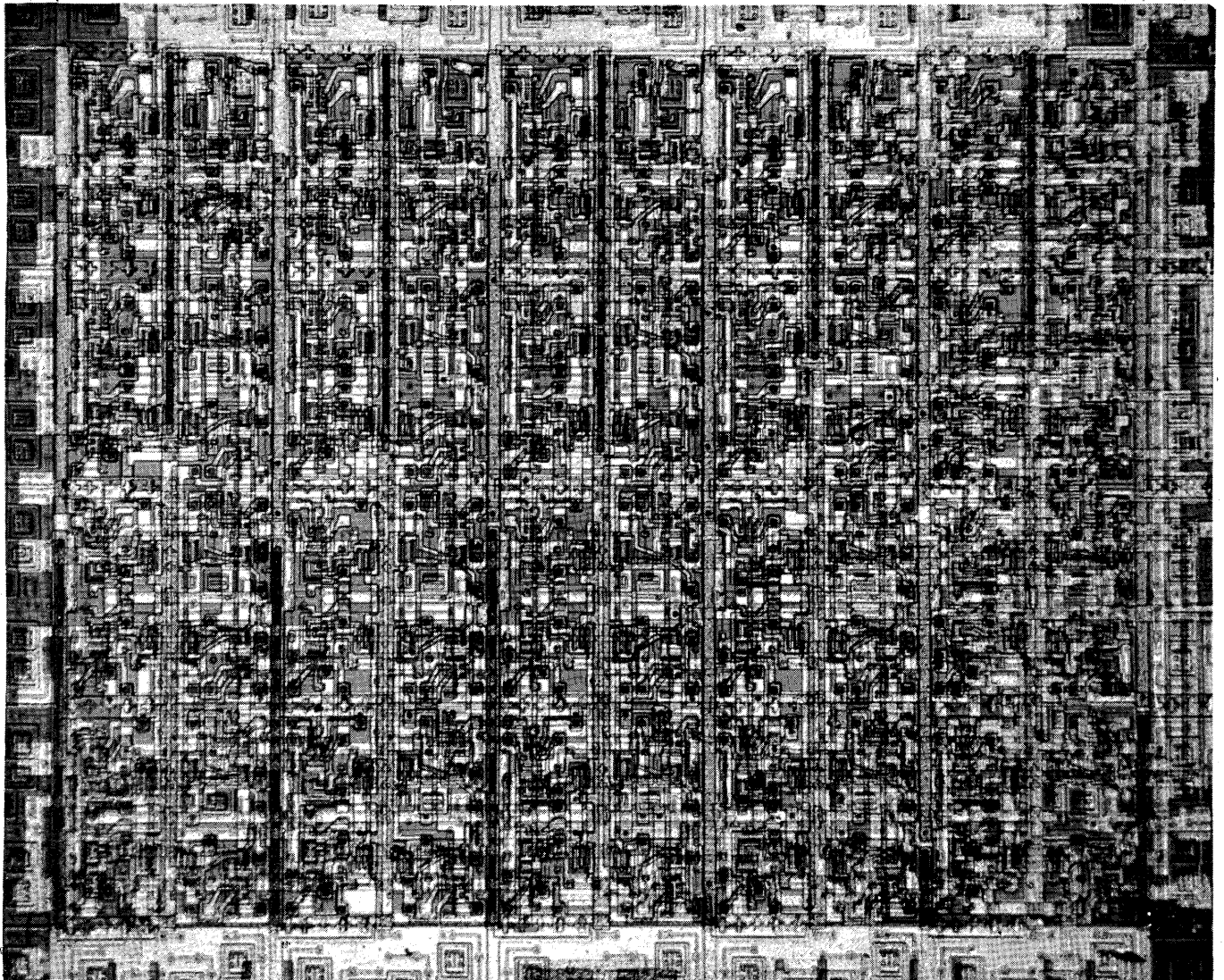


FIGURE 3—Microphotograph of AS-80

specialized hardware, of general-purpose capabilities where only specialized functions are required, of excessive programming costs, and of prohibitive processing times would be solved for both user and manufacturer. The producer of computing machines would build up an inventory of these basic blocks and assemble them to the size and configuration of the user's requirements with a minimum of system design effort. Computing systems will then be defined by the total number of bits of all working registers, rather than by the number and the size of registers. Fixed hardware implemented blocks will "personalize" the system. The user, on the other hand, will not be forced to buy capacity and features he does not need and the cost would become linearly proportional to the size and computing power required.

With this concept in mind, the current and future computing systems can be characterized as:

- Third generation: Specialized hardware, General-purpose systems
- Fourth generation: Specialized systems, General-purpose hardware

RACON-4 is a feasibility model of a fourth

generation system utilizing fourth generation hardware. The system block diagram (Figure 4) can be best explained by starting from the working register area. The workhorse of the system is a number of RA-4 register arrays. The actual number of RA-4 arrays is determined by the need and the budget of the user. Each RA-4 or groups of RA-4's can be programmed by a read-only memory to perform any of sixteen functions. The output of this memory passes through the Register-Function Matrix (RFM) which couples a number of registers together to perform identical functions and thus determines the word length. The contents of the registers are routed through the Register Data Bus (RDB) which is a single, dual, or triple-bus system, depending on the performance requirements. Both the bus system and the scratch-pad memory are controlled by the output of the micro-command memory. This memory consists of Read-only Command Memory (RCM-16) arrays, each containing 256 bits of information in a 16 x 16 arrangement, and the Read-only Memory Logic (RML) which is basically an addressing system.

A sequence (or block) of micro-commands is selected by the Instruction Library memory. This memory is composed of the same hardware as the micro-command memory. An additional read-write section is available as an option for the user, and utilizes the same RWM-16 hardware as the scratch-pad memory. The input/output Data Register is an optional feature since the scratch-pad memory could serve the same purpose, with performance restrictions.

For better understanding of the features of RACON-4, consider a processing unit consisting of sixteen four-bit variable arrays and a Read-Only Memory Module as shown in Figure 5. The memory contains the word address logic, a number of parallel words, and an interconnection matrix with 64 plug-in terminals. The four control inputs of each of the sixteen register arrays are permanently connected to the other half of the 64 terminals. With this concept, each word of the memory can program every 4-bit register to perform any of the sixteen functions. By inserting a point-to-point interconnection pattern between terminals and bit lines of the memory, the processor can be operated as a 4-bit machine with sixteen registers. It requires words 64 bits long from the memory for every step of operation. The insertion of the pattern shown in Figure 5(a) will

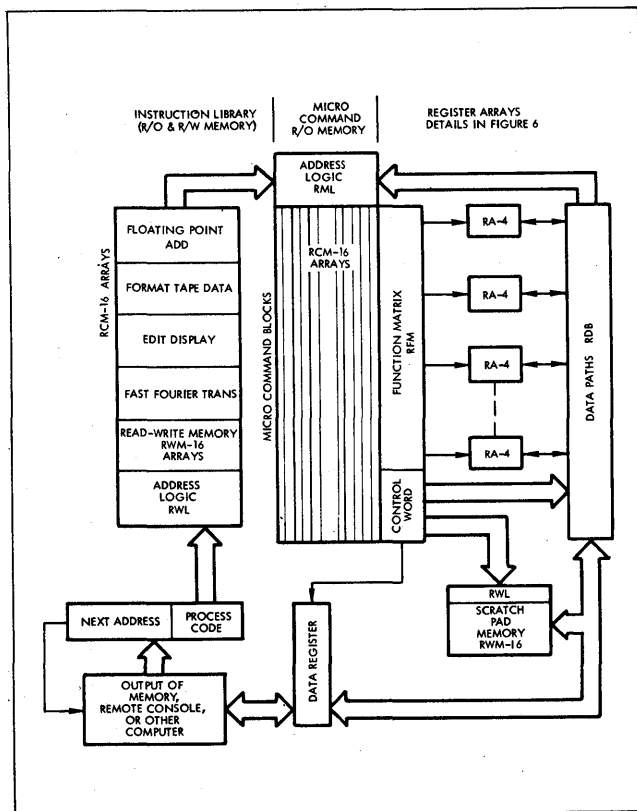


FIGURE 4—System block diagram, RACON-4

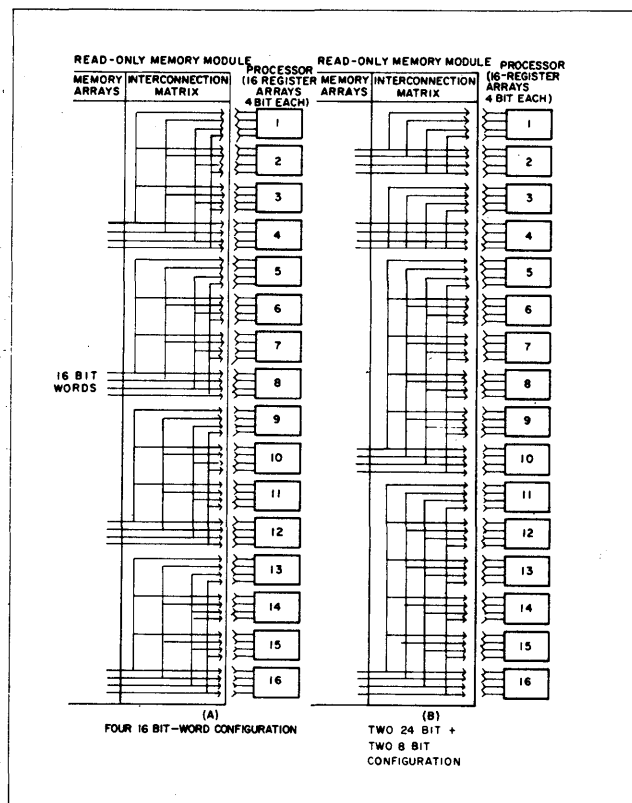


FIGURE 5—Register arrays for a 64-bit processor

result in a 16-bit machine with four registers requiring only a 16-bit word from the memory for each cycle. Another interconnection matrix could result in a 32-bit machine with two registers programmed by 6-bit words from the memory. Figure 5(b) shows a further possible configuration.

The RA-4 is a 4-bit register array similar to AS-80, but instead of the increment/decrement logic, a 4-bit full adder is incorporated. Programmable functions are increased to sixteen. Because

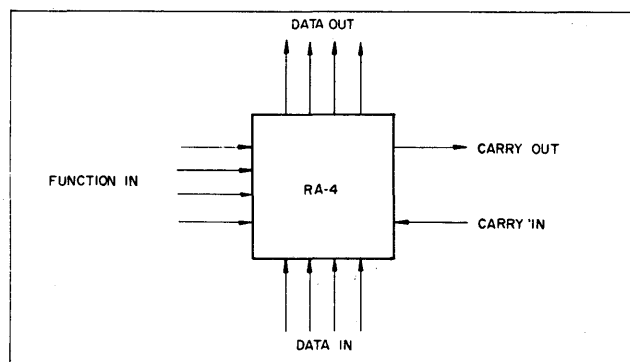


FIGURE 6—Block diagram of RA-4

of the bus oriented organization of RACON-4, the multiple entry of AS-80 was eliminated, as shown in Figure 6.

The array is capable of performing any one of the following functions in eighty nanoseconds:

- | | |
|---------------------------------|-----------------------------------------|
| 1) Load | 10) Decrement and all zero detect |
| 2) Load complement | 11) Shift left by 1 place |
| 3) Add | 12) Shift right by 1 place |
| 4) Subtract | 13) Shift right by 2 places |
| 5) Reset | 14) Add and shift right by 1 place |
| 6) Exclusive OR | 15) Add and shift right by 2 places |
| 7) Logic AND | 16) Subtract and shift right by 1 place |
| 8) Hold | |
| 9) Increment and all one detect | |

Over 160 gates are needed for the implementation of RA-4, resulting in a unit with a gate-to-pin ratio of ten.

The flow of data is entirely under the control of the micro-command memory, with the aid of RDB and the scratch-pad Memory. The process portion of the instruction to be executed is used as a starting address in the Instruction Library. The contents of this memory are one or more words which form the starting addresses of the micro-command memory. The micro-command memory contains the group of 4-bit numbers that program the register arrays to perform the appropriate micro-commands.

Linking of 4-bit groups together can similarly be accomplished by "programming" to produce any multiple of 4-bit word sizes. Furthermore, different computer words sizes may be associated with different processes programmed in the second memory. This would facilitate, for example, the effective utilization of a computer with an external device.

Because there is no restriction on the special-purpose instructions or the internal formats and word size which could be created, the basic hardware building block approach leads to a system organization which provides virtually unlimited flexibility to the user. In addition, the potential software advantages are considerable; for example, existing software can be utilized without radically modifying existing programs. New instructions can be easily added to the existing instruction set and higher meta-level instructions can be created. Consequently, both programming and machine design becomes easier and more

adaptable to any application without any loss in generality.

The entire control system, consisting of all control algorithms and computer instructions, can be represented by programmed blocks of fundamental machine operations placed in the micro-command memory. This approach differs from the micro-programmed computers in which specific computer functions are programmed, or from a standard computer in which specific functions are hardwired. Instead of designating special registers (accumulators, counters, index registers, etc.) and parameters (transfers, invert, etc.) in each of these blocks of machine operation, a variable representation is used. Consequently, the blocks are general, so that each block can, with different register and parameter designations, perform the functions of a large number of specific instructions.

Creating the entire set of computer instructions with these general micro-command blocks may be contrasted with present-day digital computer control memory techniques. In today's computers, the instruction complement consists of a collection of special-purpose instructions which differ slightly in terms of processes; there are many different instructions, but only a few unique operations. Using the general micro-command blocks as a basis, any desired set of instructions can be defined.

In fact, different sets of blocks can be specified to effect different system characteristics, depending upon the application. A representative set of blocks forming a complete (and powerful) instruction complement is listed below, together with the specific instructions accomplished within the block:

- 1) Transfer: This block implements the operations of transfer, enter, execute, single-bit tests, etc.
- 2) Count: This block implements the count (increment or decrement).
- 3) Shift: This block implements all forms of single precision/double precision left/right, open/closed, algebraic/logical shifts.
- 4) Exchange: This block implements swaps between any two registers.
- 5) Logical: This block implements the logical operations such as "and", "or".
- 6) Sign and Magnitude Add-Subtracts: This block implements all of the variations of

sign and magnitude addition and subtraction.

- 7) Mask test: This block implements the on/off tests of fields, tests of any collection of bits, and indicator resets.
- 8) Three-way comparison: This block implements all three-way comparisons as well as two-way logical comparison tests.
- 9) Multiply Divide Iteration step: This block implements the basic process for fixed/floating multiplication/division of fixed or variable length.

Because any register can be used by the micro-command block, arbitrary names for different registers are unnecessary. Because a register, such as an accumulator, has a definition in terms of its use, it may be convenient to give a name to a register in which a specific function or process is generally performed.

As a direct consequence of the fact that the general forms define basic processes, they are independent of normal computer word characteristics, such as word size and internal field definitions. Registers, in application, may have different word sizes. Consider, for example, the general form for a shift operation. A general shift operation is typically a set of the following operations:

- 1) Shift A n places and store in A.
- 2) Shift B n places and store in B.
- 3) Repeat steps 1 and 2 m times.
- 4) Shift sign of A to sign of B.

In this routine, A and B represent registers, the parameter n represents the hardwired shift algorithm, and the parameter m represents the number of times the shift algorithm is used. Obviously Steps 1 and 2, used together, represent a double precision shift; Step 1 alone is a single precision shift.

It should be observed that A and B could be different length registers.

As a result of being able to program the basic control sequences in terms of primitive forms, only a small percentage of control logic remains to be implemented. Part of this logic, primarily the creation of more complex instructions, and the control logic required to establish internal formats, can also be handled by programming techniques.

Specialized processes consist of collections of the basic forms. The program to implement these

processes utilizes the "language" of the basic general forms. Consider the floating point operations; these are programmed by combining the basic forms. A "floating add" is a collection of "transfer," "logical," "exchange," "shift," and "sign and magnitude" subroutines. "Floating subtract" is the same as "floating add" after the first subroutine, a sign change. This is analogous to the manner in which the control of complex computer instructions is presently achieved in computer design (built-in subroutines with specialized sub-orders). For example, the standard multiply instruction consists of 1) determining the sign of the answer, 2) the multiplication process of add, shift, and iterate, and 3) the final adjustment of the answer.

The programming of these specialized processes is performed in a second read-only memory called the instruction library. The ability to program this memory, using the general building block forms in the micro-command memory as a source language, allows functions normally ascribed to the software to be built into the hardware by being programmed in the instruction library. Thus specialized instructions can be created and specialized routines for which high processing speed is required, such as input/output mechanizations (reading and writing data, editing procedures, input/output buffer control) or a Fast Fourier Transform algorithm, can be implemented in addi-

tional to normal computer instructions.

This approach permits the computer to be tailored for many specific applications within the mass-produced standard computer design without the need for specialized software. Much of the complexity of a higher order language is bypassed.

CONCLUSION

Recent developments in batch fabrication of logic and memories have made it necessary to reconsider the interrelations between design effort, hardware cost, and partitioning. It is possible to achieve a low "part-number" count by making use of multi-purpose register array packages that include control logic inputs. These control inputs are supplied from read-only memories. The control inputs cause the data in the package to undergo micro-programmed operations to form the instructions of the computer.

REFERENCES

- 1 R C JENNINGS
Design and fabrication of a general-purpose airborne computer using LSI arrays
Digest of IEEE Computer Group Conference June 1968
- 2 H R BEELITZ et al
System architecture for large-scale integration
Proceedings of Fall Joint Computer Conference 1967
- 3 C J WALTER
Impact of fourth generation software on hardware design
IEEE Computer Group News July 1968

Engineering for systems using large scale integration

by C. F. O'DONNELL

North American Rockwell Corporation
Anaheim, California

INTRODUCTION

Our experience in designing and producing over 100 different MOS large scale arrays (LSA's) for a variety of systems and customers has led to a number of changes in our pattern of engineering operations. We have found that the major increase in device complexity called for in LSA's requires extensive use of design aids for acceptable development schedules and costs. In turn, the generation and use of these design aids tie device designers and the logic/system designer closely together.

These design aids include the software required for checking logic, for device layout, for the generation of device and system test programs and, in combination with suitable hardware, for device mask generation.

This paper concentrates on the status of interaction between device and logic/system designers and design aids but does include probable short term trends in the evolution of engineering in the LSI area.

To illustrate the capabilities of today's technology, we describe the Autonetics D200 computer which is built using MOS LSA's. Included in this discussion are the tradeoffs leading to its design, its characteristics and organization, and finally some details of the LSA's used in building it.

Criteria for design

To keep this talk confined to a reasonable time, I will cover only the engineering of digital systems using MOS devices in Large Scale Arrays (LSA's). As can be seen from Figure 1, in my view the change in engineering interaction as we progressed from discrete components through integrated circuits (IC's) to LSA's have been evolutionary rather than revolutionary. There has been

more of a change of emphasis on interaction between engineering functions rather than the formation of completely new groups, an adaptive response of engineering to changes in the characteristics of the devices used.

The change from discrete components to integrated circuits, for example, forced logic designers to become involved in the hardware task of board layout. Board interconnection minimization became an essential ingredient of logic partitioning. With board element counts shifting from hundreds to thousands, without new logic design criteria off-board lead requirements easily outstripped physically feasible numbers of connector contacts.

With low-cost IC's available the ability to eliminate every possible resistor, diode or transistor was no longer the most significant criterion in evaluating the efficiency of a logic designer. The ability to partition logic so relatively few expensive interconnections between boards would be required had become dominant.

While LSA's provide low-cost elements, they do intensify other system development problems. One of LSA's weakest points lies in the difficulty of insuring that the first design works, and the expense and schedule slips involved in correcting design deficiencies are minimized. With discrete components, a few overstressed parts could readily be replaced with little impact on system cost or schedules. Even with integrated circuits, jumping of connections and replacement of one standard packaged unit with another enabled the engineer to take care of the majority of his problems. When an LSA logic design error or local overstress condition occurs, then design corrections must be made, mask sets changed and a new production run of the device made before the system can be checked out.

The increased impact on system costs of logic

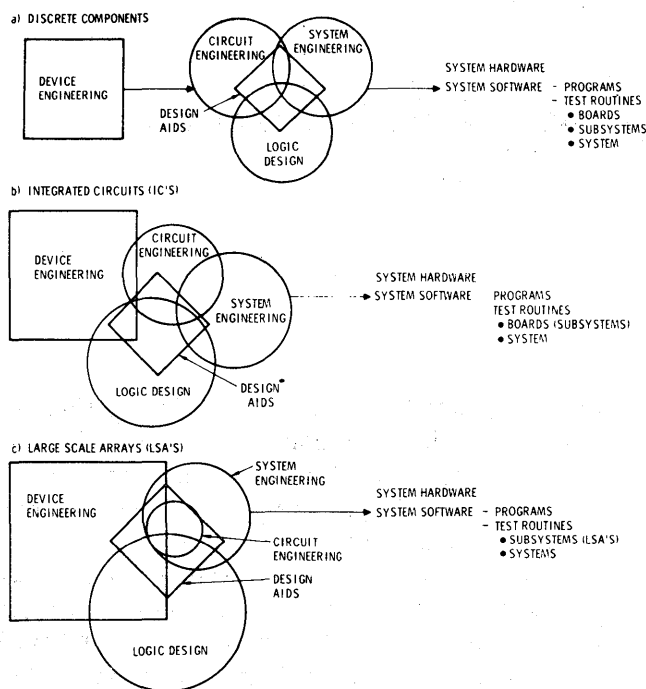


FIGURE 1—Engineering interaction, digital system design and development

errors has involved the logic designer more directly in device engineering. He has far greater impact on the device supplier than was the case in discrete component days. Each time the device supplier has moved more deeply into system fabrication he has lost some of his flexibility. With IC's he was free to design the circuit details as he saw fit provided specified operating characteristics of the device were met. However, his sales depended heavily on whether logic designers decided to use RTL, DTL or TTL circuits in their designs and the range of fan-in and fan-out ratios they required and other obvious choices. With LSA's an additional degree of flexibility has been lost. In many cases the device becomes such a specialized subsystem that it must be tailored to the needs of an individual system manufacturer.

Criteria for user

While there are problems, the incentive of producing low cost, complex electronic systems built with LSA's is sufficiently high to justify a major expenditure of engineering effort to overcome these problems. To make this objective of low cost systems feasible, the engineering system for design and development must be set up to cope with these characteristics of LSA design which differ from discrete component and IC design.

LSA's are not available from manufacturers as off-the-shelf items with the exception of a few standard functions such as shift registers. Seldom is it possible to work around a missing subsystem, which is really what a LSA is. Therefore, these devices when used can become system schedule limiting factors and major contributors to system expense. As an LSA is a subsystem, it represents a higher percentage of total system cost than previous electronic devices. Consequently, errors in original cost estimates can have greater impact on program profits than was the case for previous devices. Certainly high efficiency is called for in the system both for engineering design and development and in production.

These factors all point to the need for engineering excellence in design and accuracy in carrying engineering intent thru into hardware. Both points call for design aids and computer programs to do the detail drudgery with high accuracy, and computer controlled precision equipment for tasks such as mask generation and device assembly.

This leads to the engineering configuration shown in Figure 1-C, where the circuit design function has been largely absorbed by device engineering and logic design. The growth of logic design function has been due to its direct involvement in LSA device design and its role in developing many of the design aids used by the remaining engineering functional groups.

This intensification of interaction between device supplier and user is not without its own set of problems. Their nature can be understood by examining what happened when multilayer boards were developed for IC interconnection. These boards were both more compact and uniform than the combination of two-sided printed circuit boards and cabling they replaced. What cross talk, capacitive loading and lead resistance there was remained reasonably uniform from board to board and system checkout yield improved. However, a defective board was a complex problem.

Locating and fixing board errors particularly in internal layer interconnection routings was difficult, time consuming and costly. Because this was done in the system manufacturer's shop, surface jumper connections and other quick fix techniques could be used to avoid severe schedule penalties with their accompanying dollar costs.

In the case of a LSA however such quick fix techniques are not feasible. Furthermore, their turn around time includes the time lag in reaching

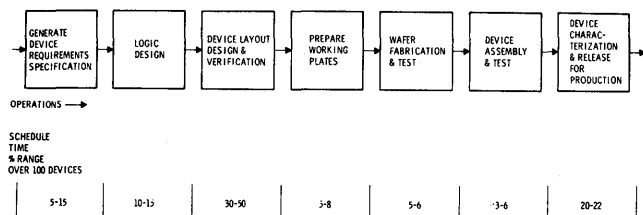


FIGURE 2—Device design cycle

agreement with the device supplier on both the nature of the problem and the fix. It may involve additional problems, such as competing priority projects in that supplier's shop. As a result, some system suppliers have felt it necessary to develop the engineering capability in-house for LSA production. Others have developed a design capability, but propose to use established semiconductor manufacturers for device production. It is still too early to say how the LSA supplier-user interface will stabilize, but it is certain that over the next few years a number of problems in this area must be worked out.

Device design cycle

Leaving the supplier-user interface problem for the moment, consider where design aids could be most profitably employed in the device design cycle. The areas of Logic Design and Device Layout show up in Figure 2 as taking half or more of the total cycle time, so must be considered prime areas for design aids. As these aids have a major impact on the skills required of device design engineers, it is worth reviewing them in some detail.

The uses of logic simulation during the early logic design phase are well known, even from discrete component days. It was necessary to write new programs for MOS 4-phase logic. While this technique dramatically reduces circuitry power requirements from dc or 2-phase designs, it does provide a fascinating problem for the logic designer. The lefthand side of Figure 3 shows the logic equations set down in format suitable for entry into the computer. The righthand side shows the situation at the nodes designated at the

FIGURE 3—MOS logic simulation program

<pre> *1301B INPUTS ART BRT RRT EA EB DMS A2B B2B R2B DB DK1 DK2 B12B C10B0 * C11B0, T12B, SYNC *1301B OUTPUTS ELD E1L E2L E3L E4L E5L E6L E7L E8B AD1B = ART BD1B = BRT RD1B = RRT DS1B = A2B (AD1B + EA) + B2B (BD1B + EB) + R2B (RD1B C01P + DMS C01P) DB1B = DS1B / (DB1B B12B-) D41B = DB1B D21B = D41B D11B = D21B ZERO = C10B0 + ZERO DS1B- ES1B = C11B1 (DS1B + ZERO) + ES1B C11B1- ELC = C11B1 ZERO + C01P ES1B- (T12B B12B-) + C11B1- CE ES1B EB1B = DS1B C10D + E11B C10D- C10B = C10B0 + C10D C11B0- E41B = EB1B E21B = E41B E11B = E21B CE = (T12B / E11B) - (CE + B12P) DC81B = DK1 DK2 C10D + DC11B / DCK DC41B = DC81B DC21B = DC41B DC11B = DC21B DCK = C10B0 (DK1 / DK2) + DCK (CK2 / DC11B)- DINH = B2B C01P (T12B B12B)- (DB + CDC)- B12B + DINH (CDC B12B)- C0C = (T12B / DC11B)- (CDC + B12B) C01P = SYNC + C01P (T12B B12B)- EL8 = (C11B1 (A2B + B2B) (DS1B + ZERO) + DINH)- (DB1B- + D21B)- EL7 = (C11B1 (A2B + B2B) (DS1B + ZERO) + DINH)- (D21B- + DB1B D11B- + DB1B- D41B) EL6 = (C11B1 (A2B + B2B) (DS1B + ZERO) + DINH)- (D21B D11B- + D41B- D11B-) EL5 = (C11B1 (A2B + B2B) (DS1B + ZERO) + DINH)- (D21B- D11B- + DB1B- D21B) EL4 = (C11B1 (A2B + B2B) (DS1B + ZERO) + DINH)- (DB1B- + D41B- + D21B D11B-) EL3 = (C11B1 (A2B + B2B) (DS1B + ZERO) + DINH)- (DB1B / D11B </pre>	<pre> 1801 1802 1803 1804 1 1805 2 1806 1807 1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821 1822 1823 1824 1825 1826 1827 1 1827 2 1828 1 1828 2 1829 1 1829 2 1830 1 1830 2 1831 1 </pre>	<pre> A2B 0000 0000 0000 0000 0000 0000 0010 1111 0001 0000 0000 0000 A2B0 1111 1111 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 AC1B 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 ALL 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 ART 0000 0000 0011 0010 1010 0000 0000 0011 1001 0000 1000 0100 1100 ART16 1010 0100 0011 1001 0000 1000 0100 1100 0000 0000 0011 1001 ART32 0000 1000 0100 1100 0000 0000 0011 0010 1010 0100 0011 1001 B022B 0001 0101 0000 1100 1000 0100 1101 1010 0100 0001 1100 0100 B02B 0001 0001 0001 0001 0001 0001 0001 0001 0001 0001 0001 0001 B12B C010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 B22B 0100 0100 0100 0100 0100 0100 0100 0100 0100 0100 0100 0100 B232B 0000 0101 0100 0011 0010 0001 0011 0110 1001 1000 0111 0001 B2B 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 B2B0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 B32B 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 B82B 1001 0001 0011 0000 0110 0010 0000 1010 1000 0100 0100 0100 H01B 0000 0000 1001 1000 0111 0001 0000 0101 0100 0011 0010 0001 B02B 0000 0000 1001 1000 0111 0001 0000 0101 0100 0011 0010 0001 BRT 1000 0000 0100 1100 0011 1000 1000 0010 1100 0001 1001 0000 BRT16 0011 1000 1000 0010 1010 0001 1001 0000 1000 0000 0100 1100 BRT32 1010 0001 1001 0000 1000 0000 0100 1100 0001 1000 1000 0010 C01P 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0011 1111 C10B0 0000 0001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 C10D 0001 1110 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 C11B0 0001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 C11B1 0010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 C9B3 0000 0000 1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 CDC 0000 0011 1100 0000 0100 0000 1100 0000 0100 0001 1100 0000 CE 0000 0001 1100 0000 0100 0000 1100 0000 0100 0011 1100 0000 D11B 0100 0011 1101 1110 0110 0110 0110 1111 0001 1011 1010 0100 0011 D21B 0010 0001 1110 1111 0011 0011 0111 1000 1101 1101 0010 0001 D41B 1001 0000 1111 0111 1001 1001 1011 1100 0110 1110 1110 0000 D81B 0100 1000 0111 1011 1100 1000 1101 1110 0011 0111 0100 1000 DB 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 DC11B 1101 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 DC21B 1110 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 DC41B 0111 0111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 DC81B 1011 1011 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 DCK 0000 0010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 </pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

LOGIC EQUATIONS

BIT BY BIT OPERATION FOR THREE CLOCK CYCLES

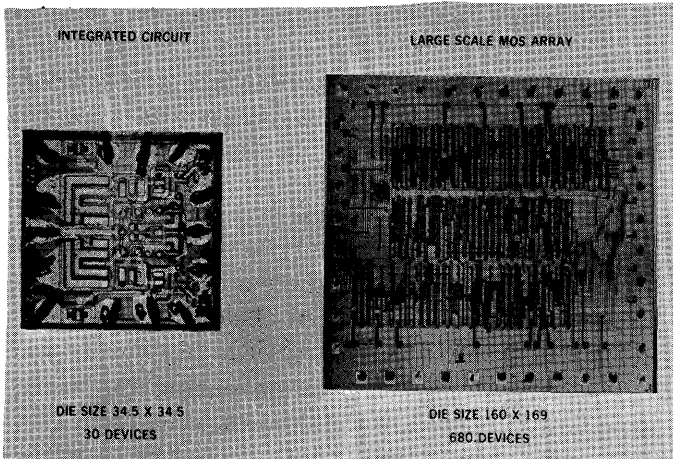


FIGURE 4—Complexity comparison

lefthand column during three clock-cycles of operation.

Other than this modification for multiphase clock operation, our logic programs are very similar to ones which have been in widespread use by computer designers for the past decade. Therefore, I propose to leave this area for that of device layout and mask preparation. Here development has been most rapid over the past few years.

Device layout

As IC technology is well developed, why should the change from IC's to LSA's pose such a problem in device layout? The answer is shown strikingly in Figure 4. The increase in device complexity from IC to LSA is so great that problems become different in kind rather than simply different in magnitude. Generating and checking a mask set for the production of 30 element dual quad-NAND gate IC of Figure 4-a is inadequate preparation for checking mask sets for 600 to 6000 element LSA's. The complete \pm arithmetic unit in Figure 4-b is a challenge to a designer. Perhaps even more daunting is the example of the latter shown in Figure 5, a 1000 bit shift register employing over 6200 active elements.

An engineer using his detailed knowledge of the dual nand gate bipolar circuit of Figure 4-a could with care verify the completeness of each mask and registration from mask to mask. When he must check for the possible omission of one of several thousand p region windows or of a similar number of interconnections, he faces an impossible task. With care, he can catch all but 1% or so of the errors, but when the points to be checked number in the thousands the resultant possible number

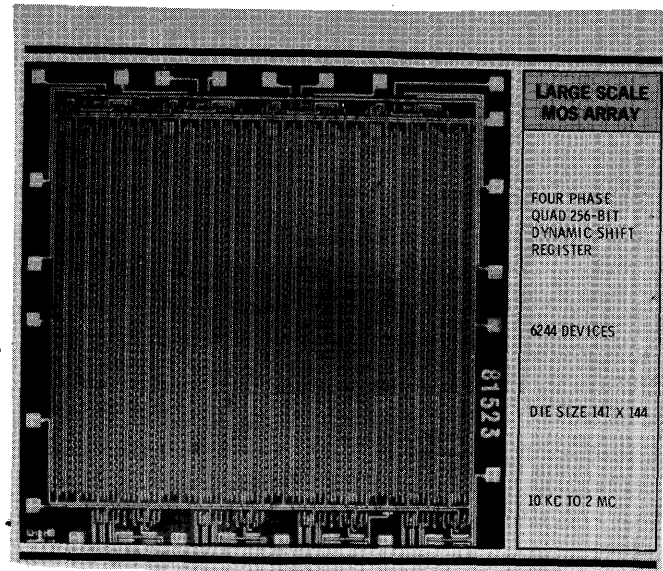


FIGURE 5—Large scale MOS array

of defects is completely unacceptable. This increase in complexity then cannot be coped with by simply being more careful. The engineer is forced to become more heavily dependent on computer programs and other hardware aids for both checking and indeed for the original device design.

The process steps required to do the layout and produce a mask set are shown in Figure 6. The top row shows the manual operations required to produce a device mask set, given the logic equations to be mechanized. In our experience approximately 1100 manhours were required to complete the operations shown for small to medium sized arrays using custom design. For one reason or another attempts have been made from time to time to use this technique to produce masks for truly large arrays in the range of 800 to 1000 elements. These attempts could not be called successful.

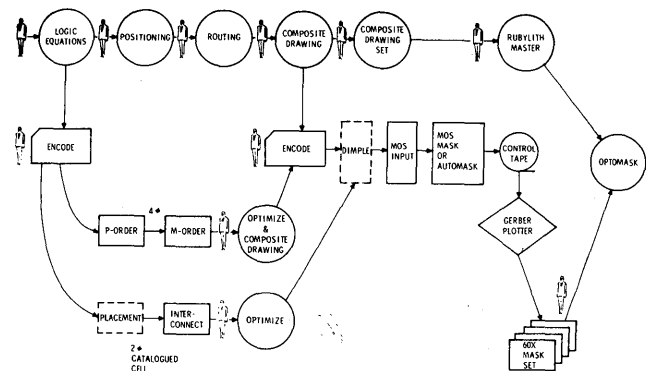


FIGURE 6—MOS-FFT LSA computer-aided design system

To increase the probability of successful device designs we wrote a series of programs for the engineer to use as aids in doing his original device layout. For 4-phase custom circuitry, we decided that a completely automatic layout program would not be a cost-effective first step.

For the present, dropping to the second line of the flow diagram of Figure 6, we have two programs which act as engineering design aids—the P-ORDER and the M-ORDER programs. The logic equations to be incorporated on the chip are written out in equation form, encoded and used as inputs for these programs. The output of the P-ORDER program is a list of logic equations ordered so as to minimize the total length of interconnecting metal and the number of crossovers. The p regions, each of which represents an equation to be mechanized, appear in sequence so common terms are grouped together. As these terms are connected by metal lines, this grouping minimizes their length.

The resultant ordered equation set is then fed to the M-ORDER program. This program interleaves the interconnecting metal to minimize total chip area. The P-ORDER program has generally provided a fairly long loosely packed array. The

M-ORDER program attempts to square up this array and fill the empty spaces reducing the total p length as well as chip area.

A number of iterative runs are made with these programs. The engineer evaluates the results after the initial 50 or so iterations and intervenes manually to rearrange the sequence of logic terms or their position on the chip whenever it is apparent to him that such a rearrangement will be an improvement. Presently, this requires that a computer printout be delivered to the engineer. The future incorporation of a graphics terminal will let us use the programs in a truly interactive way and so reduce the design process flow time.

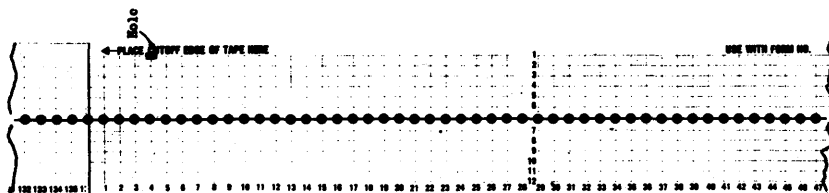
The ordered equations are now used in producing a hand drawn composite, like the section shown in Figure 7. This composite is used for encoding the input to the computer programs generating the control tape for the Gerber plotter. The total direct labor hours required for these operations on the average amount to 500 hours. This is a reduction by a factor of 2 from the time required to carry out an all-manual operation. For LSA's of the size we are currently producing these programs make all the difference between a possible and impossible design task.

```

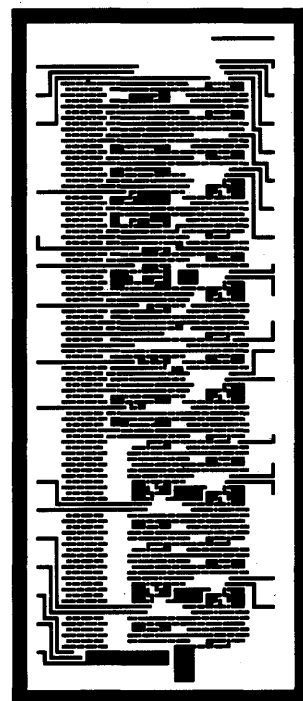
RETICULE LIST FOR NAA 1303
START NAA 1303 PWM NO. 2 04/05/66 01000,33000
01303,33000 C20C0,31C 00 00405,32000
PWM NO. 2 RETICULE 60X915
C8150,34C00
0C00C,00280 (C0000)01080 (C0000)C1890 (00000)0268C (0C000)C348C (00000)0428C (0C000)05090 (00000)0588C
(0C000)0668C (0C00C)0748C (00C00)C4290 (00000)0908C (C0000)09390
(C0C0C)33000 STOP.
PWM NO. 2 RETICULE 60X914
C814E,34C00
00520,0000C C1320(0C00C) C2120(0C0C0) 02920(00C00) C372C(00000) C452C(0000C) 05320(00000) 06120(00C0C)
C6920(00000) 07720(0C000) 08520(00000) 09320(0C0C0) C9968(0000C)
C000C,33000 STOP.
    
```

DATE 04/05/66 PAGE 1

COMPUTER GENERATED RETICULE LIST



COMPUTER GENERATED PLOTTER CONTROL TAPE



60X MOS 'P' MASK GENERATED BY THE PLOTTER

FIGURE 7—MOS mask generation

The third flow line in Figure 6 shows a different set of design aid programs used for placement and interconnection of standard catalog cells for 2-phase logic circuitry. Design direct labor hours are saved by using these standard catalog functions, as much of the detail chip layout has already been done and is available in the computer data bank. Only the placement of these cells and their interconnection remains to be carried out. There is considerable direct labor time saving using this method. On the average we find it to be about half that required for a 4-phase custom layout or approximately 250 hours. In most cases, however, there is a 10–20% penalty paid in chip size for this use. For this reason, the standard cell approach is often used during development in order to reduce schedule time. A custom design layout is then carried out if a sizeable production order follows.

To reduce human errors in physical mask production, a tape controlled automatic plotter is used. To produce the tape, it was necessary to encode, from a hand-drawn composite of the device, each incremental step of the plotter defining the p and m regions. This took up to 300 hours of technician time.

From this listing the input card deck for the MOS input program was prepared. The output of this program was used as input to the MOS-MASK program which prepared the plotter control tape. This tape, a section of which is shown in Figure 7, specifies the reticle to be used at each XY coordinate of the masks. The mask sets are first plotted at $120 \times$ or $60 \times$ final size. Proof prints are then sent to the design engineer for visual check before making the working plates. If the check is satisfactory, then standard procedures for mask set production are followed.

Three hundred hours of technician time was felt to be excessive for encoding the input to the MOS-MASK program. We then took the next step of writing a simplified intermediate program known as DIMPLE. DIMPLE takes end point encoding and translates it into incremental encoding in a form suitable for use as input to the MOS-MASK program. This relatively simple addition reduced encoding time from 300 hours to 100 hours.

One additional feature to be added is a digitizer to replace the manual listing of end point encoding. This will further reduce errors and save schedule time. It is not so much the two to three

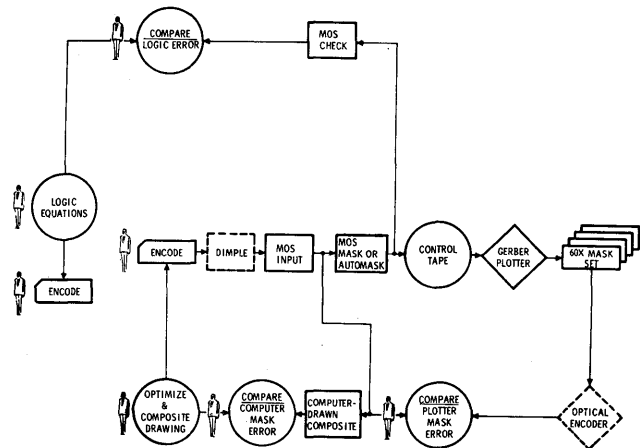


FIGURE 8—MOS-LSA mask checks

days of original encoding time we wish to save, but rather the 2–3 weeks involved in making corrections when an error has been discovered only after a device has been all through the production process.

Design checks

Even with this extensive use of design aids, it is still possible to have the occasional error creep into the design. As a result we have developed two additional checks, shown in Figure 8, to run thru before the masks enter the optomask system. One is a straightforward check on the logic mechanization. The output data from the MOS INPUT program is fed to the MOS CHECK program. There it is combined with a card deck giving the identification of each gate in terms of the logic equations mechanized. With these inputs the MOS CHECK program regenerates the logic equations mechanized on the chip and types them out. Presently, this printout is checked manually against the original set of logic equations to determine if errors have been made. The program is being modified to do this logic verification within the computer and print out only the errors.

In addition we use a computer printout, shown on the right-hand side of Figure 9, to determine that the plotter control tape is accurate. Computer printout is compared against the hand-drawn composite to be sure that no errors have been made in this translation.

Even if the plotter control tapes are perfect, there is still a possibility that the plotter will malfunction in a way not caught by our monitoring. We have under development an optical encoding

equipment which determines the location of all corners on the mask set produced by the plotter. The coordinates of these corners will then be compared with the end point encoding used as entry to the MOS MASK programs as a final check to ensure that the $120 \times$ or $60 \times$ final size original is a faithful translation of the original input to the plotting system. With this implementation we will have the highest confidence of being able to produce working devices functionally correct the first time thru the system.

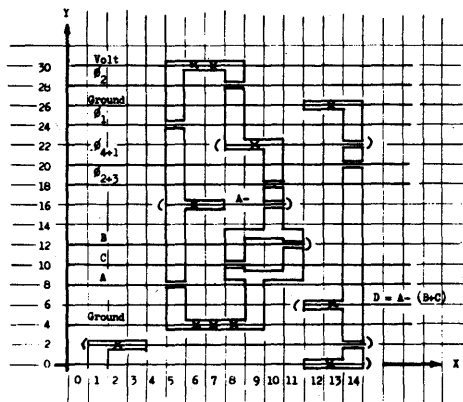
We still have not exhausted the ways in which design aids can assist in turning out a satisfactory product. Before committing to physical production, the designer needs to know that his device is not only logically correct but that it will operate without errors due to excessive gate loading or noise cross coupling for example. We had previously developed a series of programs for circuit analysis of thin film hybrid circuits and circuits using a combination of integrated circuits and discrete components. The programs exercised the circuits, simulating a variety of stress environ-

ments such as temperature extremes, high radiation levels and varying supply voltages.

Capitalizing on this experience we have written a series of programs for the analysis of MOS LSA's. MOS SNAP, for example, is used in doing an overall speed noise analysis. The capacity loading and noise coupling at any given group of nodes in the total nodes set can be determined using this program and supplied as outputs to the engineer for verification of performance.

A separate program called TRAC is used for local speed noise analysis at a restricted group of nodes in the device. Internally, the program forms an indefinite admittance matrix for each node and calculates current and voltage for the equivalent circuit, using an accurate non-linear model for each active MOS element. A representative section of circuitry for analysis, the computer input format and the graphics output to show transient response at a given node, are shown in Figure 10.

The calculations, while straightforward, are far too tedious, error prone and time consuming for an engineer to do by hand with any expectation



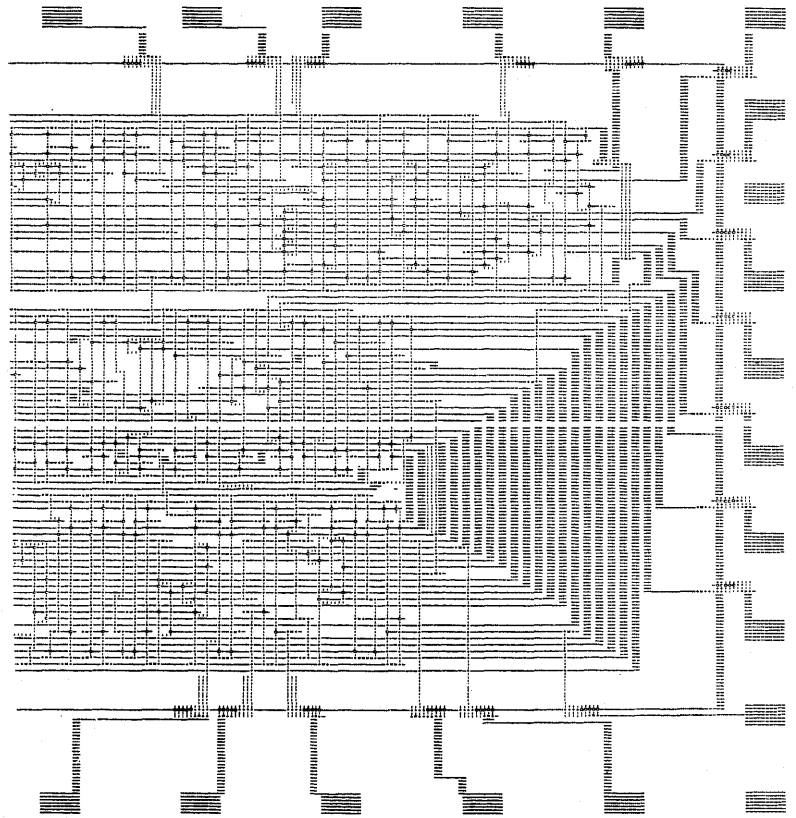
HAND DRAWN COMPOSITE

```

D38C5CA03R1514210271A2C021C027CAJAA
0D20G3G013AC0221 D1316151A01 D221 D1316C5CA0233G9G5G2A
D2C1D331A020C933CA 24G017G1G9G018A2CA2CAJ022C029CAD21029CA
210334G3G8A2C033CA 2D35A0441AD44CA0D19G023G5G2A CCD49CA

2C049CAJD24CA0241A 01D43151A 0C043C5CA 0D22G019G5G2AAA
24G017G7G1GD20A2C9C0221919CA219C D2219181A D126G017AAA
D11015G9G1A0101C1 D2516151A CCD101D2516C5CA0D12G011G1G015G5G2A
D231D111A D231 D111A0233G5G1G1A 0521A 052CAJ2C023CA2C0231A
2C0191D311A2C019CD31CA24G017G1G028AAA0231G3A
G1D22131D16151A0C022131016C5CA020G1G3G015G5G2AD1110251A0111D251A
0921G9G013G1A8C0431A8C0431A8C043CAJ2CA2CA24G019G029AD1CCD43CA
    
```

COMPUTER CODING



COMPUTER PRINT-OUT

FIGURE 9—MOS circuit coding

of useful results. With the TRAC computer program using 200,000 bytes of core memory, with a 2000 FORTRAN statement program, however, significant results can be obtained with only 3 minutes of computer running time for 20 circuit nodes.

Design aids summary

The programs described contribute substantially to device design cost reduction by reducing both the direct labor hours required and the recycles needed due to human errors in the design process. Developing these programs has been expensive and time consuming as you can deduce from the program summary descriptions in Table I, DESIGN PROGRAMS FOR CUSTOM MOS-FET 4-PHASE DEVICE. We feel that it has been worthwhile and indeed essential for providing us with an LSA design and development capability.

This paper has been a progress report on a continuing program of design aids development rather than a final description. In addition to the new programs mentioned in this paper as being currently under development, others will be starting in the near future to reduce obvious inefficiencies in our current operation. We continue to attack each area of the design and development process which has either high direct labor content or is schedule limiting. We pay particular attention to those areas in which our experience shows errors are apt to occur which cause devices to fail on their first run thru the design cycle. Our schedule times to produce and modify LSA's are still too long; our initial device design costs are still too high. From our progress to date, I feel that within the next 1-2 years we will have achieved an optimum system, balancing engineering hours on the one hand with the cost of com-

puter running time and precision design and development aid equipment on the other to minimize both costs and schedule time.

Trends

I predict that our current trend in engineering organization will continue. Within the next two years I would expect to see most engineering operations making use of LSA's set up as shown in Figure 11. Logic design and system engineering will have coalesced into a single group. Circuit engineering will have been fully absorbed by device engineering. Design aids, both program and equipment, will have equal rank with these engineering functions. Such an organization will minimize time required from system concept to production acceptance.

As for the engineering tools used by the organization, I expect a continuing proliferation of software but I also expect that these programs will remain as design aids rather than becoming a completely automated design program. The introduction of graphic terminals with time share computers will accelerate and reinforce this trend. As far as the hardware is concerned, in general it will become more rapid, with some electromechanical devices being replaced by electronic systems. In addition, the accuracy of this precision equipment will increase to allow either greater compaction of devices, therefore reducing costs by having more chips per wafer, or allowing design rules to be relaxed so greater yields can be obtained on chips of today's size.

As I have already pointed out, I expect the device design costs to be reduced sharply over the next two years as an optimum balance is reached by the engineering and by the software and hardware design and development aids.

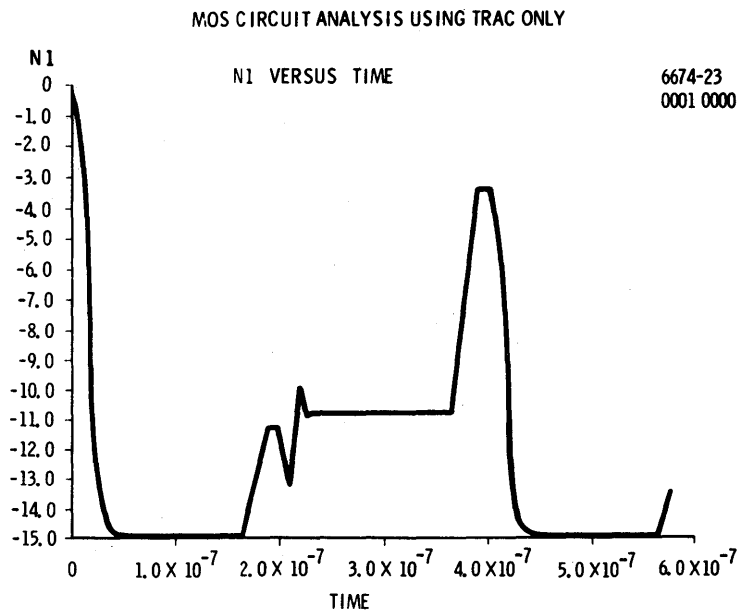
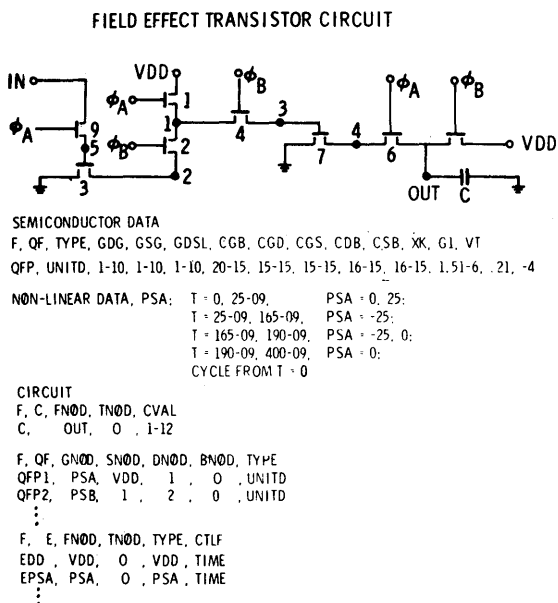
Spurred in part by the desire to be able to produce larger and larger chips with all elements working, the industry is increasing its knowledge and control of processes. This plus additional experience and the introduction of automation where desirable will lead to a rapid decrease in production costs over this two-year interval as well. Combination of design and production cost reductions will make LSA's a very formidable competitor indeed in the electronic device field by 1970.

The supplier-user interface for LSA's is difficult to predict at this time. My current feeling is that this interface will be quite flexible and econ-

PROGRAM NAME	CORE USED (BYTES)	NO. OVERLAYS	RUNNING (360-65) TIME (MINUTES)	NO. STATEMENTS (PL/I)
'P' ORDER	120K		10 - 15	1250
'M' ORDER	225K	4	20 - 30	2000
MOS INPUT	220K		1/2 - 1	1750
DIMPLE	120K		1 - 2	2500
MOS CHECK	350K	5	10 - 12	2750
MOS SNAP	150K		2 - 4	600
MOS MASK	250K	3	2 - 5	2200
MOS AUTOMASK	300K	4	2 - 5	2750

TABLE I—Computer aided design programs for custom MOS-FET 4-phase devices

FIGURE 10—Device functional analysis



PROGRAM INPUT

TRANSIENT RESPONSE

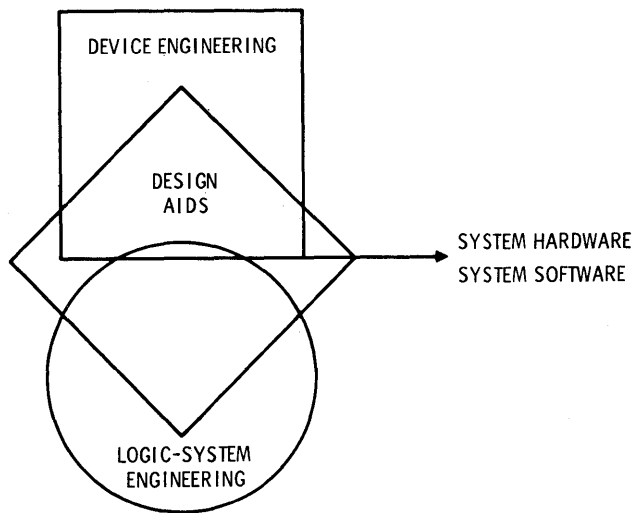


FIGURE 11—Engineering organizational trend, digital systems

omy-determined provided processes are reasonably standardized and both suppliers and users develop or acquire the necessary software and hardware design aids. This would allow users to do their initial design, possibly up to mask production, in-house and then obtain competitive bids for device production from the normal suppliers. On the other hand, if the suppliers also have a well developed device design capability then they could simply supply the LSA specifications and logic equations to the supplier and have him do the complete development job as well as production. It is still too early to say if this desirable state of affairs will in fact come about.

While our current system of designing and producing LSA's is not perfect, we have made considerable progress. Mr. Booher, a rare combination of logic designer and system engineer (though if my predictions are correct, we will see more of this in the future) will describe a General Purpose Parallel Computer developed using his MOS-FET 4-phase logic.

BIBLIOGRAPHY

1 R A RUTMAN

- An algorithm for placement of interconnected elements based on minimum-wire length*
 Proceedings of the Spring Joint Computer Conference 1964 pp 477-491
- 1 H S SCHEFFLER et al
Reliability analysis of electronic circuits
 AD 461303 Defense Documentation Center March 1965
- 3 W HOCHWALD C T KLEINER
Digital simulation of nonlinear electro-magnetic circuits
 IEEE Trans on Magnetics Vol MAG-2 No 3 September 1966 pp 532-529
- 4 *Special report on LSI*
 Electronics Vol 40 No 4 February 20 1967 pp 123-182
- 5 C T KLEINER E D JOHNSON W D ASHCRAFT
A general purpose system of digital computer codes for linear/nonlinear network and system analysis
 IEEE International Convention Digest March 1967 pp 424-425
- 6 R S MILES et al
Design and analysis of electronic circuits
 Proceedings of the SHARE and ACM Design Automation Workshop June 1967
- 7 *Special issue on CAD*
 Proceedings of the IEEE Vol 55 No 11 Nov 1967
- 8 J RHEA et al
Special report on LSI
 Areospace Technology January 29 1968 pp 24-46
- 9 L R McMURRAY et al
Transient radiation analysis by computer program TRAC
 Harry Diamond Laboratories June 1968

MOS GP Computer

by R. K. BOOHER

*North American Rockwell Corporation
Anaheim, California*

INTRODUCTION

When the 2-phase, 20-bit shift registers using P-channel enhancement mode MOS-FET's were first introduced on the market many of us did not have the foggiest idea of what a MOS-FET was. Logical designers were intrigued by the functional complexity which the technology appeared to offer. System designers were pleased with the prospect of lower power and fewer package requirements. Semiconductor manufacturers predicted that MOS would be forgotten sooner than the tunnel diode.

Autonetics studied the potential and the problems associated with direct coupled and 2-phase MOS circuits and promptly devised the 4-phase system which solved many of the problems of the 2-phase system. The first MOS-LSA using the 4-phase system was the Autonetics DDA Integrator. The first completely operational unit was produced during February 1966. This device has approximately 800 MOS-FET'S.

After gaining considerable experience with the DDA and other 4-phase MOS-LSA circuits, we felt we were ready to design a general purpose computer suitable for navigation applications.

At this point I would like to state that this computer, using MOS-LSA's, is fully operational. The computer was designed to satisfy the general requirements of a navigation computer, consequently it is neither the world's fastest nor the most versatile computer, but it does represent a quantum jump in the technology which produced it. The components are the most advanced in the industry. Numerous trade-offs had to be made before the logical equations for the machine could be written. Trade-offs were made between functional requirements, computer organization, MOS speed, IC size, IC complexity, number of leads per IC, number of IC types, and total number of IC's.

To give you a feel for the class of computer which has been built, I will touch on some of its salient features (Chart 1).

250 KC CLOCK RATE
PARALLEL, SINGLE ADDRESS ORGANIZATION
24 BIT WORD SIZE - INSTRUCTIONS & DATA
3 INDEX REGISTERS
4 INTERRUPT CHANNELS - WITH LOCKOUT
35 INSTRUCTIONS - MOST INDEXABLE
8 SEC ADD INSTRUCTION TIME
108 SEC MULTIPLY INSTRUCTION TIME
4K MEMORY - EXPANDABLE TO 32K
4 SEC MEMORY CYCLE TIME
24 IC'S IN CENTRAL PROCESSOR UNIT
8 IC TYPES IN CPU

CHART 1—MOS-GP characteristics

We were quite conservative when we evaluated the speed capability of the MOS circuits. In particular, our design was based on a 250 kHz clock rate using a patented 4-phase gating scheme. The gating scheme allows fairly complex logic equations to be evaluated at each level of gating but only allows, at most, four levels of gating during each clock time.

The machine operates in parallel with the conventional single address organization. A serial

machine was under consideration at one time, but it was abandoned for two reasons; it was slower and, surprisingly, it was more complex. The word size is 24 bits for both instructions and data. This came about principally from its intended use as a navigation computer. Three hardware index registers are provided, although one is pre-empted for use by the interrupt system. Four interrupt channels with programmable lock-out are provided. They operate on a rotational priority basis. Thirty-five instructions are provided, including a 108 μ sec multiply, a 108 μ sec sum of products multiply, and a 112 μ sec divide. Most of the other instructions require 8 μ sec. The main exceptions are the shift type instructions which are of variable duration.

The computer is designed to address a 32K word memory, however, we currently are using a 4K word core memory. This core memory happens to have a 2 μ sec cycle time, although the computer really only requires a memory with a 4 μ sec cycle time to meet the stated instruction execution times. In the not too distant future we expect to operate the MOS-LSA's at 1 MHz clock rate, at which time we will require a memory with a 1 μ sec cycle time. The central processor unit is mechanized with 24 MOS-LSA's of 8 types (Chart 2).

The computer organization is fairly conventional, using an accumulator, a lower accumulator, an operand buffer register, a program counter,

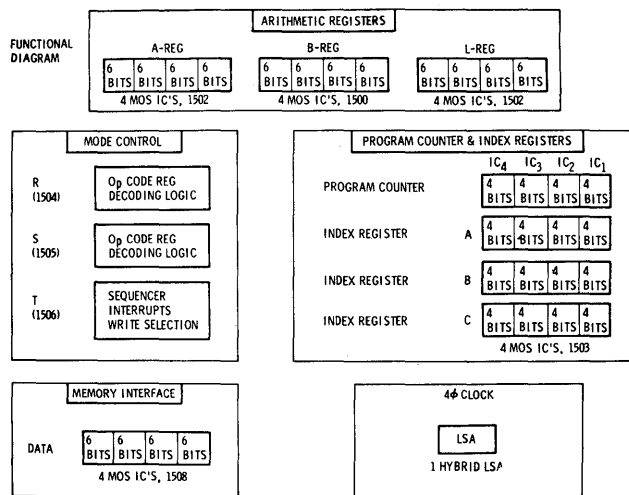


CHART 2—Arithmetic and control section, MOS-GP

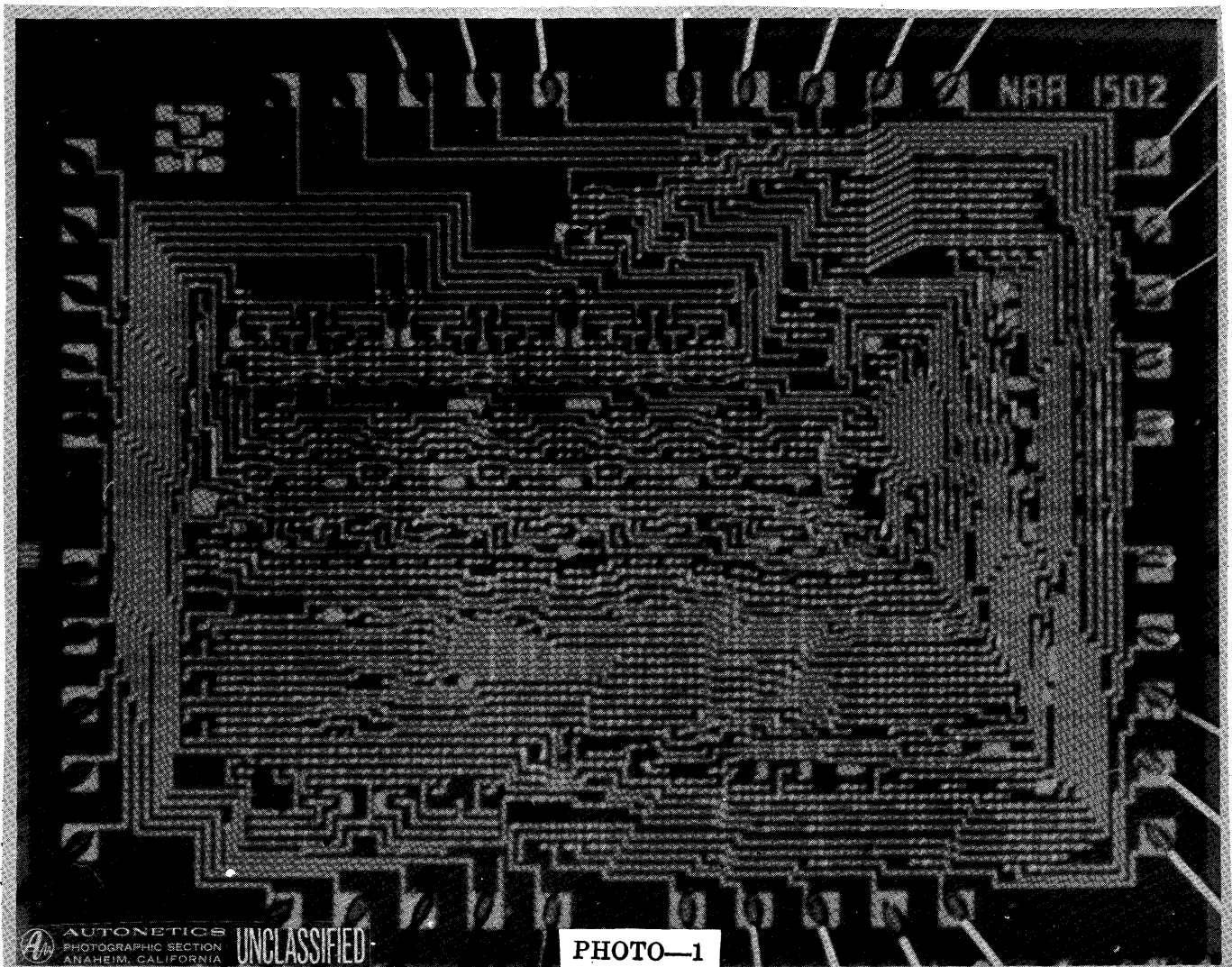
and three index registers. Four 1502 LSA's are used to mechanize the 24-bit accumulator. Four more 1502 LSA's mechanize the 24-bit lower accumulator. Four 1500 LSA's mechanize the 24-bit buffer register. Four 1503 LSA's mechanize a 16-bit program counter and three 16-bit index registers. One each of the 1504, 1505, and 1506 LSA's mechanize the mode control. These 19 LSA's really constitute the arithmetic and control portion of the computer. In addition to these LSA's, the Central Processor Unit (CPU) card also has four 1508 LSA's to mechanize a core memory interface and one hybrid LSA to mechanize the 4-phase clock.

The 1502 (Photo #1) mechanizes 6 bits of an accumulator and is used in this machine to mechanize both the A and the L registers. It has four control inputs with signals which are exclusive; that is, the signals do not occur simultaneously. The ADD control causes the 1502 to add the number presented on a first set of operand input lines to the number held in the 1502's internal register. The sum is left shifted one binary place as it is inserted into the 1502's internal register. This somewhat peculiar operation was designed to optimize the multiply and divide instructions.

The add and subtract instructions require a compensating right shift which is the function of the SAR (shift A right) control line. In addition to the ADD and SAR control lines a CLR (clear) control line will clear the register to zero, and CPY (copy) control line will cause the register to copy the number presented on a second set of

operand input lines. The latter function was included principally to provide certain transfers of data between registers during multiply and divide operations.

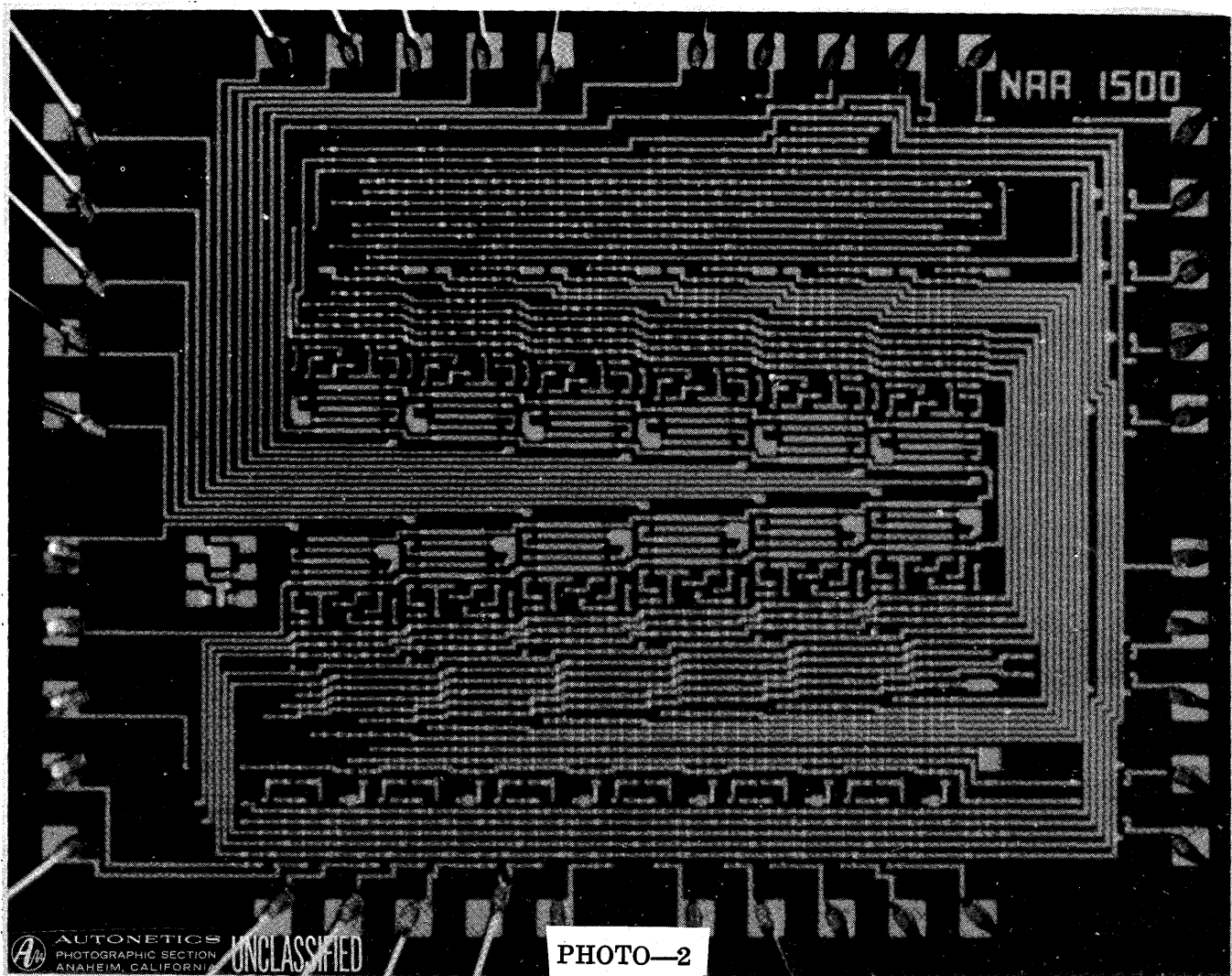
The compensating right shift which is required when executing an add instruction is performed before the left-shifted add, rather than after the left-shifted add. This provides the time required to complement the operand during the execution of a subtract instruction. The actual complementing is done in the 1500 LSA's. The 1502 also mechanizes a "look-ahead" carry scheme. The actual add (left-shifted) is performed in one clock time—that is four levels of gating. In fact, during divide operations the computer develops quotient bits at the rate of one per clock time. The 1502 LSA has 658 MOS-FET's. The die size is 110 mils by 140 mils.



The 1500 LSA, (Photo #2), mechanizes 6 bits of the buffer register. The buffer register copies memory and supplies operands to both the A and the L registers. For instructions where masking is required, the masking is performed as the operand is copied into the buffer register. For instructions where complementing of the operand is required, the complementing is performed between the buffer register and the drivers which supply the operand to the A register or the drivers which

supply the operand to the L register.

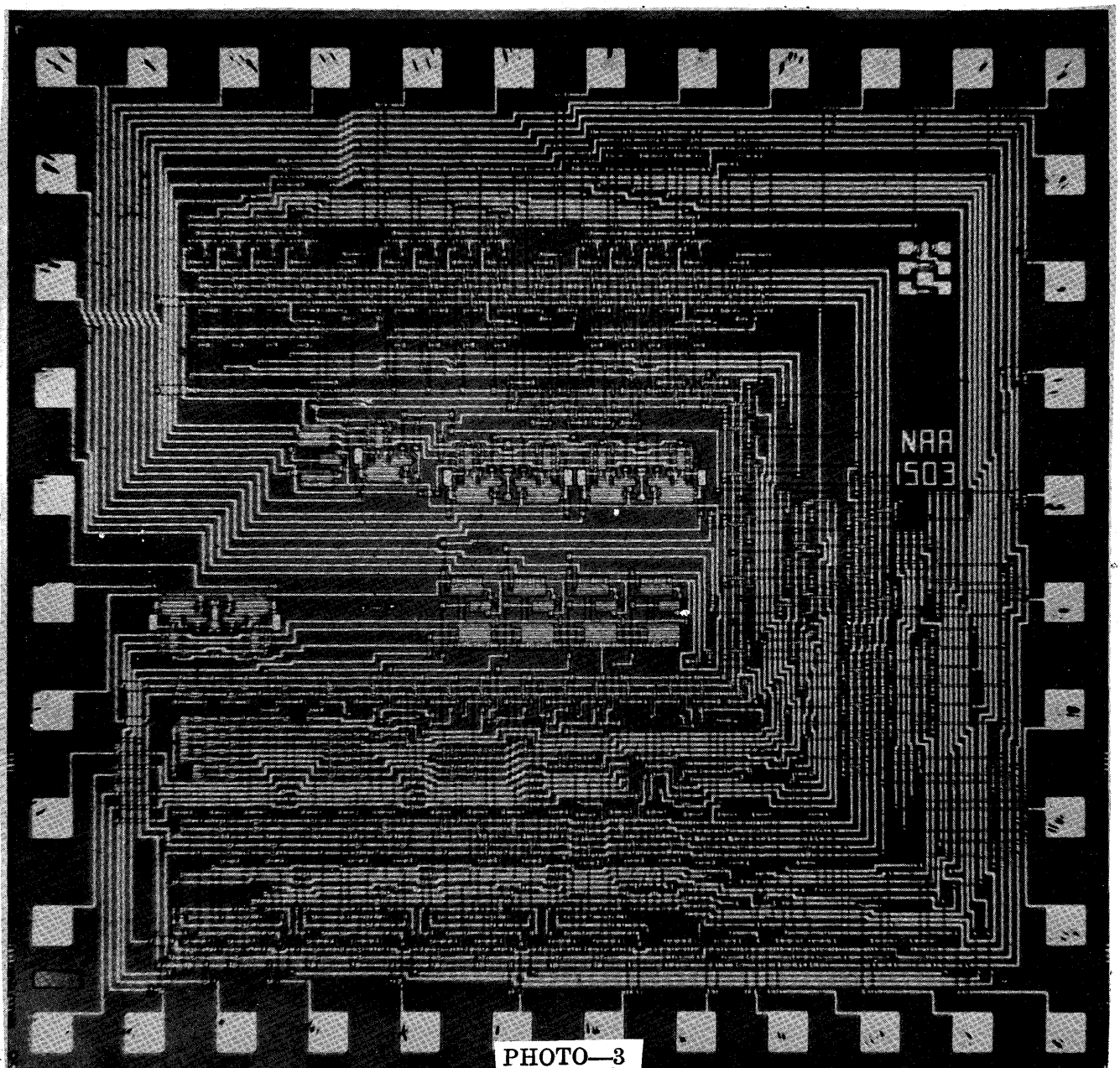
In addition to these functions, certain "dummy" operands are supplied to the A and/or L registers at certain times. One example of this is when the L register is made to act like an up/down counter during multiply operations by supplying a "dummy" operand of +1 or -1 unit. The 1500 LSA has 393 MOS-FET's. The die size is 110 mils by 140 mils.



The 1503 LSA (Photo #3) mechanizes four bits of the program counter and four bits of each of the three index registers. The functions performed by the 1503 LSA are tied in very closely to the overall structure of the machine. Its main function is to provide the addressing information required to operate the memory. The program counter takes care of instruction addressing by counting as required and by copying the appropriate address when a transfer instruction is executed. Operand addressing is simply a matter of copying the contents of the address field of the instruction word if no indexing is required. When indexing is required, the contents of the appro-

priate index register is subtracted from the contents of the address field of the instruction word. Each of the three index registers can be loaded, read out, tested for zero and decremented, and loaded with the complement of the contents of the program counter. The last function is used to set up a subroutine linkage.

In addition to these functions, the 1503 LSA provides memory addressing during a bootstrap fill mode and cooperates with input/output LSA's to set up the interrupt address when an interrupt request is honored. The 1503 LSA has 1053 MOS-FET's. The die size is 160 mils by 170 mils.

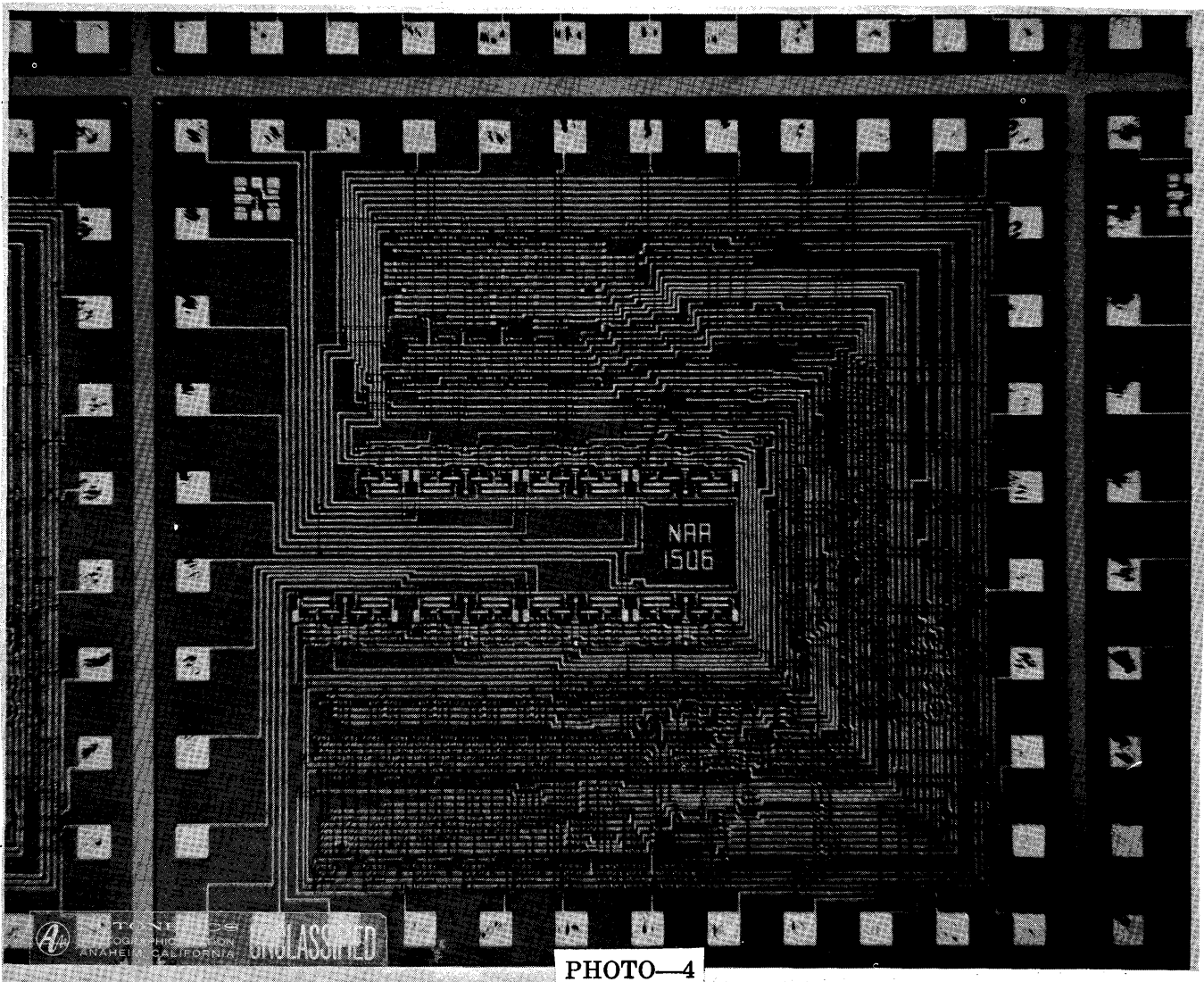


The 1506 (Photo #4) mechanizes the basic control signals required for timing. A five stage counter is used to provide the timing for long instructions such as multiply, divide, and the shifts. All other instructions have only a first clock time of execute and a last clock time of execute. Indexed instructions have an indexing clock time inserted before their first clock time of execute.

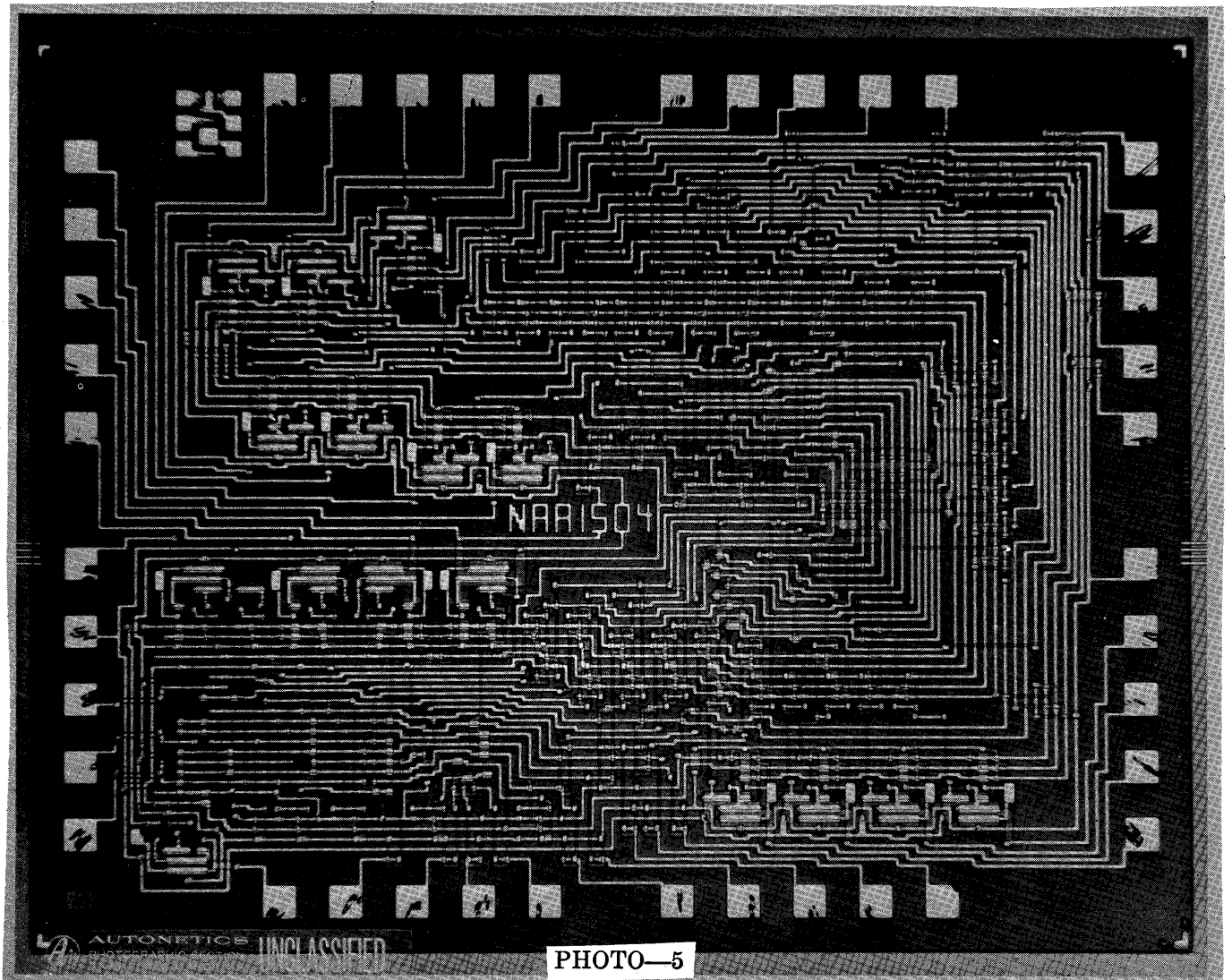
The 1506 LSA also mechanizes four interrupt channels with programmable lock-out. Simul-

taneous interrupt requests are resolved by using a rotational priority scheme. When an interrupt request is honored, the appropriate channel is enabled and an interrupt which is timed to the current instruction sequence is issued.

The 1506 LSA also mechanizes a number of miscellaneous control signals. The memory write-select signals are included in this group. The 1506 LSA has 1016 MOS-FET's. The die size is 160 mils by 170 mils.

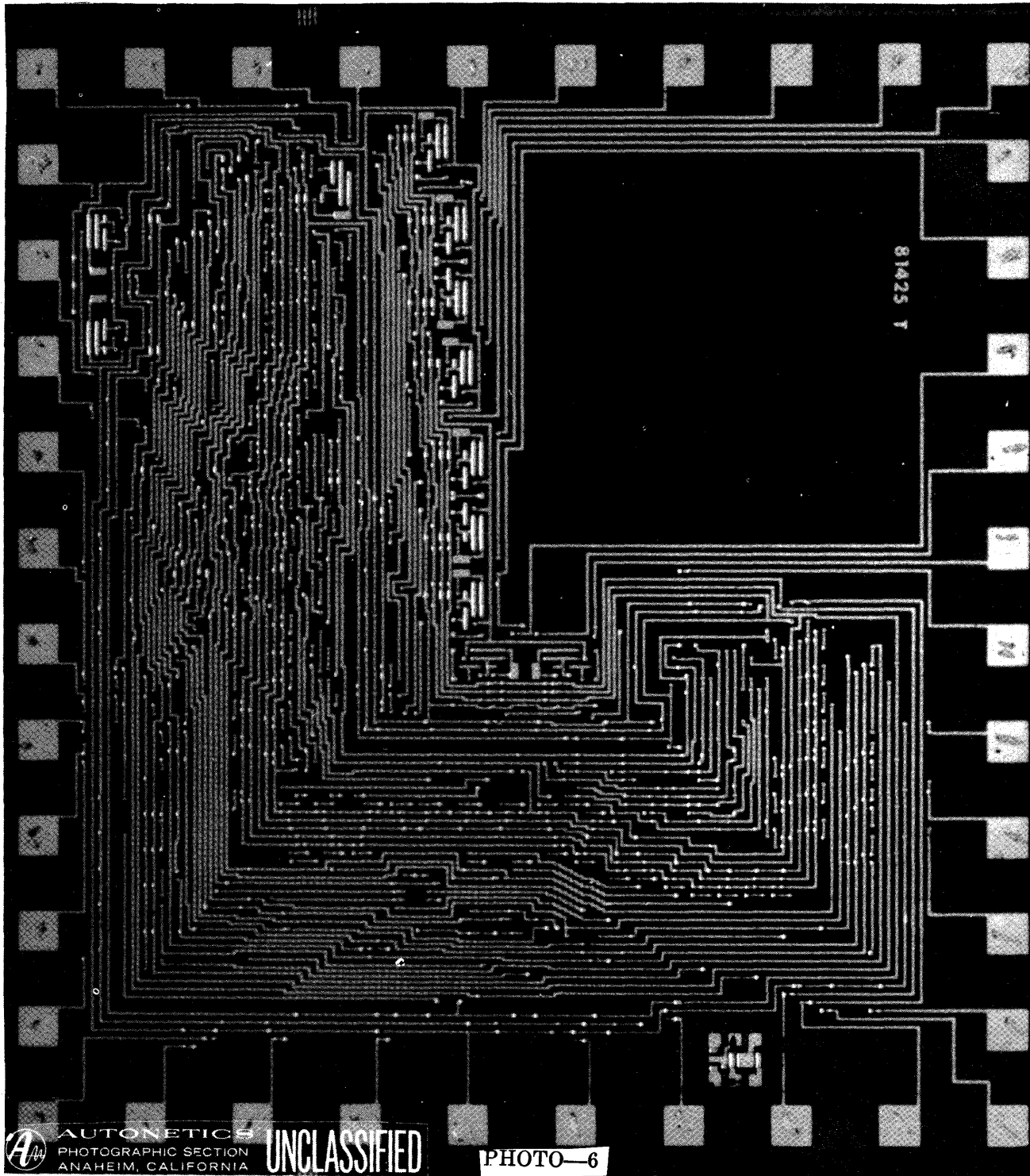


The 1504 LSA (Photo #5) generates some of the basic mode control signals related to indexing, writing into memory, and executing transfer instructions. In addition, it generates some control signals related to the execution of arithmetic instructions. The 1504 LSA has 614 MOS-FET's. The die size is 110 mils by 140 mils.



The 1505 LSA (Photo #6) generates most of the control signals related to the execution of arithmetic and shift type instructions. It also generates the low order carry data required for arithmetic operations in the A and L registers.

In addition, it processes the high and low order data bits of the A and L registers for arithmetic and shift type instructions. The 1505 LSA has 838 MOS-FET's. The die size is 160 mils by 170 mils.



81425 T

 AUTONETICS
PHOTOGRAPHIC SECTION
ANAHEIM, CALIFORNIA

UNCLASSIFIED

PHOTO-6

The 1508 LSA (Photo #7) mechanizes six bit positions of a core memory interface. The circuits employed are considerably different from the 4-phase circuits used in the other LSA's. The 1508 LSA performs a selection of one of four data sources. A sample and hold function is then performed to demodulate the 4-phase type signal. A

level shift function is then performed to make the signal compatible with the current bi-polar levels. Finally, the 1508 LSA provides the drive required by the bi-polar system and at the same time provides the capability for inverting the output. The 1508 LSA has 142 MOS-FET's. The die size is 88 mils by 104 mils.

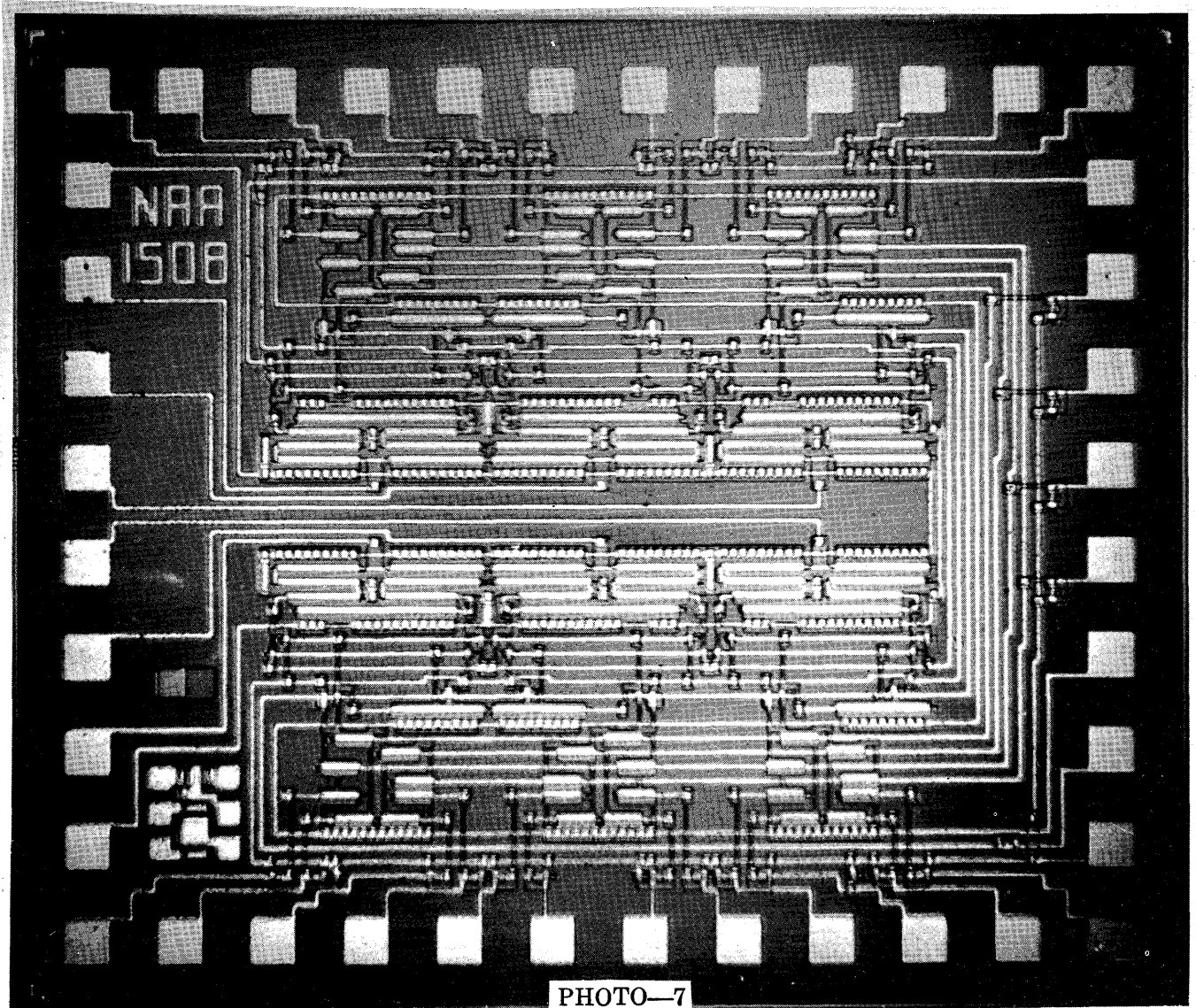
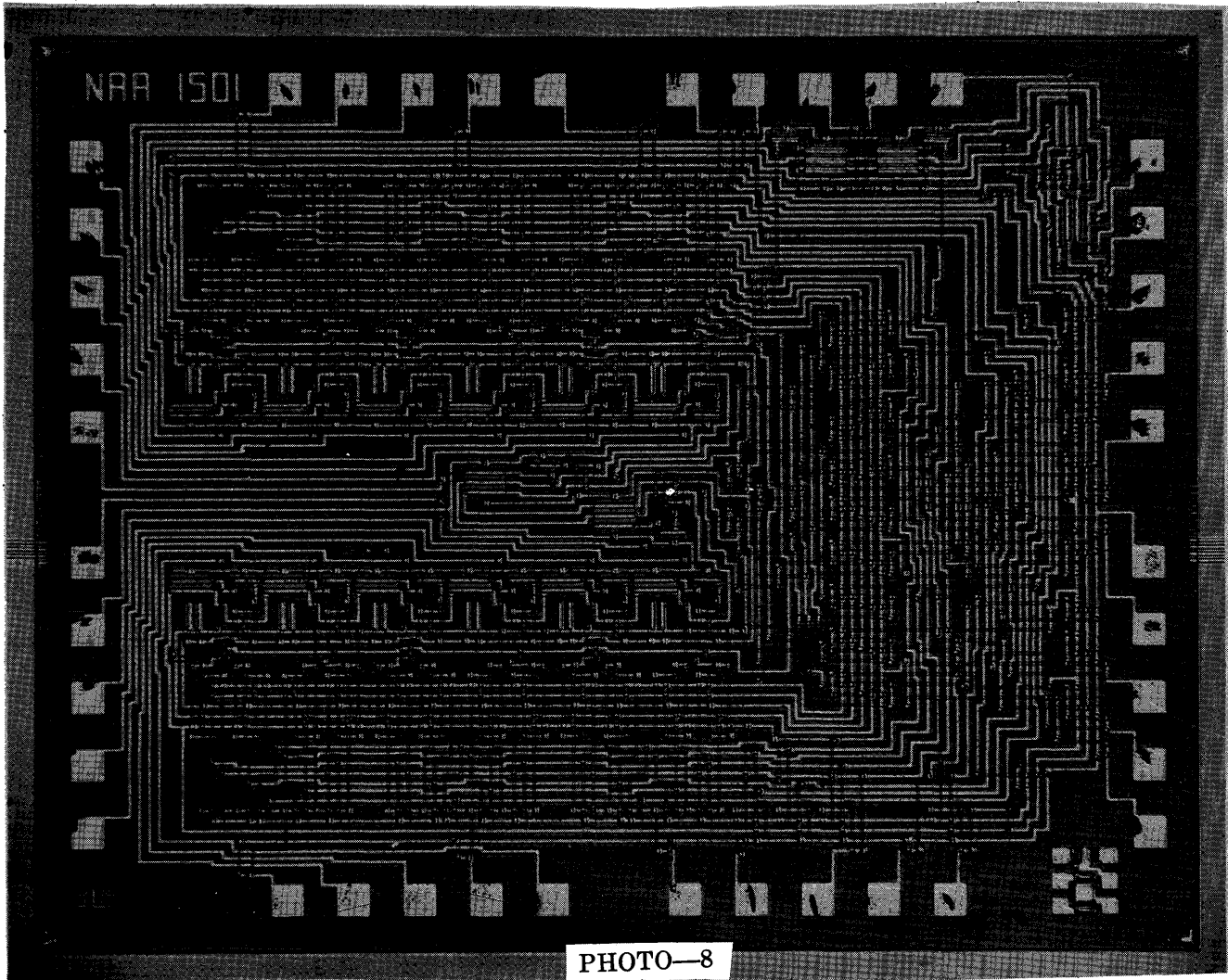


PHOTO-7

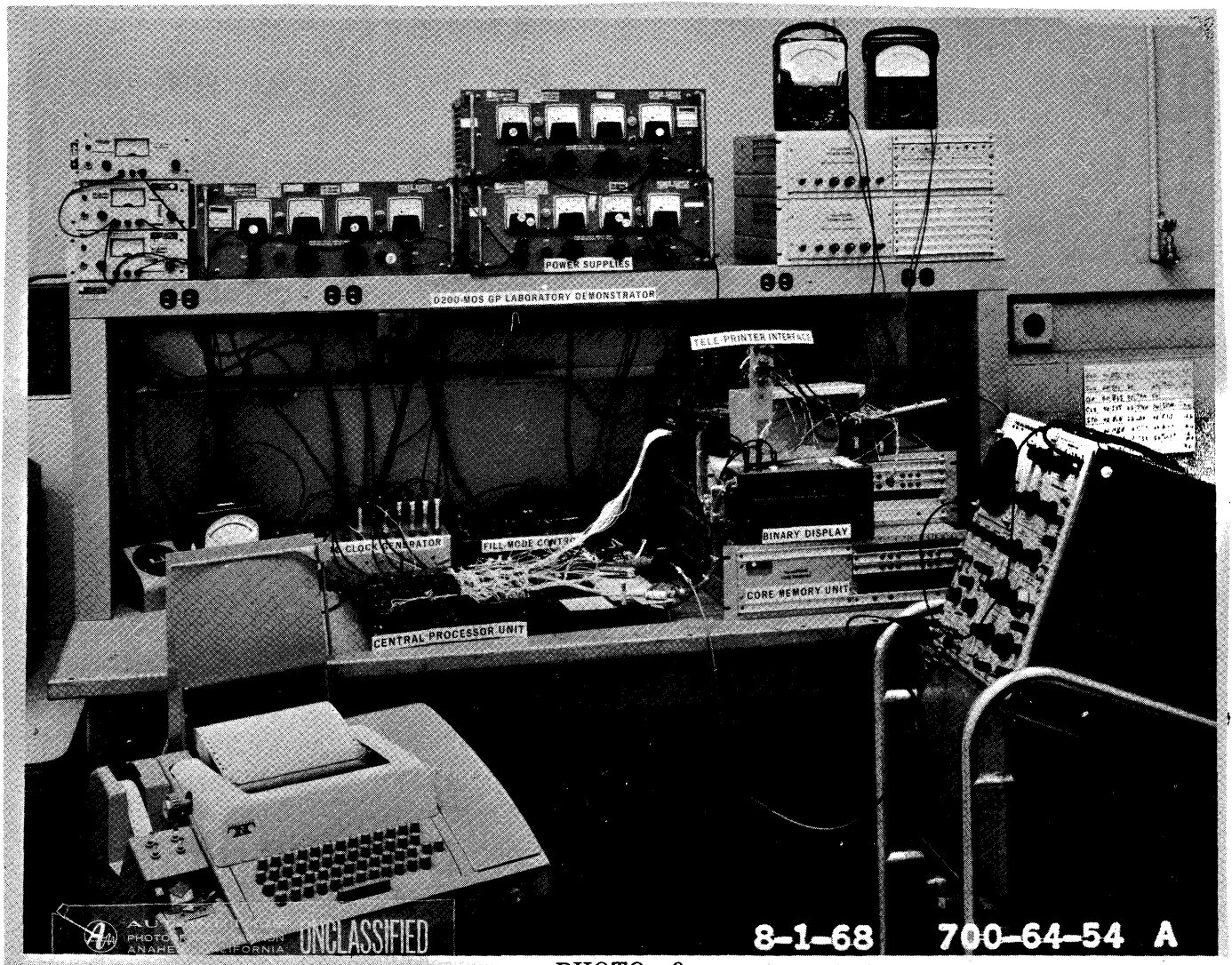
Currently, not much has been done in the input/output area, however, one LSA has been developed specifically for the input area. In particular, two 1501 LSA's (Photo #8) mechanize a buffered input channel. This circuit was included with the development of the CPU, since it was required to

load programs and data into the memory. The 1501 LSA has 732 MOS-FET's. The die size is 110 mils by 140 mils. No specific output circuits have been developed, however, a few existing circuit types have been successfully applied to satisfy current output requirements.



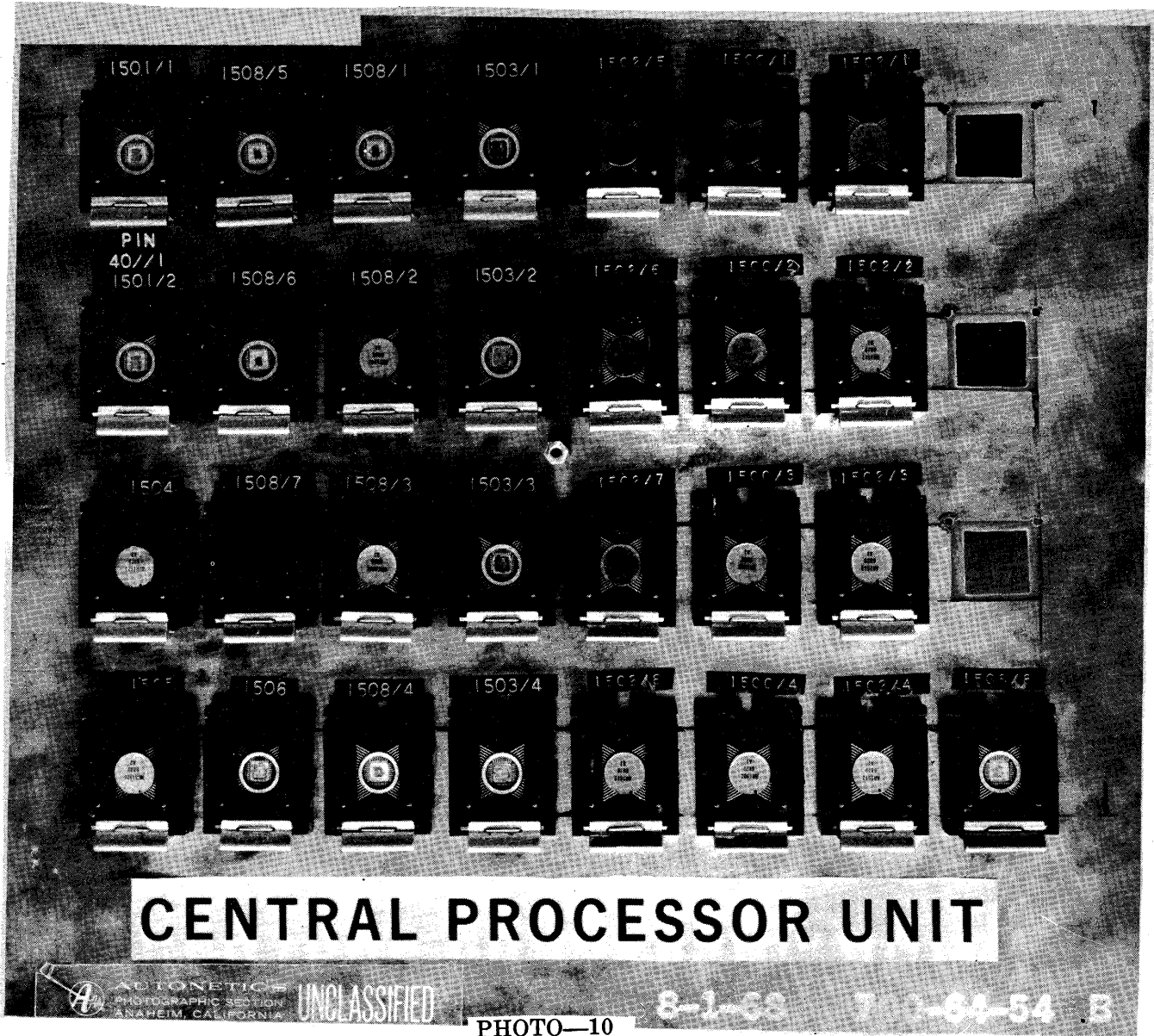
The breadboard computer (Photo #9) is small, and, like most breadboards, it is a bit of a hodge-podge. Most of the bulk is made up of lab power supplies, commercial memories, and miscellaneous

boxes. The central processor, together with one input channel, consists of one large board with interconnected LSA's.



The LSA's are clamped into holders (Photo #10).
A mock-up (Photo #11) shows how the com-

puter will be packaged for cockpit mounting in its initial application.



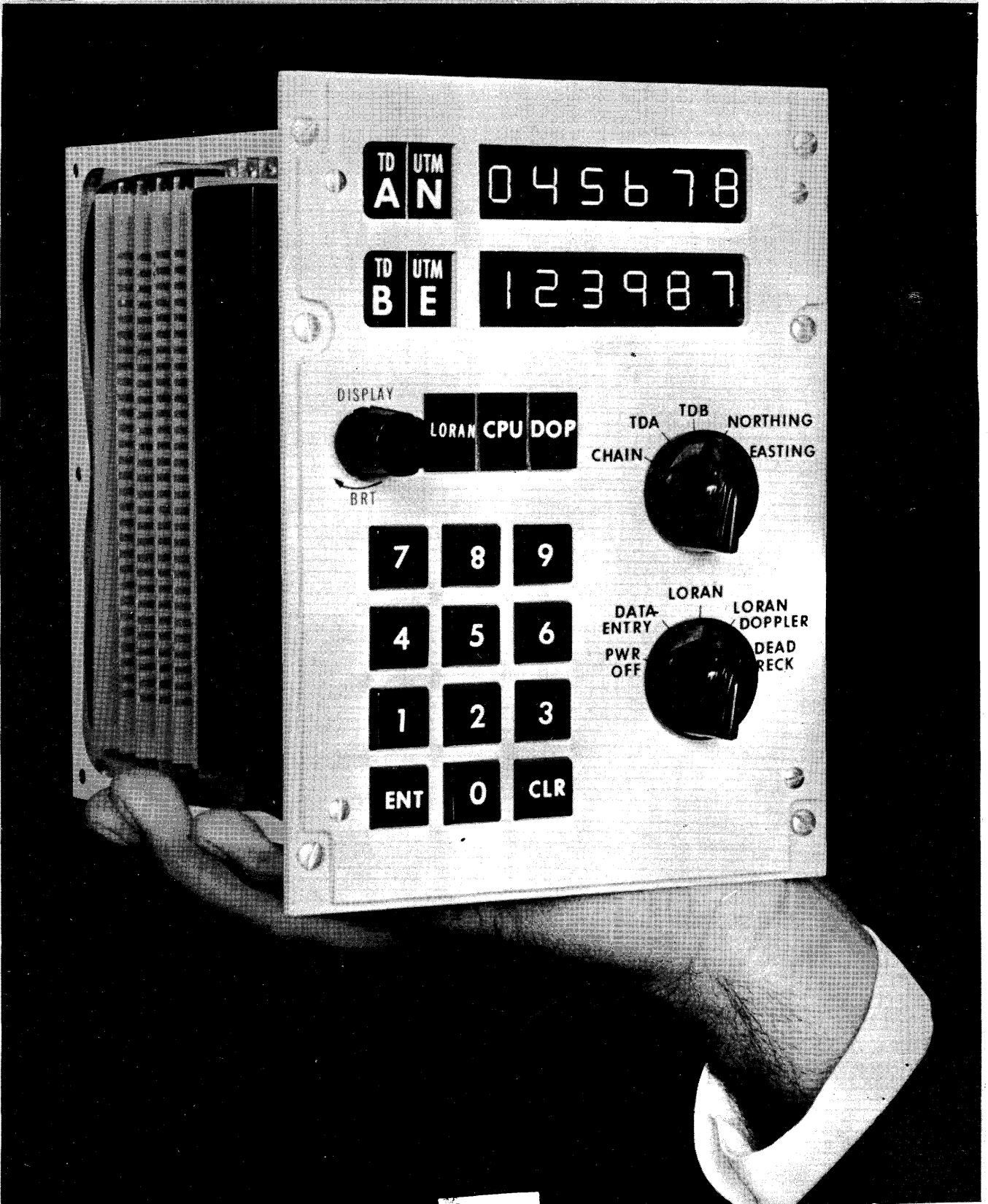


Photo #11

Hardware / software interaction on the Honeywell model 8200

by THEODORE F. HATCH, JR. and
JAMES B. GEYER

Honeywell, Incorporated
Waltham, Massachusetts

INTRODUCTION

The purpose of this paper is to describe the multiprogramming and multiprocessing features provided by the Honeywell Model 8200 and its Mod 8 Operating System. Many hardware features are included on the Model 8200 to enhance its multiprogramming performance, and these are utilized by the Mod 8 Operating System to reduce system overhead, to provide efficient usage of the computer's throughput capacity for any program mix, and to simplify the user's job of scheduling programs.

The Model 8200 is intended for large-scale computer installations having many business-oriented applications involving a high volume of input/output data manipulation and processing. Both local and remote job entry are provided. The monthly rental of a Model 8200 ranges from \$38,000 to \$75,000.

Model 8200 Hardware

Multiprocessor

The Honeywell Model 8200 consists of two central processors, an Input/Output Controller, and two or more memory modules which are shared among the processors (see Figure 1). The memory size ranges from 256K characters (two modules) to 1,024K characters (eight modules). Memory addresses are interleaved across all modules, and the separate modules cycle independently and can be accessed simultaneously. The memory cycle time is 750 nanoseconds for 4 characters.

The nonprogrammable Input/Output Controller operates asynchronous to the central processors

and can accommodate data transfer rates ranging from 1,333 KC to 3,833 KC. The peripheral control unit capacity is 48 trunks or 96 trunks, and the maximum number of simultaneous data transfers is 16 or 34.

Unlike most multiprocessing computers, the Model 8200 contains two dissimilar central processors — one word-oriented (the Multiple Processor) and one character-oriented (the Support Processor). The Multiple Processor treats 8 characters as one 48-bit word and executes 250,000 average 3-address instructions per second. This processor, which is upward-compatible with the Honeywell H-800 and H-1800 computers, performs hardware-controlled multiprogramming of up to eight job programs plus a monitor or executive program. The Support Processor performs variable-length, 2-address instructions at an average rate of 100,000 instructions per second. This processor is essentially the same as the Honeywell Model 4200 Central Processor. Both central processors can take advantage of memory overlap when more than the minimum memory modules are available.

Peripheral devices

The Honeywell Model 8200 has extensive peripheral device handling capacity, encompassing unit record, communications, and high-speed data-transfer equipment.

These peripheral devices can be attached to the Input/Output Controller via three (optionally, six) I/O sectors; each sector can accommodate 16 peripheral control units. All of the I/O sectors except one can sustain a total transfer rate of 500 KC, divided among up to 6 of the attached

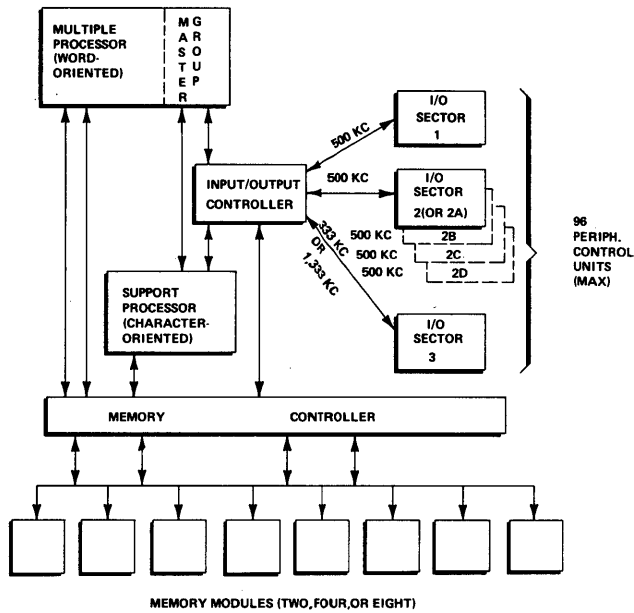


FIGURE 1—Honeywell 8200 system block diagram

control units. The remaining sector can sustain a transfer rate of 333 KC divided among up to 4 control units, or 1333 KC for a single control unit. Thus, as many as 8 (optionally, 17) high-speed tapes can be active, or 5 (optionally, 11) high-speed disk transfers can be in progress concurrently.

Hardware features

The Model 8200 has many features that simplify programming of the Monitor and enhance the multiprogramming capability of the system. Among these are (1) hardware multiprogramming, (2) Supervisor Mode (3) memory protection and relocation, and (4) peripheral device protection and reassignment.

Hardware multiprogramming

Multiprogramming sequencing in the Multiple Processor is illustrated in Figure 2. The Multiple Processor has the facility to execute as many as nine programs concurrently under control of eight program groups and a master group. A three-address-instruction format is used whereby each instruction describes a complete operation, such as "add A to B and store the result in C." The result is that each instruction is a self-contained entity, leaving no partial results in the accumulator or in common operational registers. Each of the eight program groups and the master

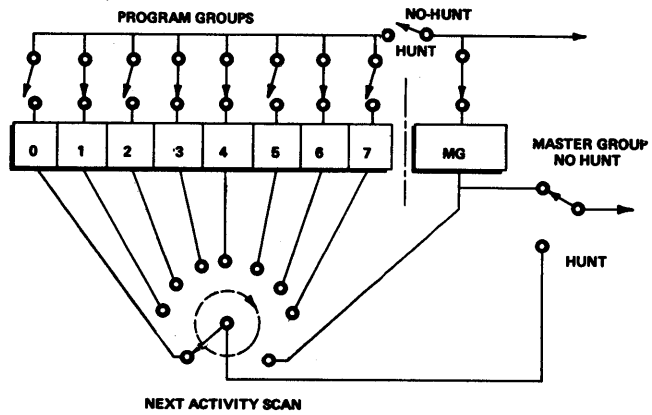


FIGURE 2—Multiprogram sequencing in the multiple processor

group in the Multiple Processor has a set of 32 addressable registers, including sequencing counters, index registers, and work registers (see Figure 3). Since each program group has the facilities to direct and execute a program, there is no requirement for loading or unloading of registers — even of the sequencing counter — when control is switched from one program to another. Hardware in the Multiple Processor executes an instruction from one active group and then scans for the next active group and executes an instruction from it. This scan process is completely overlapped with execution of the preceding instruction. When any program becomes unable to proceed (pending completion of some I/O operation), it is automatically removed from the multiprogramming sequence until the peripheral device is available; during this interval, the time-slices

AU1 AU2	ARITHMETIC CONTROL REGISTERS
SC	SEQUENCE COUNTER
CSC	COSEQUENCE COUNTER
SH	SEQUENCE HISTORY REGISTER
CSH	COSEQUENCE HISTORY REGISTER
UTR	UNPROGRAMMED TRANSFER REGISTER
MXR	MASK INDEX REGISTER
X0—X7	INDEX REGISTERS 0—7
R0—R7	GENERAL PURPOSE REGISTERS 0—7
S0—S7	GENERAL PURPOSE REGISTERS 0—7



FIGURE 3—Multiple processor control group

Peripheral protection and reassignment

The Input/Output Controller contains a peripheral steering table which is set up by the Monitor when a job program is allocated peripheral devices (see Figure 5). The contents of the peripheral steering table can be changed only by master group, and are used by the Input/Output Controller to determine which processor should receive each peripheral interrupt. The table is also used by the Input/Output Controller at the start of an I/O request issued by the Support Processor to verify that the processor has been granted permission to use the addressed peripheral control unit. If it does not have permission, the Input/Output Controller causes the Support Processor to make a call to master group. Thus, without the overhead cost of peripheral instruction verification by software, programs running in the Support Processor are prevented from using peripherals not allocated to them.

When a Multiple Processor program group wishes to initiate a peripheral operation, it executes an I/O request (Peripheral Control Order). The execution of an I/O request causes the issuing group to be "stalled" (taken out of the multiprogramming sequencing) and an interrupt to master group to be made. Before the interrupt is made, however, the Multiple Processor, using the peripheral address and drive number in the I/O request, looks at an entry in the peripheral protection/reassignment table set up by the Monitor at job allocation time. This entry contains a lock that must be matched by the requesting group's key; if they do not match, the I/O request interrupt is turned into an illegal-peripheral-device interrupt. If the lock and key match, the I/O request is copied into a reserved area of main memory within the Monitor. During this copying, the peripheral address and drive number are taken from the table entry containing the lock value. The buffer address is added to the base register before being stored. The I/O request interrupt is then made to master group which picks up the verified and reassigned I/O request and processes it.

The hardware verification and reassignment of an I/O request eliminates the need for this function to be done by software, thus speeding up the process by at least a factor of 10. All software now need do is verify the seek address for mass storage devices. The memory address of the buffer is checked by the Input/Output Controller

during data transfer as described above.

Peripheral reassignment is useful when a program has been dumped in a checkpoint procedure and then later reloaded. When the program is re-run, the physical data files need not be mounted on the same physical control units. The Monitor simply makes the correct entries in the peripheral protection/reassignment table.

Program scheduling and multitask control

The scheduling function on a large-scale computer involves the dynamic management of many resources, including memory space, auxiliary storage space, peripheral devices, central processor usage, and input/output channel usage. Scheduling can be divided into two broad categories: the allocation of main memory, peripheral devices, and mass storage areas to each program (referred to herein as Program Scheduling) and the sharing of central processor and I/O channel time among those programs that are currently resident in memory (herein referred to as Multi-task Control).

Program scheduling

Program Scheduling in the Mod 8 Operating System consists of two phases. The first phase selects a set of programs to be executed concurrently in such a way that as many available resources as possible are made busy. The second phase assigns unused tape and disk pack drives for the mounting of volumes needed later and informs the operator of these assignments.

In order to maximize the utilization of system resources in a multiprogramming environment, it is beneficial to have a large number of diverse programs continually available for execution; also, the facility for allocation of resources among programs must be flexible. The Mod 8 Operating System allows submission of new job requests from a variety of sources at any time, thereby making it convenient for the user to constantly replenish the supply of programs waiting to be scheduled. Further, a variety of program types can be accommodated and scheduled for concurrent execution, thus increasing the probability that the available program mix can be multiprogrammed efficiently. For example, programs for the word-oriented Multiple Processor, programs for the character-oriented Support Processor, Honeywell 800/1800 programs using compatibility mode, and communications message-pro-

cessing programs can be run at the same time. Each program is restricted to its own set of assigned resources by hardware protection features and is unable to interfere with other programs or the Resident Control System.

Assignment of resources

Flexibility in the assignment of Model 8200 resources is indicated by the following capabilities.

1. Any portion of memory can be assigned by the Mod 8 Operating System for use by either the Multi Processor or the Support Processor.
2. Any tape unit can be assigned to any program regardless of the nature of the program and regardless of which tape control it is connected to.
3. A single disk device can be shared by several programs regardless of which processor they use.
4. Each program occupies a contiguous block of memory, which can be dynamically re-assigned (by moving the program and changing its base register) in order to free up a large contiguous unassigned area.

Scheduling algorithm

Each job submitted to the operating system is given a scheduling priority by the user. The Mod 8 scheduling routine, which is called into operation every time new jobs are submitted or there is a change in resource availability, proceeds one priority level at a time:

1. The highest priority level is considered first;
2. Available resources are calculated for this priority level (the details of this calculation are a function of the priority level, since some priorities can acquire memory currently used by certain lower priorities, and since no priority level can use resources on which a higher level has a claim);
3. Requirements of programs at this level are determined (currently running programs have no outstanding requirements);
4. The most profitable program* is selected

*A program is considered "profitable" to run now if it requires the use of currently unused resources and does not also require the use of any currently busy resource that cannot be shared efficiently. Thus, a program that requires only memory plus disk files on a currently unused disk control would be considered more profitable than a program that requires memory plus a tape file on a busy tape control. Also, a program that requires a sequen-

- from among those for which enough resources are available; if none remain in this level, the scheduling routine goes to step 6;
5. Main memory, program groups, and tape and mass storage work areas for the most profitable program are allocated; the status of available resources is changed to reflect this allocation; and step 4 is repeated;
6. Step 2 is performed for the next lower priority level if any programs remain that haven't been processed.

Program protection and isolation

A necessary attribute of any multiprogramming system is the protection of each program and the Resident Control System from all other programs.

In the Model 8200, program protection is complete and absolute. It is accomplished through a combination of hardware and software, and, significantly, this protection costs very little overhead time. The protection features also assist in detection of programming errors, since any accidental attempt by a program to stray beyond its boundaries is immediately trapped by hardware and/or software.

The hardware elements involved in program protection are listed below:

1. Lock-and-key protection (see Figure 4).
2. A peripheral protection/reassignment table (see Figure 5).
3. All illegal or potentially dangerous instructions are trapped to the resident Monitor and generate an interrupt to master group.
4. An endless uninterruptible loop within a program, which prevents all other programs from operating, is automatically terminated by hardware after a set interval of time.

The hardware elements above are complemented by the following software elements to provide program protection in an optimal manner.

1. The Multiprogram Control (MPC) instruction is useful, within a program which uses more than one program group, to initiate and terminate asynchronous tasks and to test for task completion. Because of the danger that program A might affect a task within program B, all MPC instructions are trapped (by hardware) to master group.

tially-accessed disk file on a busy disk pack device would be considered unprofitable because of the probable wasted arm positioning time between the two active files on the same disk.

The Monitor masks out any reference to groups not belonging to this program, and performs the remaining actions specified in the instruction.

2. Monitor functions are requested by the monitor call instruction, which identifies the desired Monitor function and specifies parameter information. The Monitor verifies that each monitor call is legitimate to the extent that it will not cause the Monitor to get into a loop, partially destroy itself, or affect any other current program. In addition, during loading of program segments, the Monitor employs the hardware memory protection feature to prevent one program from loading into another program's area.
3. Since there may be several independent data files stored on the same mass storage device, the peripheral protection/reassignment table is not sufficient to prevent a program from referring to another program's file. However, the limits of each mass storage file used by every running program are stored by Monitor in resident (protected) memory, and whenever a program issues a peripheral order to mass storage (which is trapped to master group), the specified mass storage address is compared to the known limits for this file. Thus, references to peripheral files not assigned to a program are prevented.

As may be observed from the preceding hardware and software features, the protection of each program from all others is a result of the *isolation* of each program (see Figure 6). A program cannot affect *anything* — e.g., main memory, control memory, peripheral devices — outside of its own assigned resources.

Multitask control

Multitask Control is concerned with the dynamic allocation of central processor time and input/output channel usage among several programs or tasks which are all resident in memory. In the Model 8200, each task is executed either by the Multiple Processor or by the Support Processor. Support Processor tasks, which are generally either communications message control tasks or media conversion tasks (such as card-to-disk), are executed by a conventional interrupt-driven monitor which turns on the active task with the highest priority after processing each interrupt (such a priority directed system is referred to

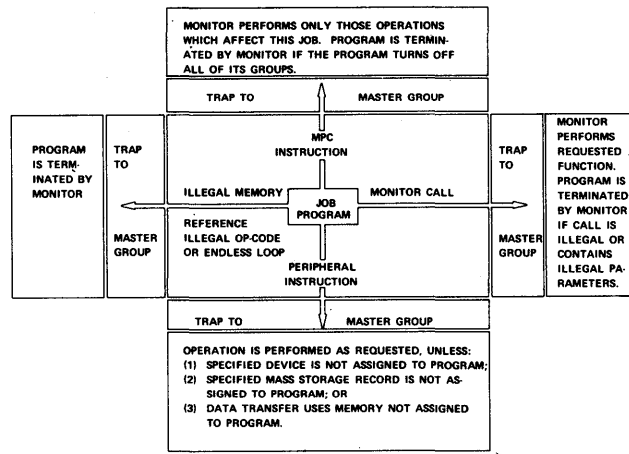


FIGURE 6—Isolation of individual programs

herein as “vertical multiprogramming”). Multiple Processor tasks, on the other hand, are each assigned a program group and are executed in an interleaved manner by hardware (this method, in which one instruction is executed from each task in cyclical fashion, is referred to as “horizontal multiprogramming”). Although a single Multiple Processor program can include more than one task (each being assigned a separate program group), the terms “multiprogramming” and “multitasking” are used interchangeably in this discussion.

As the following discussion will show, vertical multiprogramming is advantageous when it is necessary to give priority to some programs that have external timing requirements. This is the reason that remote communications lines, for example, are controlled from the Support Processor. However, it is our contention that horizontal multiprogramming, as provided by the Model 8200 Multiple Processor, is well suited to maximizing the throughput of a group of typical business-oriented data processing programs that make heavy use of relatively fast input/output devices.

For any program p to which specific data files on specific devices have been assigned, there is a number R_p which is the ratio of the central processor time used by the program to the time during which one or more of its files are active (assuming that the program is running alone). For a program which issues peripheral orders with regularity, the program is peripheral-bound if $R_p < 1$, and it is processor-bound if $R_p > 1$.

For an installation having mostly processor-bound programs ($R_p > 1$), the only method to pro-

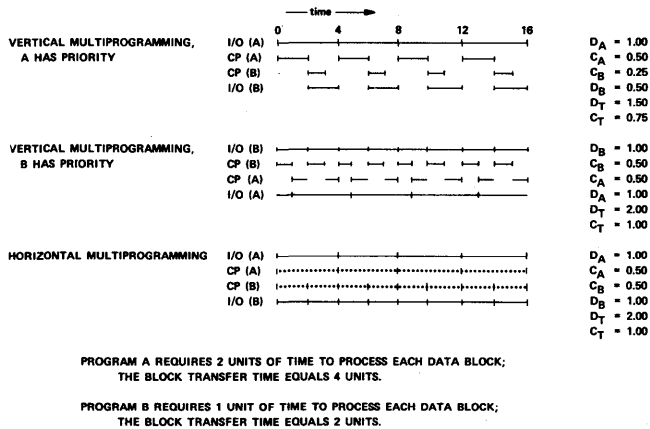


FIGURE 7—Effect of wrong priority choice

vide increased throughput is to get more central processors or a faster one. However, if the installation has a number of peripheral-bound programs, its overall throughput may be significantly improved by adding peripheral devices and memory and using a multiprogramming operating system. The degree of improvement will depend on the nature of the programs and on the characteristics of the operating system.

If several peripheral-bound programs for which $\sum R_p \leq 1$ are executed concurrently under an ideal multitasking system, each program should proceed at its solo speed (i.e., its speed when run alone). For such a set of programs, horizontal multiprogramming will perform as well or better than a vertical system because the necessary central processor operations are always executed in less than the available time (which is determined by peripheral devices); the only difference between the two systems is the order in which the central processor instructions are executed.

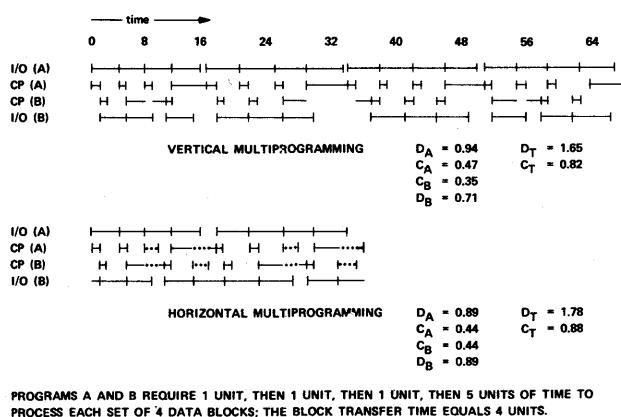


FIGURE 8—Programs oscillate between peripheral-bound and processor-bound

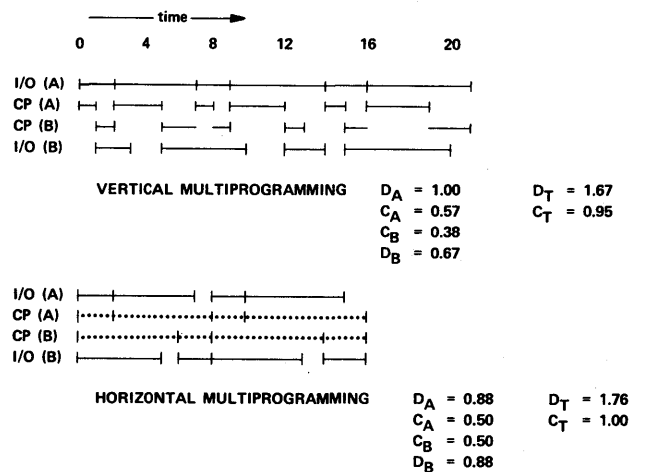


FIGURE 9—Program combination is processor-bound

A more interesting situation occurs when programs for which $\sum R_p \geq 1$ are executed concurrently or when one or more of the programs oscillates between being peripheral-bound and being processor-bound. Figures 7-9 illustrate three simple cases in which horizontal multiprogramming of the type performed on the Model 8200 Multiple Processor provides better overall throughput than fixed-priority vertical multiprogramming. In these three figures, D_A and D_B are the fractions of a single data channel used by programs A and B, respectively, while C_A and C_B represent the percentage of central processor utilization by each program. D_T is the total number of data channels used, and C_T is the fraction of central processor time used by the two programs together.

Figure 7 is a case in which the priorities of the two programs affect their overall performance in vertical multiprogramming; only if program B is given a higher priority than program A are both programs executed at their solo speed. In Figure 8, both programs operate at less than I/O transfer speed because they oscillate between being peripheral-bound and processor-bound; in this case, multiple buffering (i.e., more than two buffers per file) would allow them to run at I/O transfer speed. Figure 9 is a case where programs A and B could not both be expected to run at solo speed under any system because $R_p > 1$. The program given highest priority under vertical multiprogramming runs faster than it runs under horizontal multiprogramming; however, the total throughput of both programs is better with hori-

zontal than with vertical multiprogramming.

The main advantages of the horizontal multiprogramming provided by the Model 8200 Multiple Processor are (1) task switching is accomplished by hardware with zero time overhead and (2) combinations of programs that make heavy use of peripheral devices tend to obtain better overall throughput with horizontal than with vertical multiprogramming. The above examples describe several cases where horizontal multiprogramming performs better than vertical multiprogramming, even assuming zero task switching overhead for both methods. Proper choice of priorities when using vertical multiprogramming may provide better throughput for one of the programs when compared to its performance under horizontal multiprogramming; however, the combined throughput for all programs still would not exceed that for horizontal multiprogramming. Further, when complex data processing programs are involved, it is very difficult to know enough about the characteristics of these programs to intelligently decide which priorities to assign in order to produce the highest overall I/O throughput, and usually the set of programs which will run concurrently changes from day to day so that the choice of relative priorities would have to be done on a daily basis. In practice, when using vertical multiprogramming, programs are assigned priorities on the basis of their assumed importance and time requirements, if any; as a consequence, overall I/O throughput suffers.

In the Mod 8 Operating System, user-specified priorities are used to influence the original scheduling of programs (i.e., their starting times), but once started, the use of horizontal multiprogramming tends to provide maximum overall I/O throughput.

Organization of the resident control system

The Mod 8 Resident Control System (Physical I/O, Monitor, and parts of Logical I/O) is organized to provide the following characteristics and capabilities:

1. Adaptive to different amounts of reserved memory (minimum of 8,192 words) and to different work loads;
2. Functionally expandable (by adding overlay routines) without increasing the reserved memory requirement;
3. Usable concurrently by several programs;
4. Responsive to input/output requests on a

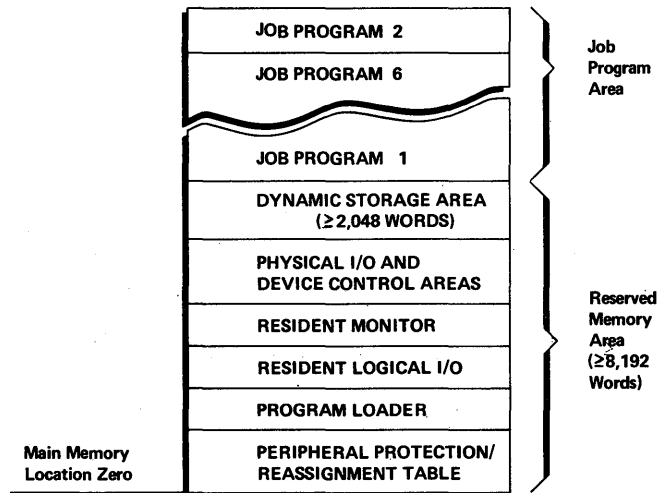


FIGURE 10—Mod 8 memory organization

more urgent basis than requests for less time-dependent services.

The first two capabilities above have been achieved by dividing the reserved memory area into two parts: a permanent storage area and a dynamic storage area (see Figure 10).

The permanent storage area consists of tables used by hardware and/or software (peripheral protection/reassignment and resource allocation tables), the resident routines that execute in master group (Physical I/O and parts of the resident Monitor), and resident routines that execute in one of the program groups (i.e., several commonly used Monitor routines and resident Logical I/O).

Only the most commonly used Monitor routines, such as the Loader, the Job Input File Reader, and the Job Output File Writer, are permanently in memory. All others are overlay routines which are loaded into any available part of the dynamic storage area from mass storage when needed. However, if an overlay routine is called frequently it is likely to be in memory when needed, in which case it will not be reloaded from mass storage (overlay routines do not destroy themselves during execution, and hence may be re-executed without being reloaded). Also allocated from within the dynamic storage area are work areas required by the overlay routines during execution. Because these work areas are allocated independently from the overlay routines themselves, it is possible for overlays to call other overlays in a subroutine fashion without all such overlays being in memory at the same time. For inactive overlays to which control must be re-

turned later, only the work areas need be retained in memory.

The dynamic storage area is also used to store job description information for current and waiting jobs. The fewer jobs there are, the more memory is available to contain overlay routines and work areas. Because no fixed division is made between memory available for overlays and memory available for job descriptions,* the Monitor is able to optimize its performance in the context of the current workload.

The third and fourth capabilities above have been obtained by dividing the functions of the resident control system into three priority levels and by executing the highest priority functions in master group no-hunt mode, the next highest priority functions in master group hunt mode, and the lowest priority functions in program group mode (see Figure 11).

Physical I/O

I/O requests and interrupts are handled in master group, running in no-hunt mode. When an I/O service request is executed by an active program group, that group is put in the 'stalled' state by hardware. The I/O request is then verified by hardware using the peripheral protection/reassignment table. For mass storage devices, a check must be made by software to verify that the reference is within a file area available to the requesting program. The Input/Output Controller uses the memory protection system of locks and keys, so the buffer address and range of a data transfer request need not be checked by Physical I/O.

For non-mass storage devices, the group is left stalled until the I/O request is accepted by the peripheral device. For mass storage, the group is stalled until the preceding order for the same file has been completed. In general, this means that a program can only make use of one or two data buffers per file. Double buffering is all most applications need, allowing processing of a block to overlap data transfer. If the amount of central processor time to process each block fluctuates greatly and many buffers should therefore be used to smooth the operation, a second task can be set up to read the blocks in advance.

*Although there is no fixed division between these areas, the job description information is never permitted to expand to the point where the largest overlay could not fit, but is saved on mass storage instead.

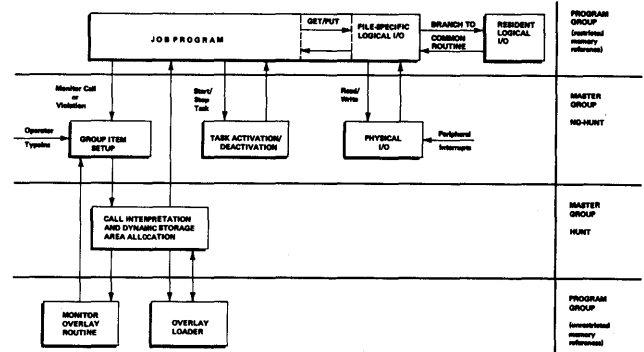


FIGURE 11—Operating modes of the Mod 8 resident control system

Physical I/O is organized as follows. Each peripheral control unit in the system has a control area reserved for it in the permanent storage area. This serves as a work and status area for a device specific routine which can handle any number of peripheral control units of the same type. This allows the number and address of each peripheral control unit to be specified at system initialization time, as long as the system has been generated to handle the required types of control units.

Mass storage requests are queued by drive. For each drive, a seek order is issued for the data transfer on the closest cylinder to the current one in a fixed direction (i.e., first from circumference to center, then from center to circumference, etc.).* For those drives which are presently positioned to the desired cylinder, orders are issued in a round robin fashion.

An I/O request is verified upon receipt and held in a protected memory area until the peripheral order is issued; thus, the peripheral order can be issued immediately when the peripheral device is ready to receive it.

Having the file protection operation handled by Physical I/O enables the Mod 8 Operating System to allow the user to code his own I/O routines for special devices or for special file handling and yet not worry about device protection. The user also does not have a high overhead imposed on his own I/O operations.

Monitor

Although all interrupts within the Multiple Processor are received by master group no-hunt mode, those which are not critical for maintaining I/O throughput are passed along to master group

*This is the SCAN procedure described by Denning.¹

hunt mode and processed by the Monitor. These "non-critical" interrupts consist of:

1. Monitor calls (programmed requests for services such as load segment, print memory, rerun program, terminate program, typeout on the console, schedule media conversion job, etc.)
2. Operator typeins (once the operator has completed an unsolicited typein, to indicate some external condition or request some system action, master group hunt mode is called upon to interpret the typein and initiate the appropriate processing)
3. Program errors detected by hardware (baricade violations, illegal or privileged instructions, etc.)
4. Peripheral device errors associated with a particular program and detected by Physical I/O.

Although interrupts to master group no-hunt are queued by hardware, calls passed from master group no-hunt to master group hunt mode (to process "noncritical" interrupts) are queued by software. This is accomplished via a table of "group items," into which master group no-hunt stores call information and from which master group hunt reads that information. Any time that resident Monitor is activated in master group hunt mode, it scans all of the group items to determine which one(s) is awaiting action. There is one group item for each of ten possible call sources: Multiple Processor program groups zero through seven, the Support Processor, and the operator's console. The Group Item Setup routine stores status information within the appropriate group item and prevents further interrupts from the same source (e.g., if a program uses groups 3 and 4 and group 3 issues a monitor call, both groups 3 and 4 are turned off to prevent further calls until this one has been processed).

Thus, as many as ten calls to the Monitor may be outstanding at the same time, and the processing of these calls does not prevent further high-priority (peripheral) interrupts from being handled.

Even though the response time to a segment load request (for example) is not as critical as the response to a peripheral interrupt, it is still important in a multiprogramming system to avoid tying up the Monitor for extended periods in behalf of one program if this prevents other programs from obtaining Monitor services. Thus, if the Monitor is performing a diagnostic memory

dump for program A (which may require 5 to 10 seconds), it is desirable for the Monitor to be able to load a segment of program B (which may require only 0.5 seconds) during this time. In other words, the Monitor itself should be multiprogrammed.

This is accomplished in the Mod 8 Operating System by utilizing Multiple Processor program groups (normally the same group which issued the monitor call) to perform monitor functions, and by providing a dynamic storage area within the reserved memory area which can contain more than one monitor overlay routine.

Master group hunt mode merely turns on a Multiple Processor program group to execute a monitor function and then continues its scan of the group items in parallel with that monitor function. Since many of the monitor routines consist of reentrant code, this means that several different monitor functions, or even the same monitor function, may be executed simultaneously for several different requesting programs, and that the Monitor doesn't appear "busy" to a calling program until it has run out of space in its dynamic storage area.

Logical I/O

The Logical I/O system has the main objective of keeping to a minimum the number of instructions necessary to process a data file. This is done as follows.

1. Privileged instructions are not used in Logical I/O, thus limiting the number of interrupt-causing instructions. This is accomplished by coding most of Logical I/O as re-entrant coding residing in the permanent storage area. The hardware memory protection allows this area to be read by any group but changed only by master group. Thus, any or all groups can be executing the resident Logical I/O coding at the same time.
2. File-specific coding is included within the job program. This coding is specialized when the file is first opened, to the requirements of the file format and device type. Putting the basic operations of blocking/unblocking and generation of I/O requests for sequential processing in file-specific coding, instead of doing these operations with a common interpretive subroutine, improves execution time in this area manyfold. Device independence is obtained by specializing a

file at open time when the device type must finally be known. Reruns are still handled easily since the peripheral address used by the job program does not change on restarting even when the actual device has a different address; the address is changed only in the peripheral protection/reassignment table.

The usual once-per-file functions (open, close, force end of volume) are coded as overlays to save memory. These functions also do not use privileged instructions and are re-entrant.

The coding to handle nonsequential file processing is handled by interpretive subroutines to save on memory space. This GET/INSERT function coding resides in two basic overlays (one for direct access file format, the other for indexed sequential file format) and are kept in memory only while a file of the corresponding format is open.

SUMMARY AND CONCLUSION

The Honeywell Model 8200 and the Mod 8 Operating System have been introduced as a multiprocessor/multiprogramming computer system. The interaction of hardware and software has been described, especially in the areas of protection and multiprogramming, as have the Operating System's scheduler and resident control system. It is argued that the horizontal multiprogramming used on the Model 8200 provides greater I/O throughput with less attention from the user than does the more conventional vertical multiprogramming.

The potential throughput of a Honeywell Model 8200 system is considerable, ensuing from the speed of its two central processors (250,000 and 100,000 average operation per second, respectively) and the transfer rate of its input/output system (up to 3,833,000 characters per second). However, the degree to which any data processing system achieves its potential (i.e., the fraction of its capabilities that are devoted to the customer's work and not either idle or involved with house-keeping) is a function of the job mixture and of the operating system overhead. Because as many as eight programs plus communications and media conversion tasks can run simultaneously on a Model 8200 (and these may be either checkout or production programs), it is highly probable that a well-balanced (heterogeneous) job mixture will be available for processing at all times. The hardware and software features that have been described, which result in low central processor overhead and rapid response to requests for input/output and monitor action, make it practical to operate as many as five or six typical programs in parallel with very little degradation in the individual performance of each. Thus, the Model 8200 with the Mod 8 Operating System should come closer to achieving its full hardware potential than do competing machines in its market class.

1 P J DENNING

Effects of scheduling on file memory operations
1967 SJCC Proceedings p 9

2 A C HIRTLE

A look at the model 8200

Honeywell Computer Journal Vol 2 No 1 p 5 April 1968

Measurement and analysis of large operating systems during system development

by D. J. CAMPBELL and W. J. HEFFNER

General Electric Company
Phoenix, Arizona

INTRODUCTION

Some months ago, representatives of one of the better known software houses contacted us with this proposal: They wished to sell us a tool for advancing our techniques in developing real-time systems. Their techniques allowed the exact reproduction of any observed sequence of real-time events. Thus, when a particular sequence caused a system error, the sequence could easily be reproduced so that the error could be analyzed and corrected, and the correction verified. A powerful tool, indeed.

Yet we were not interested. We explained that the particular errors which would be most effectively analyzed by this technique did not cause us very much difficulty in our systems.

While the presentation was a failure in the eyes of the software firm, it verified our belief that very few standard packages exist to assist in the measurement of operating systems. Our problem was not reproducing sequences of events, but rather simply finding out what, in fact, was going on inside the system.

What is measurement and why measure?

By measurement of any system, we mean the gathering of quantitative data on the behavior of that system. For instance, timing runs on programs are measuring program performance. Likewise, simulations of systems are measuring tools of that system, since they give performance or behavior data on the system studied. The accounting information for user jobs is a measuring tool of an operating system; it gives measures of system resources used in running user jobs. Even the lowly memory dump is a measuring tool of a system because it shows how the system behaved.

Due to their complexity, operating systems are particularly difficult to measure. In many cases, an operating system will correctly run each user job, but still be

grossly inefficient in using the computing power of the system.

Cantrell³ describes a measuring technique that found system inefficiencies which had caused approximately 30% degradation in system performance for almost two years. And nobody suspected that it was there! There really is a large potential pay-off in adequately measuring an operating system, despite the difficulties of applying the yardstick.

We have been engaged in the development, maintenance, and extension of multiprogramming, multiprocessing operating systems for five years. During that time we have produced three major rewrites of the operating system for the same large-scale computer system. The latest version, called GECOS III—a totally integrated, on-line, remote batch, and time-sharing system—is described in recent literature.^{1,2} Our experience in the development of these systems also has led to the development of a series of techniques for the measurement and analysis of the behavior of our operating systems. One of these techniques has been described by Cantrell and Ellison.³ This paper discussed additional measurement techniques, limitations of each, values of each, and specific lessons learned by applying these techniques to GECOS III.

Types of measurements

For purposes of discussion, it is convenient to group measurement techniques into two classes: hardware techniques and software techniques. The hardware techniques may be further subdivided into standard hardware features that may be used for measurement purposes and special hardware instrumentation, specially added for the sake of analysis. Software techniques generally can be divided into three classes: simulation models of the system, measurement processes interior to the system, and finally, exterior measurement pro-

cesses imposed on the system.

Hardware measurements

Hardware techniques have a long history. Anyone who used the IBM 650 can remember the address stop switches. When these were set, the computer would come to a halt when the indicated address was reached. Another more sophisticated hardware technique was the Trapping Transfer mode of the IBM 704. In this mode, the computer interrupted itself each time a transfer instruction was to be taken. Instead of transferring, it passed control to a fixed cell where a user program recorded the event, and passed control afterward to the correct transfer point. Today most systems have similar hardware features; however in many cases they are operative only from maintenance panels by product service personnel.

These techniques have passed out of the repertoire of the software developers. The necessity of manual intervention made the address stop useless. The hardware trapping schemes suffer from three major disadvantages. First, the processor burden of analyzing each transfer can multiply running times by factors of three or more. Second, even if one were willing to pay the tremendous cost of processor (and elapsed) time, the huge volume of data produced can often prove to be quite indigestible; for example 700 pages of tracing information and somewhere the one mistaken path. It could take days to wade through to find the interesting place.

While sufficient money, time and patience may overcome these two disadvantages, the third disadvantage of transfer trapping is crushing for any real-time or interrupt driven system. The act of trapping, analyzing and recording each trapped event so changes timing within the system that system behavior without trapping cannot be duplicated when trapping is used. There is many a tale told by programmers debugging I/O supervisors about "hardware" errors that would mysteriously go away when trapping was used to find the error. Of course, what happened was that as soon as trapping was turned on, the interrupts that gave rise to the error occurred at different places within the system. It was the early experiences of this sort that gave rise to the myth of the sensitivity and consequent difficulties of real-time systems.

Another set of hardware measurement devices, present on almost every computer, and often ignored by programmers, is normal error-faulting procedure. As an example, overflows occur with orders of magnitude less frequently than transfers, therefore, it is possible to tie a system measuring function onto the occurrence of the fault. For instance, at least one FORTRAN object-time debug package is made to operate by replacing the

instructions to be trapped by special faulting instructions.

Of the many special hardware devices added to a system for measurement purposes, no single tool is of greater potential power and versatility than the oscilloscope. Unfortunately, few programmers have the requisite knowledge of the hardware logic to make intelligent use of the device, even if the computer manufacturer would let him poke around inside the cabinets.

There is one special hardware device that we have found effective. This is a "black box" that can be attached to the processor that passively examines each instruction to be executed. This device has a built-in counter to record the occurrence of any given data pattern in the instructions; it may be used to record the number of times a particular instruction, say Multiply, is performed. Or it can count the number of times a particular cell is referenced. Since it is passive, the device does not appreciably alter the timing of the system. The major disadvantage of this kind of a monitor is the set-up time. There is rewiring to do each time the function is to be changed. Cantrell and Ellison³ describe a method for obtaining this information with a software monitor without inordinate overhead, and this method we believe is superior to the hardware monitor.

In summary, the various hardware devices for recording system monitoring information are of limited interest to the system developer. Generally, they suffer from lack of flexibility and, in some cases, slowness. However, as a course of last resort, such methods find their usefulness when all else fails. Apparently, combinations of hardware-triggered software packages, like the FORTRAN debug package previously mentioned, offer a good solution to tracing problems.

Software measurements—simulation

In turning our attention to the software measurement tools, the first topic to be discussed is simulation models. Today, there is perhaps no single technique more in vogue than simulation. As part of the development of the GECOS III system, a simulation model was developed. Although much effort and expense was put into the model, it proved to be of limited usefulness. Perhaps the specific difficulties we experienced were atypical, but it is worthwhile mentioning them as at least one case history. The major bottleneck was time. The simulation model was begun as soon as possible, but it was not debugged until some months after the skeleton system worked. Thus many of the design questions that might have been answered through the model were in fact answered by initial running of the system. Because implementation preceded simulation, the model became obsolete before it ever worked. When results began to arrive from the simulation, it was impossible to decide

if the results represented the current system or an earlier version.

On the other hand, several developers had access to a time-sharing system, and a number of simple simulations were written to check specific points. Since the designer did these to help make a specific design decision, they were done quickly and the results were used. For example, I/O requests are not necessarily done in order when latency reduction techniques are used on discs or drums. It is necessary therefore to ensure that any particular I/O demand is not forgotten forever. A simulation was done to find out the minimum time a request could be ignored without a decrease in device throughput. If outstanding requests are ignored too long, the process owning the I/O request is unduly delayed. Conversely, when an old request is forced, a longer latency than usual may result. Thus, total device throughput suffers. With a simple program we found that a request could be bypassed no less than twice the average queue length. If specific requests are forced to be serviced sooner, then total transfer rate decreases rapidly. We feel that these simulation studies were eminently successful for us.

Our conclusion on the use of modeling techniques is that ambitious large-scale models generated by professional model makers are less helpful than simpler work done by the system developers themselves. An interesting sidelight on this subject is that results from any simulation are useful only if the user actually believes in the simulation. An act of faith is required. The large, complex simulation is less likely to be understood by a developer than a simple model he constructs himself. Thus there is considerable hesitancy to change designs based on results from the large-scale simulation programs.

Internal system measurement

System recording is the second main type of software measurement. In our opinion, it is this area that is most often ignored by system developers, and one in which we believe we can make a contribution. There are four techniques of importance here:

- a) System design that allows for adequate measurement
- b) Built-in system auditing techniques
- c) Event tracing
- d) Performance analysis and recording

Let us now discuss each of these in detail.

System design amenable for measuring

The importance of the initial system design for measurement purposes cannot be overstated. For example,

unless it is possible to find out exactly where the processor spends its time, it may be nearly impossible to account for some significant amount of overhead. In the initial phases of GECOS III development, we did not distinguish between the time spent processing interrupts and the time spent waiting for interrupts to occur when all programs in the system were waiting for I/O completion. Thus, when we came to measure actual interrupt processing time, the data were not there. Consequently, a change was made to ensure the necessary distinction.

As another example of design requirements for measurement, consider the set of all programs in the system at any one time that are waiting for the processor. In an early version of GECOS, this set was defined by an elaborate set of tests conducted by the system dispatcher each time dispatching was done. It is clear that the number of jobs waiting for the processor in a multiprogramming system are a measure of multiprogramming interference. For in a uniprogramming system, the single job cannot ever wait for the processor. The length and behavior of the dispatcher queue is a most critical measure of the system. Thus it is very important to design the system so that data about the length and the wait time in the dispatcher queue can be easily measured. Our design is currently inadequate in this respect since we cannot obtain data on wait time in the dispatcher queue, although we do know the length of the queue. The same arguments can be repeated for virtually every important function in the system. For example, the behavior of the I/O queues is as important as that of the system service functions, such as reading input, peripheral allocation, and so on must be separately recorded. Thus, it is important to design each function of the system so that it may be separately analyzed and studied.

A second design provision for measurement is the inclusion of system event counters to show the number of occurrences of low-frequency events. For instance, each memory compaction or program swap is counted. Memory compaction is the movement of all jobs in core to one end or the other so that all unused memory space becomes contiguous. Swapping is the removal of a job from core, in favor of a higher priority job. A study of the number of times memory compaction took place showed us that we had to introduce a damping factor to inhibit these compactions.

When we allowed compactions to occur whenever necessary to get more jobs into core, we found that the system actually slowed approximately 20% in throughput. The system was so busy moving core about that it never got around to doing any user work. At another time in development, we found that a program priority was being set incorrectly by observing an unusually

large number of program swaps. This particular program was being swapped in and out continuously. If we did not have these built-in tools, it would have been next to impossible to see that things were going wrong inside the system, because there were no obvious exterior symptoms of these bugs, except decreased system performance.

System auditing

The next important interior measurement technique is the inclusion of adequate system auditing. To "audit" means to examine and verify, and that is exactly what we mean here. At any number of places within a system, entries are moved from one table to another or into or out of a given queue. If all is correct, the transactions are legal and each table or queue is consistent both before and after.

In many cases, it can be argued that it simply is not possible for erroneous entry to creep into a queue. It often is quite amazing to see how a rather simple error at the beginning of a process can balloon into scores of strictly illegal transactions later on.

The symptom of one of the most difficult errors we had in debugging the system was that the entry in a table of base address values was illegally zero. After several days of study, we finally found that a particular job was being doubly entered into the system and assigned two different index numbers. The job was actually allocated twice and put into execution twice. When the first copy terminated, the base address table was being cleared for the other copy. The double data had passed through at least three different internal queues, each time incorrectly and each time further complicating the troubles. No auditing was done on entries passing into these queues. Finally we were able to lay this bug to rest when we installed a series of checks on new entries in each of the queues. After this had been done, the real culprit was found and corrected within a day. We also found it necessary to install a check on one threaded list queue each time it was referenced. The list was becoming unsewn, and we couldn't find out who was doing it until we audited the list. A great deal more of this kind of auditing is needed than one might suspect.

A second variety of internal auditing that we made considerable use of was to checksum critical tables at every reference. For instance, there are tables showing available space on disc and drum units. An erroneous store into one of these tables can lead to assigning unavailable space to a file. The first time anything goes wrong is when the true owner of the file again references it, and then it is too late. By continually checking the table, a ruined table is discovered immediately,

while the footprints of the culprit are still fresh. By using this technique, our troubles have been minimal with ruining files. However, we have found it necessary to install some additional audits on these tables. When space is given back to the available pool, we added checks to verify that the space definition is within reason.

As a second part of the effort to ensure the veracity of files, we checksum all system files as they are loaded into core for execution. In earlier versions of the system, countless hours were wasted re-editing the system because we suspected a system failure occurred when the files had been written over accidentally. After spending the time to edit, all too often we then found that the bug was still there. With checksums we know that if the file loads, it is correct, and we are not distracted from the real problems by worries of overwritten files.

Event tracing

So far, we have discussed a variety of techniques used in our system to provide for a very limited form of measurement: finding bugs. Now we turn to the technique used to provide data for performance measurement. We call this technique the event trace. A brief history of the tracing methods we have employed makes the event trace more understandable.

In the first versions of the operating system, it was almost impossible to infer what had been happening prior to a system failure. Casting about for a solution to this problem, the developers noticed that all communication between modules of the system passed through a common routine—the equivalent of a FORTRAN CALL and EXIT. In this routine, it was possible to record each intermodule transfer in a circular list. Thus, at any time, the last transfers could be seen, and from this, the operation of the system could be summarized.

This trace table was a tremendous advance in easing the job of analyzing system failures, yet a number of disadvantages were found. In the first place, it was discovered that the processor time used to make these flow trace entries was inordinate in many cases. Which I/O had terminated when control passed into the interrupt handler? Which I/O was started next? Was there any error on the terminating I/O? An ordinary flow trace just can't say.

It was apparent from our studies that the need was for a trace to show the important events, or decisions, made within the system. At the same time, data appropriate to the event should be captured. We call this kind of trace an event trace because it records system events, not necessarily system flow. The following list shows the events that merit a trace entry, along with the data interest:

EVENT	DATA
IO Interrupt	Time of day, Location of interrupt
Interrupt Queue Value	Current values after interrupt
Process Interrupt	Interrupt status, pub
Connect IO	IO entry location, pub, device, command
System Module CALL	Location of call, Module name entry point
System Module GO TO	Location of go to, Module name entry point
System Module EXIT	From and To location
Dispatch to Program	Location in program, time of day
Master Mode Entry	Location, entry type
Fault	Location, fault type
Return from Interrupt Processor	Time of day
Enter Status Return	Location of IO entry, pub
Leave Status Return	Location of IO entry, pub
Slave Road Block Broken	Program number
Slave Relinquish Broken	Program number
Interrupted Program to Head of Processor Queue	Location in program, program number
Interrupted Program to Tail of Processor Queue	Location in program, program number
Call Device Module	Pub, IO request location
Start IO Error Recovery	Program number, time of day
Start Abort Processing	Program number, time of day
Start Program Swap	Program number, time of day
Start Courtesy Call	Program number, time of day
Leave IO Error Recovery, Abort, Swap or Courtesy Call	Program number, time of day
Enable Program	Program number
Start Activity	Time of day
Start Memory Compaction/Swap	Number of program to move/ swap time of day
End Memory Compaction/Swap	Program number, time of day
End of Activity	Program number, termination code
Can't allocate	Device, number required program
Shared Device Space Refusal	Device, amount requested, amount available
New Job to System	Job ID, time of day
Program Number Assigned	Program number, job ID
Job to Peripheral Allocator	Program number, time of day
Activity to Core Allocator	Program number, time of day
System Output Ready	Job ID
System Output Printing	Job ID
System Output Punching	Job ID
System Output Printing Finished	Job ID
System Output Punching Finished	Job ID
IO Channel Idle	Pub, time of day
IO Demand Queue Length	Pub, length, time of day

This list is by no means exhaustive, there are some

fifty different events that are traced by GECOS III.

As an extra degree of flexibility, each type of event trace can be turned off or on at system start-up time. Thus the trace, when fully on, is an exceedingly detailed picture of the system behavior. For ordinary purposes, many of the individual traces are turned off, giving a rougher picture of a longer time interval. As in previous versions, the trace entries are recorded in a circular table. In a production environment, all traces are turned off, this provides the greatest system speed that can be achieved. We have found that the normal traces cause a system speed degradation of only a few percent. Timing of the system is not disturbed by this.

The implementation of the trace allows easy addition of new entries and modification of the data in existing entries. Trace entries are coded in-line where desired. An execute instruction is used to test if tracing is on or off. If the trace is off, control passes to the following instruction. Otherwise, control passes to the tracing control routine where the state of the routine is saved, and then control passes back to the second instruction following the execute. An index register is set in the control routine to allow the user to transfer back into it.

In the user's in-line code, the 72 bits of trace data are placed in the accumulator and quotient registers. Since the state of his program is saved, he may destroy the contents of any register if necessary. When the data has been put into the registers, he can transfer through the index register to one of three entry points in the trace control routine. One of these points adds time of day to the data; another inserts program number and processor number; the third stores the data in the table without modification. After the data are stored in the trace table, the state of the program is restored, and control passes to the instruction following the execute code.

If some traces are off and others on, a test is made in the routine that stores the data within the trace table against the trace type presented. If that trace is off, the data aren't stored. At the same time, the execute instruction that triggered the trace entry is found and is modified into a no operation instruction. Thereafter, the trace control routine will be bypassed. Thus, no processor time will be spent generating unwanted trace entries. Figure 1 is a flow chart of these routines.

After this trace was implemented, we found it difficult reading the trace table in the octal memory dumps produced by system failures, so we wrote a routine to expand the trace into English language. The effort required to do this was modest, and has paid for itself manyfold.

Figure 2 shows a portion of the expanded trace table as included in a system dump. This figure is the beginning of a system memory dump. On the first line is the

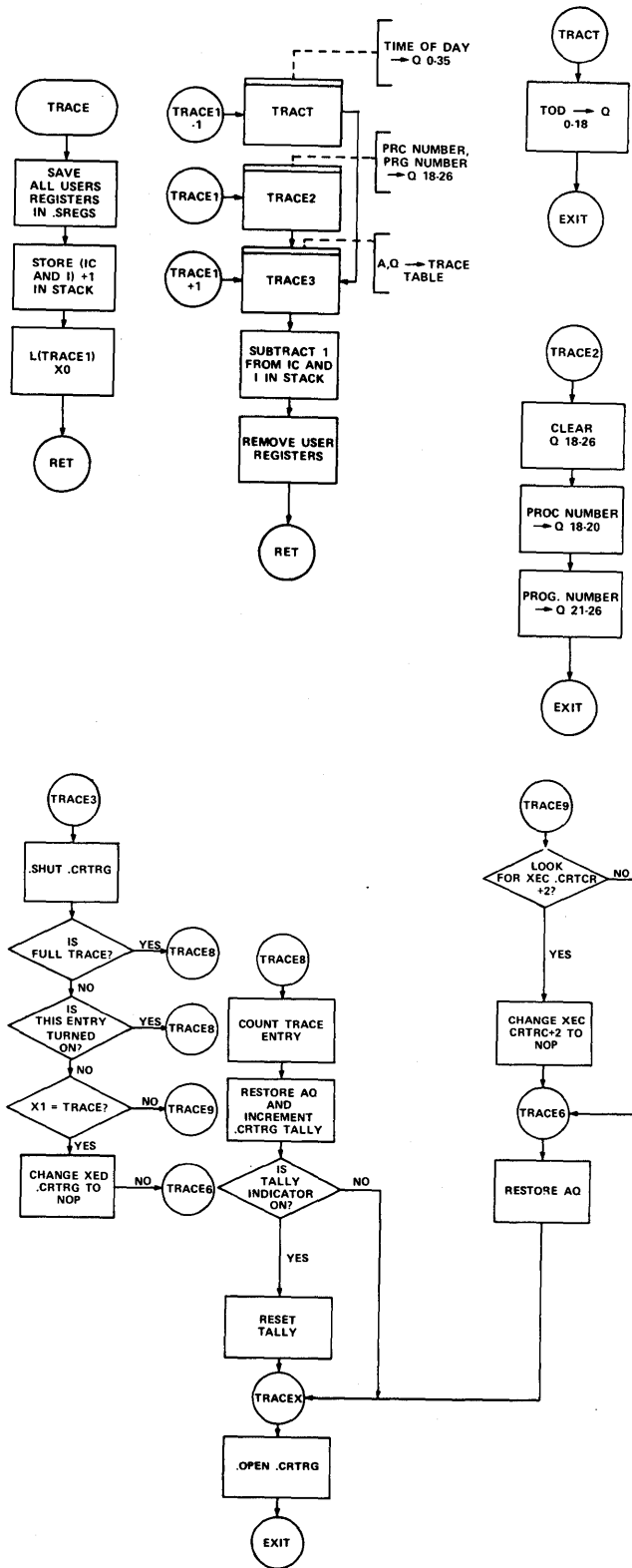


Figure 1—System trace

system and fault identification. This is followed by the register control at the time of dump in the next two

lines. Under the heading "Trace Table" is the expansion of the event trace into text. On the left hand column is the cell address of each entry. Note that there are two trace entries per line. Each entry is two words long and the addresses increase by four on each successive line.

The data that can be obtained from the event trace are useful for far more than simple system debugging. The trace provided the data for microscopic measurements of specific processes within the system. For instance, we were able to determine that interrupt processing time was within bounds. Also, we were able to verify that system load time was up to specifications.

One of the more interesting measures obtained from this data was the frequency distribution of interrupts. The dispatching rule of GECOS makes this distribution very important. A new dispatch is made after processing each interrupt. By recording the time of day of a great many interrupts, we were able to assure ourselves that we were not dispatching too often.

Performance analysis with exterior tools

While the internal counters and trace go a long way in providing the tools needed for system measurement, they do not provide a method for long term measurement. To do this, a third major type of measuring technique is called. This is the use of exterior tools to measure system performance.

Once a system is working, the important question is how does it work for long periods of time? The analysis of performance requires summarizing data on system behavior that is difficult to extract from the trace. To obtain this data three avenues of approach are available—all exterior techniques—and each has been successfully used for specific purposes. First, an analysis of the standard system accounting data has been made on occasion. While this data show precisely the resources used, and the elapsed time, it is next to impossible to infer what else is going on within the system. And, of course, those system functions that are invisible to the user, like memory compaction, are not reported on the accounting data.

The second technique we used was to record the trace entries on a magnetic tape for later analysis. A program was written to extract any desired subset of the trace entries from the tape and to print them, along with time of day, differences in time of day between successive entries, and the time of day differences between successive like entries.

An analysis of this kind of data allows a measurement of swapping time and swapping frequency, for instance. In general, timing studies of any specific system function can be made with this kind of data. Unfortunately, it is not possible to easily measure the degree of system

utilization using this technique. For instance, it is not possible using this trace data to determine the length of the dispatcher queue. Figure 3 shows an example of this data.

This figure shows event trace entries from a GECOS III run that were saved on a magnetic tape and then summarized. The summarizing program adds the left-most and the last three columns on the right. The first of these three columns contains the change in time of day (TRACE DELTA) between each successive pair of trace entries which contain time of day as part of their data. For instance, the first column shows that 8.70 milliseconds elapsed between the first and second lines on the trace. The second column shows the time difference between successive like entries.

Here one can see that the time between the three dispatches was 25.36, 28.59 and 33.83 milliseconds. Finally,

GECOS 0 VERSION 0 04/11/68 CPO FPO SYSTEM FAULT IC 042612 IM 53527256534 LM 104 SYSTEM ID 10TEST

IC 042612 IR 002200 BA 042336 ER 000 AR 007121003410 QR 010142007161 TR 00007745
 XO 004300 X1 042160 X2 000000 X3 041212 X4 777154 X5 042000 X6 000001 X7 000000

TRACE TABLE- LAST ENTRY 037520

037474	LV	STIO-IC+I	014243	0000	WORD2	000000001000	IDLE CHAN-	NDEX	000000	TIME OF DAY	003430665374
037500	RTN	- INT-WORD1	000000000017		WORD2	003430665437	DISPATCH-	IC+I	005465 XXX2	PRC 0 PRG 01	TOD 003430665473
037504	MME	FAULT-IC+I	042706	0002	PRC 0 PRG 01	TYPE GESPEC	.CALL-	IC+I	042725	0002	PRC 0 PRG 01 .MIOS ENTRY 04
037510	.CALL-	IC+C	042737	4002	PRC 0 PRG 01	.MIOS ENTRY 07	MME	FAULT-IC+I	043063	0002	PRC 0 PRG 01 TYPE GEENDC
037514	LV	GEPR,ABT,SWAP,CC-			PRC 0 PRG 01	TIME 003430666212	DISPATCH-	IC+I	042635	XXX2	PRC 0 PRG 01 TOD 003430666252
037520	.CALL-	IC+I	042612	0022	PRC 0 PRG 01	.MFALT ENTRY 01	IOC	CTRS S-T-I	120600	0000	000000000000
037524	PROC	TI- IO	ENTRY 041212	Q CNT 06	PRG 01	QUE 400000000020	CONNECT-	IO	ENTRY 041212	20000	PMX 250120000000
037530	CHAN	QUE.- NDEX	0020	SIZE 00001	TIME OF DAY	003423767077	LV	STIO-IC+I	014243	0000	WORD2 041212001000
037534	RTN	- INT-WORD1	000000000017		WORD2	003423767165	TERM	INT= IC+I	004174	0062	TIME OF DAY 003424020164
037540	IOC	CTRS S-T-I	120700	0000		000000000000	PROC	TI- IO	ENTRY 041212	Q CNT 07	PRG 01 QUE 440000000020
037544	LV	STRET-IC+CX	014207	0020	PRC 0 PRG 01	ENTRY 041212	LV	STIO-IC+I	014243	0000	WORD2 000000001000
037550	IDLE	CHAN- NDEX	000020		TIME OF DAY	003424020351	RTN	= INT-WORD1	000000000017		WORD2 003424020376
037554	DISPATCH-	IC+I	005465	XXX2	PRC 0 PRG 01	TOD 003424020432	.CALL-	IC+I	034357	0002	PRC 0 PRG 01 .MIOS ENTRY 01
037560	CONNECT-	IO	ENTRY 041212	20000		PMX 340120000000	CHAN	QUE.- NDEX	0020	SIZE 00001	TIME OF DAY 003424020567
037564	LV	STIO-IC+I	014752	0000		WORD2 041212001000	TERM	INT= IC+I	005466	2002	TIME OF DAY 003424020660
037570	IOC	CTRS S-T-I	121000	0000		000000000000	PROC	TI- IO	ENTRY 041212	Q CNT 10	PRG 01 QUE 400000000020
037574	CONNECT-	IO	ENTRY 041212	20000		PMX 250120000000	CHAN	QUE.- NDEX	0020	SIZE 00001	TIME OF DAY 003424021007
037600	LV	STIO-IC+I	014243	0000		WORD2 041212001000	RTN	= INT-WORD1	012762000217		WORD2 003424021075
037604	DISPATCH-	IC+I	005466	XXX2	PRC 0 PRG 01	TOD 003424021132	IDLE	PROCESSOR-		PRC 0	TIME OF DAY 003424021171
037610	TERM	INT= IC+I	004174	0062		TIME OF DAY 003424037135	IOC	CTRS S-T-I	121100	0000	000000000000
037614	PROC	TI- IO	ENTRY 041212	Q CNT 11	PRG 01	QUE 400000000020	LV	STRET-IC+CX	014207	0020	PRC 0 PRG 01 ENTRY 041212
037620	LV	STIO-IC+I	014243	0000		WORD2 000000001000	IDLE	CHAN- NDEX	000020		TIME OF DAY 003424037322
037624	RTN	- INT-WORD1	000000000017		WORD2	003424037347	DISPATCH-	IC+I	005465	XXX2	PRC 0 PRG 01 TOD 003424037403
037630	.CALL-	IC+I	040210	0022	PRC 0 PRG 01	.MIOS ENTRY 04	.CALL-	IC+I	040274	2022	PRC 0 PRG 01 .MIOS ENTRY 01

Figure 2—Tracetable

ments; however, this is sufficient to allow measurements of dispatch, or interrupt processing time. As the system grows, measurement of swapping and so forth can be made. Finally, comes the day that the parts work individually and interesting questions revolve around the relationships between the component parts.

At this time, the third exterior measuring tool is needed. This is what we call a system monitor. The monitor is a user program that is allowed to break into the system itself. It collects and summarizes a great number of the parameters available in the system. These are displayed at several-second intervals on a printer or cathode-ray tube. Both devices have their place. The CRT is used for continuous display during normal use of the system. The printer is needed when specific analysis is to be made of particular jobs.

An example of the printer output from this monitor is found in Figure 4. Figures 5 through 12 illustrate the

data displayed on the cathode-ray tube by this monitor program. When the program is called by the CRT terminal, the display shown on Figure 5 is presented. With this display we can pick a sampling interval and also pick one of the given specific monitor displays. Once the time interval and display have been chosen, the monitor program passively samples the system at the rate chosen and displays the data. The user at the CRT terminal may break in with a request for a new display at any time. Figures 6 through 12 are samples of the displays numbered one through seven in Figure 5.

The first display, system configuration, is shown in Figure 6. This shows the devices found on each of the 16 independent pubs or input-output channels. It also shows in the second column the number of devices actually available. It will be noted from Figure 6 that one tape unit on pub one is unavailable. Figure 7 is a display of a great deal of data of interest to the system designer.

18219 01 07-18-67		GECOS III TRACE		(TIME IN MS)	PAGE	19	TRACE DELTA	EVENT DELTA	TOD
035465	IDLE PROCESSOR	PRC 0					8.70	50.80	39695.31
035466	TERMINATE INTERRUPT		IC&I 003276 0062				6.94	19.05	39702.25
035472	RETURN FROM INT. PROCESSOR	PRG 00 SCT	000000 CH&IOC 00 STAT1 00000000XX00				1.41	17.50	39703.66
035473	TERMINATE INTERRUPT		IC&I 003276 0062				4.92	6.33	39708.58
035477	INITIATION INTERRUPT		IC&I 007102 0042				1.73	25.38	39710.31
035502	RETURN FROM INT. PROCESSOR	PRG 00 SCT	000000 CH&IOC 00 STAT1 00000000XX00				1.20	7.86	39711.52
035503	DISPATCH TO PROGRAM	PRC 0 PRG 77	IC&I 004451 0772				.45	25.36	39711.97
035506	IDLE PROCESSOR	PRC 0					1.69	18.34	39713.66
035507	TERMINATE INTERRUPT		IC&I 003276 0062				23.50	28.58	39737.16
035513	INITIATION INTERRUPT		IC&I 007102 0042				1.72	28.56	39738.88
035516	RETURN FROM INT. PROCESSOR	PRG 00 SCT	000000 CH&IOC 00 STAT1 00000000XX00				1.23	28.59	39740.11
035517	DISPATCH TO PROGRAM	PRC 0 PRG 20	IC&I 004451 0202				.45	28.59	39740.56
035524	IDLE PROCESSOR	PRC 0					3.78	30.69	39744.34
035525	TERMINATE INTERRUPT		IC&I 003276 0062				16.97	24.16	39761.31
035531	RETURN FROM INT. PROCESSOR	PRG 00 SCT	000000 CH&IOC 00 STAT1 00000000XX00				1.38	22.58	39762.69
035532	TERMINATE INTERRUPT		IC&I 003276 0062				8.31	9.69	39771.00
035536	INITIATION INTERRUPT		IC&I 007102 0042				1.73	33.86	39772.73
035541	RETURN FROM INT. PROCESSOR	PRG 00 SCT	000000 CH&IOC 00 STAT1 00000000XX00				1.20	11.25	39773.94
035542	DISPATCH TO PROGRAM	PRC 0 PRG 77	IC&I 004451 0772				.45	33.83	39774.39
035545	IDLE PROCESSOR	PRC 0					3.22	33.27	39777.61
035546	TERMINATE INTERRUPT		IC&I 003276 0062				48.16	54.77	39825.77
035552	RETURN FROM INT. PROCESSOR	PRG 00 SCT	000000 CH&IOC 00 STAT1 00000000XX00				1.36	53.19	39827.13
035553	TERMINATE INTERRUPT		IC&I 003276 0062				12.53	13.89	39839.66

Figure 3—Trace summary

TSIXY 01 08-25-67

Figure 4—Printer monitor

PAGE 3

MEMORY MAP(%=PROC, TIME/DELTA TIME)	%PU-MU-DR-DS	PRG(#)	CORE	SWAP	W/CR	W/PRF)	QUEUE(DISP)	CC	SYSO	P-AL	C-AL)	TOD
000-031 0 1%191%GGGGGGGGGGGS<S00%3333S0	9 67 49 57	7	7	0	0	0	0	0	0	0	0	0.020
032-063 0%2222S00%7												
064-095 SS00%UUUUUUUUUUUU												
096-127 SS04%UUUUUUUUUUUS01%UUU												
000-031 0 1%189%GGGGGGGGGGGS<S00%3333S0	11 67 49 57	7	7	0	0	0	0	0	0	0	0	0.021
032-063 0%2222S00%7												
064-095 SS00%UUUUUUUUUUUU												
096-127 SS05%UUUUUUUUUUUS01%UUU												
000-031 0 1%191%GGGGGGGGGGGS0S00%3333S0	9 67 49 57	7	7	0	0	0	1	0	0	0	0	0.022
032-063 0%2222S00%7												
064-095 SS00%UUUUUUUUUUUU												
096-127 SS05%UUUUUUUUUUUS01%UUU												
000-031 0 2%187%GGGGGGGGGGGS<S<1%3333S<	13 53 47 57	6	6	0	0	0	0	0	0	0	0	0.022
032-063 1%2222S00%7												
064-095												
096-127 SS04%UUUUUUUUUUUS01%UUU												
000-031 0 1%190%GGGGGGGGGGGS0S00%3333S0	10 53 47 57	6	6	0	0	0	0	0	0	0	0	0.023
032-063 0%2222S00%7												
064-095 SS06%UUUUUUUUUUUS01%UUU												
096-127												
000-031 0 1%190%GGGGGGGGGGGS<S<1%3333S0	10 53 47 57	6	6	0	0	0	1	0	0	0	0	0.024
032-063 0%2222S00%7												
064-095 SS05%UUUUUUUUUUUS01%UUU												
096-127												
000-031 0 1%191%GGGGGGGGGGGS<S00%3333S0	9 53 47 57	6	6	0	0	0	0	0	0	0	0	0.024
032-063 0%2222S00%7												
064-095 SS04%UUUUUUUUUUUS01%UUU												
096-127												
000-031 0 2%190%GGGGGGGGGGGS<S<1%3333S0	10 53 47 57	6	6	0	0	0	0	0	0	0	0	0.025
032-063 0%2222S00%7												
064-095 SS05%UUUUUUUUUUUS01%UUU												
096-127												
000-031 0 1%189%GGGGGGGGGGGS0S00%3333S0	11 53 47 57	6	6	0	0	0	0	0	0	0	0	0.026
032-063 0%2222S00%7												
064-095 SS06%UUUUUUUUUUUS01%UUU												
096-127												
000-031 0 1%191%GGGGGGGGGGGS<S<1%3333S0	9 53 47 57	6	6	0	0	0	0	0	0	0	0	0.026
032-063 0%2222S00%7												
064-095 SS05%UUUUUUUUUUUS01%UUU												
096-127												

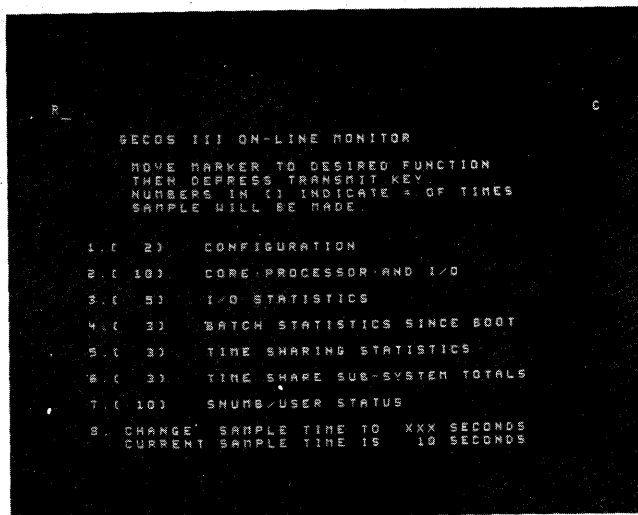


Figure 5—Monitor options

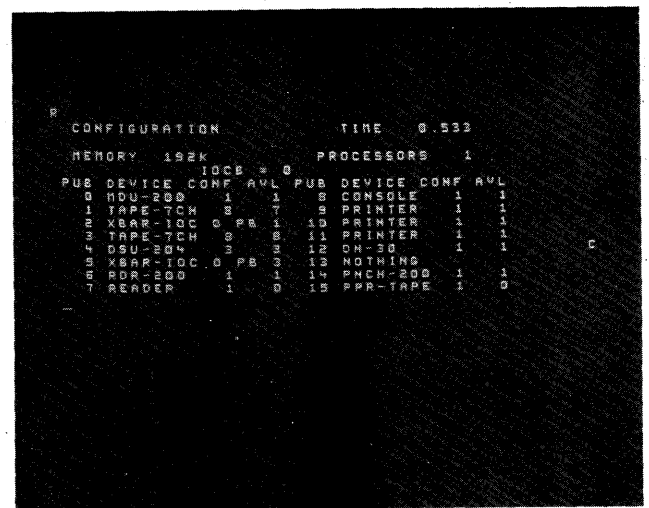


Figure 6—Configuration

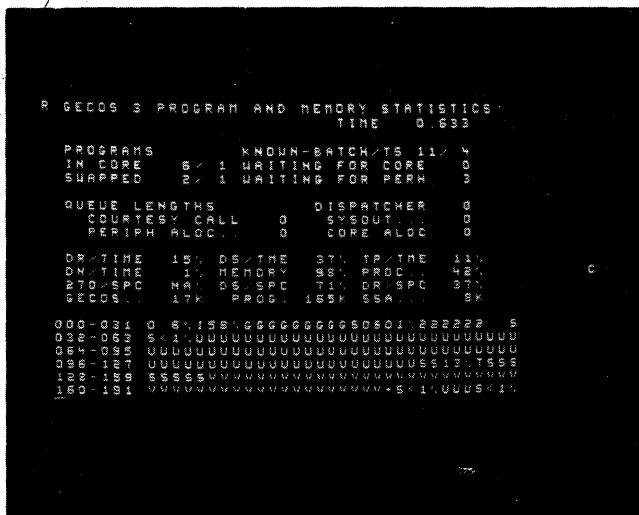


Figure 7—Program and memory statistics

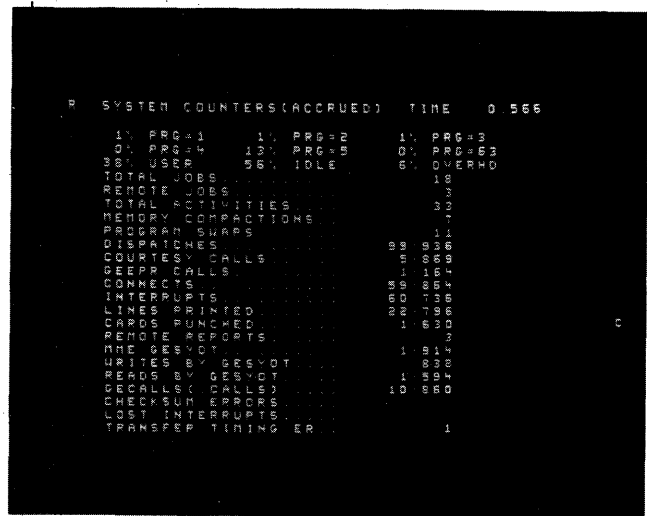


Figure 9—Accured statistics

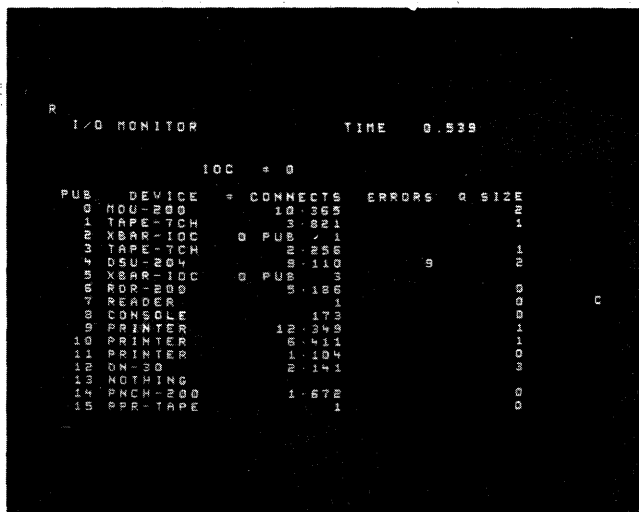


Figure 8—I/O statistics

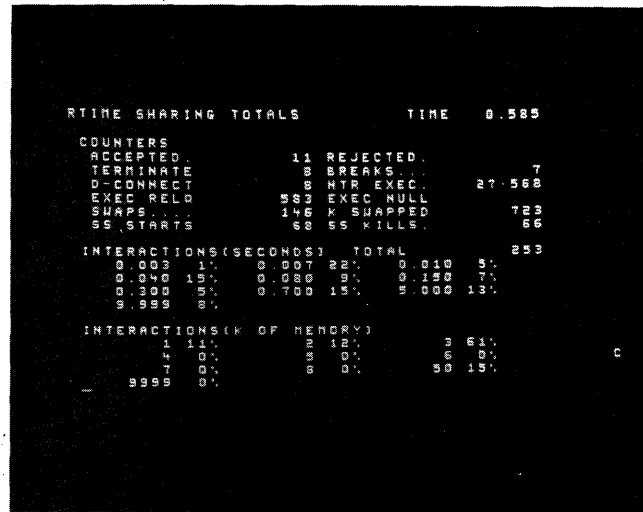


Figure 10—Time sharing statistics

The top section shows the status of all programs known to the batch system and to time-sharing. There are, for instance, eleven batch programs and four time-sharing users on the system. The second section shows the queue lengths of unprocessed demands made on each of the major system components. In general, these queues have length zero where no part of the system is saturated. The dispatcher queue length is of particular interest because its length is a measure of multiprogramming interference.

The third section of the display shows channel busy time by device type as well as memory and processor use. The percent of available disc and drum space currently in use is also shown. Finally there is a summary of core usage in this section.

The fourth section of the display is a diagram of core utilization. On the left are shown memory address ranges while each symbol on the right stands for 1024 words of core. The following meanings are assigned to the symbols:

- O overhead percentage for the system
- I idle percentage for the system
- G occupied by the resident executive (a hard core monitor)
- S user program slave service area
- XX% percent of processor time used by that user in the sample interval
- U user program
- TSS—S time-sharing executive
- VVV—V core available for time-sharing users
- + or * core space in use by time-sharing user.
- blank available (unused) core

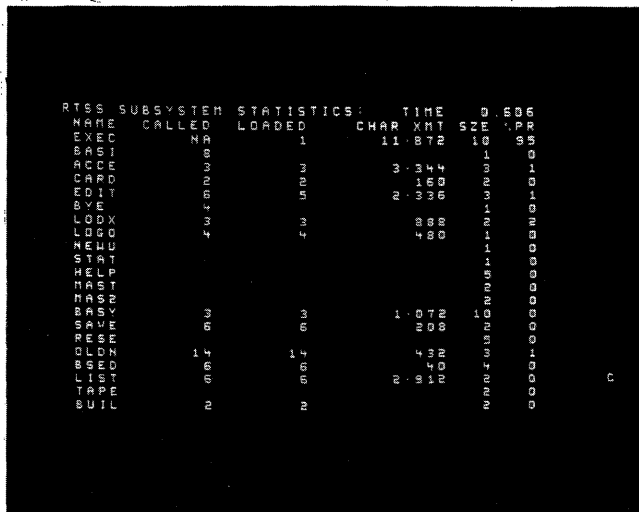


Figure 11—Time sharing subsystem usage

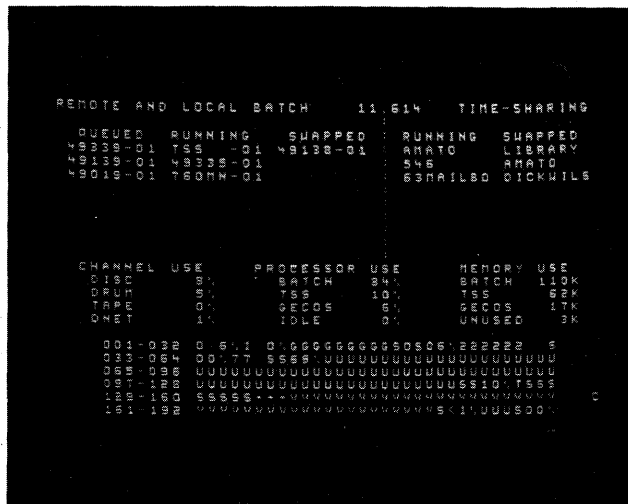


Figure 12—User status

- 22--22 peripheral allocator program
- 33--33 system output printer

From this display we can see that only two blocks of core are unused within the batch world. In the time-sharing system, only one user is in core. Since the plus sign is at the high end of the time-sharing core, this user is an old interaction. The monitor program itself happens to be the second to last program (indicated by $S < 1\%$ UUU) in the last line of the display. Note that since this program is small and uses very little processor time, it does not noticeably bias the measures it is taking.

The fourth display (Figure 9) shows summaries of total processor utilization by some system functions and

also the number of occurrences of certain system events. In this display the following program numbers are related to the following functions.

Program	Function
1	core allocation
2	peripheral allocation
3	system output disperser
4	remote input collector
5	time-sharing systems
63	on-line input collector

When the accumulated processor time is less than one percent, the display shows zero. This explains why there is no time shown for program four while below, we can see three remote jobs have passed through the system.

Figures 10 and 11 summarize various data from the time-sharing system. In GECOS III, the whole time-sharing system is treated as a single batch job. The time-sharing executive makes its own internal scheduling decisions. The first display summarizes the data generated by the time-sharing executive. The time and space profiles of interaction will be of particular interest to the system designer. The next display (Figure 11) shows the usage of the various time-sharing subsystems. Notably absent here is FORTRAN which was not in the system when these photographs were made. It has since been added to the system.

The last display is the one we most often use. It is illustrated in Figure 12. This display identifies all jobs and time-sharing users known to the system. It will be noted that there are seven batch users including TSS, the time-sharing system, and 760 MN, the monitor. One has been in execution and is now swapped out of core. There are six time-sharing users. Unfortunately the display of time-sharing users in core does not exactly match the core swap. This is because the passive monitor does not get all its data at the same time, so between the top display and the memory map below, there has been movement of users within the time-sharing system. The dots on the screen below time of day indicate the difference between batch and time-sharing users. The middle display shows channel, processor and memory use summaries. The bottom display is a memory map like Figure 7.

By studying this system monitor, we are able to continuously verify that the system is behaving properly. We have this display set up in the development manager's office with a second screen in the computer room. When we observe anomalous behavior we are able to get a system dump immediately so that we can trace the problem. We have found this monitor to be our most powerful tool in tuning our system for maximum performance.

Another monitor has been produced to find the degree of multiprogramming interference in I/O. This interference is the delay between the time a particular I/O request is issued by a program until it actually gets started. This program analyzes all I/O demands in terms of the particular logical file, frequency of demand and amount of interference. This tool is helpful in deciding how best to assign particular files for best I/O overlap.

CONCLUSION

We have described a large number of measurement techniques we have employed in developing our operating systems. The number and variety of means demonstrates the many different problems faced by the system

developer. If we have learned any single lesson from our efforts in this area, it is that continuous measurement of a system is an absolute necessity if the system is to be kept working at top efficiency. It is truly amazing how seemingly minor changes in a system can have profound effects on overall performance.

REFERENCES

- 1 D J CAMPBELL W F COOK W J HEFFNER
Software Age January 1968 p 8
- 2 D J CAMPBELL W F COOK W J HEFFNER
Datamation November 1967 page 77
- 3 H N CANTRELL A L ELLISON
SJCC 1968 *Performance measurement*

Thrashing: Its causes and prevention

by PETER J. DENNING

Princeton University*
Princeton, New Jersey

INTRODUCTION

A particularly troublesome phenomenon, thrashing, may seriously interfere with the performance of paged memory systems, reducing computing giants (Multics, IBM System 360, and others not necessarily excepted) to computing dwarfs. The term *thrashing* denotes excessive overhead and severe performance degradation or collapse caused by too much paging. Thrashing inevitably turns a shortage of memory space into a surplus of processor time.

Performance of paged memory systems has not always met expectations. Consequently there are some who would have us dispense entirely with paging,¹ believing that programs do not generally display behavior favorable to operation in paged memories. We shall show that troubles with paged memory systems arise not from any misconception about program behavior, but rather from a lack of understanding of a three-way relationship among program behavior, paging algorithms, and the system hardware configuration (i.e., relative processor and memory capacities). We shall show that the prime cause of paging's poor performance is not unfavorable program behavior, but rather the large time required to access a page stored in auxiliary memory, together with a sometimes stubborn determination on the part of system designers to simulate large virtual memories by paging small real memories.

After defining the computer system which serves as our context, we shall review the working set model for program behavior, this model being a useful vehicle for understanding the causes of thrashing. Then we shall show that the large values of secondary memory access times make a program's steady state processing efficiency so sensitive to the paging requirements of

other programs that the slightest attempt to overuse main memory can cause service efficiency to collapse. The solution is two-fold: first, to use a memory allocation strategy that insulates one program's memory-space acquisitions from those of others; and second, to employ memory system organizations using a non-rotating device (such as slow-speed bulk core storage) between the high-speed main memory and the slow-speed rotating auxiliary memory.

Preliminaries

Figure 1 shows the basic two-level memory system in which we are interested. A set of identical processors has access to M pages of directly-addressable, multi-programmed *main memory*; information not in main memory resides in auxiliary memory which has, for our purposes, infinite capacity. There is a time T , the *traverse time*, involved in moving a page between the levels of memory; T is measured from the moment a missing page is referenced until the moment the required page transfer is completed, and is therefore the expectation of a random variable composed of waits in queues, mechanical positioning delays, page transmission times, and so on. For simplicity, we assume T is the same irrespective of the direction a page is moved.

Normally the main memory is a core memory, though it could just as well be any other type of directly-addressable storage device. The auxiliary memory is usually a disk or drum but it could also be a combination of slow-speed core storage and disk or drum.

We assume that information is moved into main memory only on *demand* (demand paging); that is, no attempt is made to move a page into main memory until some program references it. Information is returned from main to auxiliary memory at the discretion of the *paging algorithm*. The information movement across the channel bridging the two levels of memory is called *page traffic*.

A *process* is a sequence of references (either fetches or

*Department of Electrical Engineering. The work reported herein, completed while the author was at Project MAC, was supported in part by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract No. Nonr-4102(01).

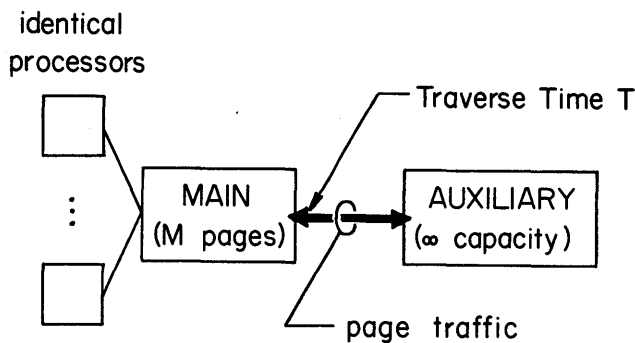


FIGURE 1—Basic two-level memory system

stores) to a set of information called a *program*. We assume that each program has exactly one process associated with it. In this paper we are interested only in *active* processes. An active process may be in one of two states: the *running* state, in which it is executing on a processor; or the *page wait* state, in which it is temporarily suspended awaiting the arrival of a page from auxiliary memory. We take the duration of the page wait state to be T , the traverse time.

When talking about processes in execution, we need to distinguish between real time and virtual time. *Virtual time* is time seen by an active process, as if there were no page wait interruptions. By definition, a process generates one information reference per unit virtual time. *Real time* is a succession of virtual time intervals (i.e., computing intervals) and page wait intervals. A *virtual time unit* (vtu) is the time between two successive information references in a process, and is usually the memory cycle time of the computer system in which the process operates.

In this paper we take $1 \text{ vtu} = 1 \text{ microsecond}$, since 1 microsecond is typical of core memory cycle times. The table below lists estimates of the traverse time T for typical devices, using the approximate relation

$$T = T_a + T_t$$

where T_a is the mechanical access time of the device and T_t is the transmission time for a page of 1000 words.

Storage Device	T_a	T_t (page = 1000 words)	$T = T_a + T_t$
thin film core	0	10^2 vtu	10^2 vtu
bulk core	0	10^3 vtu	10^3 vtu
high speed drum	10^4 vtu	10^4 vtu	10^4 vtu
moving-arm disk	10^6 vtu	10^3 vtu	10^6 vtu

The working set model for program behavior

In order to understand the causes and cures for

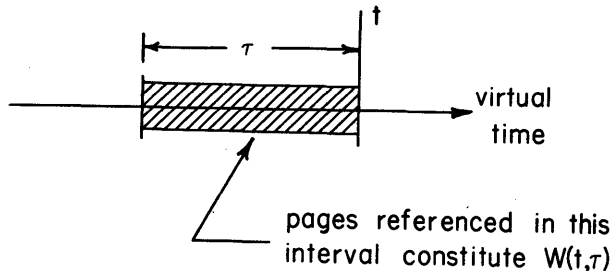


FIGURE 2—Definition of working set

thrashing, it is necessary to understand some basic properties of program behavior. The working set model for program behavior, discussed in detail in reference², is a useful way for understanding these properties, so we review it here.

By a *program* we mean the set of pages to which a process directs its references. A basic program property, that of *locality*, is the non-uniform scattering of a process's reference across its program during any virtual time interval. That is, a process tends to favor some of its pages more than others. During disjoint virtual time intervals, the set of favored pages may be different. Locality has been observed to various degrees in existing programs,^{3,4} and it can be considerably enhanced if programmers design their algorithms to operate locally on information, one region at a time.

The *working set* of information $W(t, \tau)$ associated with a process at time t is the set of pages referenced by the process during the virtual time interval $(t - \tau, t)$. The concept is illustrated in Figure 2.

The *working set size* $\omega(t, \tau)$ is the number of pages in $W(t, \tau)$. Observe that $\omega(t, \tau) \leq \tau$, since no more than τ distinct pages can be referenced in an interval of length τ ; that $\omega(t, 0) = 0$, since no references can occur in zero time; and that $\omega(t, \tau)$ is a non-decreasing function of τ , since more references can occur in longer intervals $(t - \tau, t)$.

The working set model owes its validity to locality. A working set measures the set of pages a process is favoring at time t ; assuming that processes are not too fickle, that is, they do not abruptly change sets of favored pages, the working set $W(t, \tau)$ constitutes a reliable estimate of a process's immediate memory need.

Intuitively, a working set is the smallest set of pages that ought to reside in main memory so that a process can operate efficiently. Accordingly, τ should be chosen as small as possible and yet allow $W(t, \tau)$ to contain at least the favored pages. In principle, then, τ may vary from program to program and from time to time. A *working set memory allocation policy* is one that permits a process to be active if and only if there is enough uncommitted space in main memory to contain its working set.

Define the random variable x_s to be the virtual time interval between successive references to the same page in a program comprising s pages; these *interference intervals* x_s are useful for describing certain program properties. Let $F_{x_s}(u) = \Pr[x_s \leq u]$ denote its distribution function (measured over all programs of size s), and let \bar{x}_s denote its mean.

The relation between the size of a program and the lengths of the interference intervals to its component pages may be described as follows. Let process 1 be associated with program P_1 (of sizes s_1) and process 2 be associated with program P_2 (of size s_2), and let P_1 be larger than P_2 . Then process 1 has to scatter its references across a wider range of pages than process 2, and we expect the interference intervals x_{s_1} of process 1 to be no longer than the interference intervals x_{s_2} of process 2. That is, $s_1 > s_2$ implies $\bar{x}_{s_1} > \bar{x}_{s_2}$.

Memory management strategies

It is important to understand how programs can interfere with one another by competing for the same limited main memory resources, under a given paging policy.

A good measure of performance for a paging policy is the *missing-page probability*, which is the probability that, when a process references its program, it directs its reference to a page not in main memory. The better the paging policy, the less often it removes a useful page, and the lower is the missing-page probability. We shall use this idea to examine three important paging policies (ordered here according to increasing cost of implementation):

1. **First In, First Out (FIFO)**: whenever a fresh page of main memory is needed, the page least recently paged in is removed.
2. **Least Recently Used (LRU)**: whenever a fresh page of main memory is needed, the page unreferences for the longest time is removed.
3. **Working Set (WS)**: whenever a fresh page of main memory is needed, choose for removal some page of a non-active process or some non-working-set page of an active process.

Two important properties set WS apart from the other algorithms. First is the explicit relation between memory management and process scheduling: a process shall be active if and only if its working set is fully contained in main memory. The second is that WS is applied individually to each program in a multiprogrammed memory, whereas the others are applied globally across the memory. We claim that applying a paging algorithm globally to a collection of programs may lead to undesirable interactions among them.

How do programs interact with each other, if at all,

under each of these strategies? How may the memory demands of one program interfere with the memory allocated to another? To answer this, we examine the missing-page probability for each strategy.

In a multiprogrammed memory, we expect the missing-page probability for a given program to depend on its own size s , on the number n of programs simultaneously resident in main memory, and on the main memory size M :

$$(1) \quad (\text{missing-page probability}) = m(n, s, M)$$

Suppose there are n programs in main memory; intuitively we expect that, if the totality of their working sets does not exceed the main memory size M , then no program loses its favored pages to the expansion of another (although it may lose its favored pages because of foolish decisions by the paging algorithm). That is, as long as

$$(2) \quad \sum_{i=1}^n \omega_i(t, \tau_i) \leq M$$

there will be no significant interaction among programs and the missing-page probability is small. But when n exceeds some critical number n_0 , the totality of working sets exceeds M , the expansion of one program displaces working set pages of another, and so the missing-page probability increases sharply with n . Thus,

$$(3) \quad m(n_1, s, M) > m(n_2, s, M) \quad \text{if } n_1 > n_2$$

This is illustrated in Figure 3.

If the paging algorithm operates in the range $n > n_0$, we will say it is *saturated*.

Now we want to show that the FIFO and LRU algorithms have the property that

$$(4) \quad m(n, s_1, M) \geq m(n, s_2, M) \quad \text{if } s_1 > s_2$$

That is, a large program is at least as likely to lose pages than a small program, especially when the paging algorithm is saturated.

To see that this is true under LRU, recall that if program P_1 is larger than P_2 , then the interference intervals satisfy $\bar{x}_1 > \bar{x}_2$: large programs tend to be the ones that reference the least recently used pages. To see that this is true under FIFO, note that a large program is likely to execute longer than a small program, and thus it is more likely to be still in execution when the FIFO algorithm gets around to removing its pages. The interaction among programs, expressed by Eq. 4, arises from the paging algorithm's being applied globally across a collection of programs.

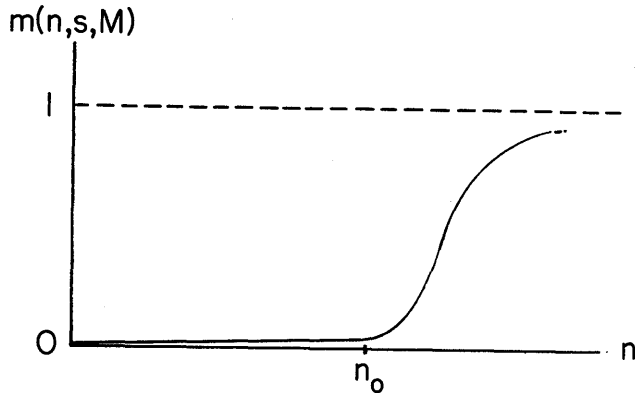


FIGURE 3—Missing-page probability

Finally, we note that under a WS algorithm, the missing-page probability is independent of n and M since eq. 2 is always satisfied. The missing-page probability depends only on the choice of τ ; indeed,

$$\begin{aligned}
 m_s(\tau) &= \text{Pr}[\text{missing page is referenced}] \\
 &\quad \text{in size } s \text{ program} \\
 (5) \quad &= \text{Pr}[\text{page referenced satisfies } x_s > \tau] \\
 m_s(\tau) &= 1 - F_{x_s}(\tau)
 \end{aligned}$$

where $F_{x_s}(u) = \text{Pr}[x_s \leq u]$ has already been defined to be the interference distribution. Therefore, the WS algorithm makes programs independent of each other. We shall show shortly that this can prevent thrashing.

From now on, we write m instead of $m(n,s,M)$.

Steady state efficiency and thrashing

Suppose that a certain process has executed for a virtual time interval of length V and that the missing-page probability m is constant over this interval V . The expected number of page waits is then (Vm) , each costing one traverse time T . We define the *efficiency* $e(m)$ to be:

$$(6) \quad e(m) = \frac{(\text{elapsed virtual time})}{(\text{elapsed virtual time}) + (\text{elapsed page wait time})}$$

Then,

$$(7) \quad e(m) = \frac{V}{V + VmT} = \frac{1}{1 + mT}$$

Clearly, $e(m)$ measures the ability of an active process to use a processor.

Figure 4 shows $e(m)$ for five values of T :

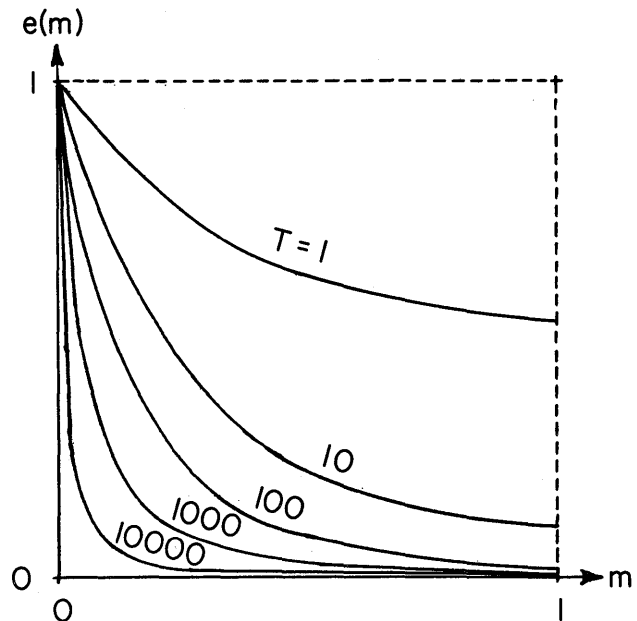


FIGURE 4—Efficiency

$T = 1, 10, 100, 1000, 10000$ vtu

where $T = 10000$ vtu may be regarded as being typical of the fastest existing rotating auxiliary storage devices.

The slope of $e(m)$ is

$$(8) \quad e'(m) = \frac{d}{dm} e(m) = \frac{-T}{(1 + mT)^2}$$

which means that, for small m and $T \gg 1$, $e(m)$ is extremely sensitive to a change in m . It is this extreme sensitivity of $e(m)$ to m -fluctuations for large T that is responsible for thrashing.

To show how the slightest attempt to overuse memory can wreck processing efficiency, we perform the following conceptual experiment. We imagine a set of $(n + 1)$ identical programs, n of which are initially operating together, without sharing, in memory at the verge of saturation (that is, $n = n_0$ in Figure 3); then we examine the effect of introducing the $(n + 1)$ -st program.

Let $1, 2, \dots, (n + 1)$ represent this set of $(n + 1)$ identical programs, each of average size s . Initially, n of them fully occupy the memory, so that the main memory size is $M = ns$. Let m_0 denote the missing-page probability under these circumstances; since there is (on the average) sufficient space in main memory to contain each program's working set, we may assume $m_0 \ll 1$ and that $e(m_0)$ is reasonable (i.e., it is not true that $e(m_0) \ll 1$). Then, the expected number of busy processors (ignoring the cost of switching a processor) is:

$$(9) \quad p = \sum_{i=1}^n e_i(m_o) = \frac{n}{1 + m_o T}$$

Now introduce the $(n + 1)$ -st program. The missing-page probability increases to $(m_o + \Delta)$ and the expected number of busy processors becomes

$$(10) \quad p' = \sum_{i=1}^{n+1} e_i(m_o + \Delta) = \frac{n + 1}{1 + (m_o + \Delta)T}$$

Now if the pages of n programs fully occupy the memory and we squeeze another program of average size s into memory, the resulting increase in the missing-page probability is

$$(11) \quad \Delta = \frac{s}{(n + 1)s} = \frac{1}{n + 1}$$

since we assume that the paging algorithm obtains the additional s pages by displacing s pages uniformly from the $(n + 1)$ identical programs now resident in main memory. The fractional number of busy processors after introduction of the $(n + 1)$ -st program is

$$(12) \quad \frac{p'}{p} = \frac{n + 1}{n} \frac{1 + m_o T}{1 + (m_o + \Delta)T}$$

We assume that the traverse time T is very large; that is, T (in vtu) $\gg n \gg 1$. We argue that

$$\Delta = \frac{1}{n + 1} \gg m_o$$

To show this, we must show that neither $\Delta \approx m_o$ nor $\Delta \ll m_o$ is the case. First, $\Delta \approx m_o$ cannot be the case, for if it were, we would have (recalling $T \gg n \gg 1$):

$$(13) \quad e(m_o) \approx e(\Delta) = \frac{1}{1 + \Delta T} \\ = \frac{1}{1 + \frac{T}{n + 1}} \\ = \frac{n + 1}{n + 1 + T} \ll 1$$

which contradicts the original assumption that, when n programs initially occupied the memory, it is not true that $e(m_o) \ll 1$. Second, $\Delta \ll m_o$ cannot be the case; for if it were, then we would have (from Eqs. 7 and 13)

$$1 \gg e(\Delta) = e\left(\frac{1}{n + 1}\right) \gg e(m_o)$$

once again contradicting the original assumption that, when n programs initially occupied the memory, it is not true that $e(m_o) \ll 1$. Thus, we conclude that $\Delta \gg m_o$.

When $T \gg n \gg 1$ and $\Delta = \frac{1}{n + 1} \gg m_o$, it is easy to show that

$$(14) \quad \frac{p'}{p} \approx \frac{n + 1}{T} + (n + 1)m_o \ll 1$$

The presence of one additional program has caused a complete collapse of service.

The sharp difference between the two cases at first defies intuition, which might lead us to expect a gradual degradation of service as new programs are introduced into crowded main memory. The excessive value of the traverse time T is the root cause; indeed, the preceding analysis breaks down when it is not true $T \gg n$.

The recognition that large traverse times may interfere with system performance is not new. Smith,⁵ for example, warns of this behavior.

Relations among processor, memory, traverse time

We said earlier that a shortage of memory space leads to a surplus of processor time. In order to verify this statement, we shall answer the question: "Given p , what is the smallest amount of main memory needed to contain enough programs to busy an average of p processors?" We define $Q(p)$ to be this quantity of memory, and then show that p may be increased if and only if $Q(p)$ is increased, all other things being equal.

Suppose there are n identical programs in main memory, each of average size s and efficiency $e_i(m_i) = e(m)$. The expected number of busy processors is to be

$$(15) \quad p = \sum_{i=1}^n e_i(m_i) = n e(m)$$

so that

$$(16) \quad n = \frac{p}{e(m)} = p(1 + mT)$$

Then the expected memory requirement is

$$(17) \quad Q(p) = ns = ps(1 + mT)$$

This relationship between memory requirement and traverse time is important. If for some reason the pag-

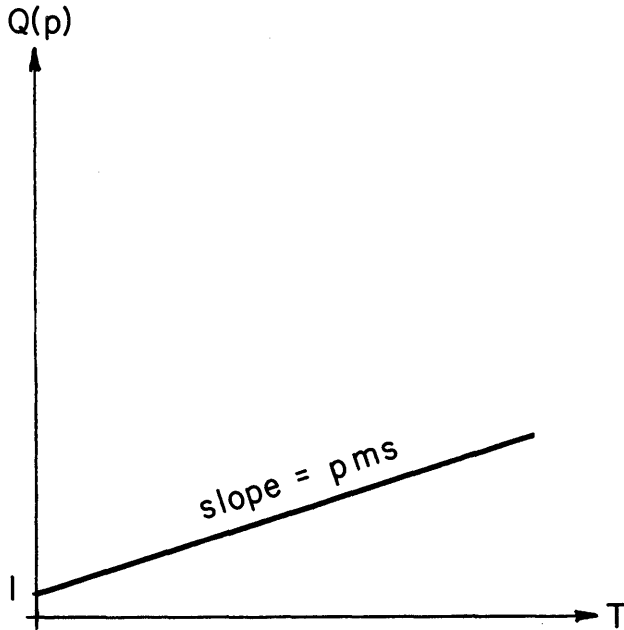


FIGURE 5—Relation between memory size and traverse time

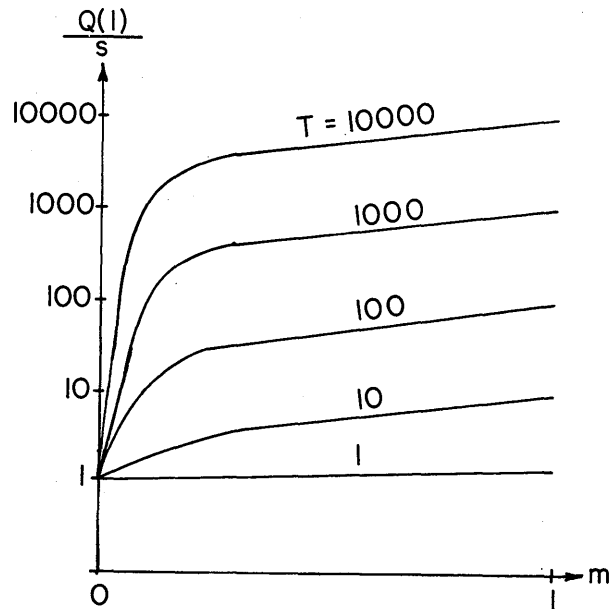


FIGURE 6—Single-processor memory requirement

ing algorithm does not make m sufficiently small, then $mT \gg 1$ because $T \gg 1$. In this case we have $Q(p) \approx psmT$, almost directly proportional to the traverse time T (see Figure 5).

Reducing T by a factor of 10 could reduce the memory requirement by as much as 10, the number of busy processors being held constant. Or, reducing T by a factor of 10 could increase by 10 the number of busy processors, the amount of memory being held constant.

This is the case. Fikes *et al.*⁶ report that, on the IBM 360/67 computer at Carnegie-Mellon University, they were able to obtain traverse times in the order of 1 millisecond by using bulk core storage, as compared to 10 milliseconds using drum storage. Indeed, the throughput of their system was increased by a factor of approximately 10.

In other words, it is possible to get the same amount of work done with much less memory if we can employ auxiliary storage devices with much less traverse time.

Figure 6, showing $Q(p)/ps$ sketched for $p = 1$ and $T = 1, 10, 100, 1000, 10000$ vtu, further dramatizes the dependence of memory requirement on the traverse time. Again, when m is small and T is large, small m -fluctuations (as might result under saturated FIFO or LRU paging policies) can produce wild fluctuations in $Q(p)$.

Normally we would choose p so that $Q(p)$ represents some fraction f of the available memory M :

$$(18) \quad (Q)p = fM \quad 0 < f \leq 1$$

so that $(1-f)M$ pages of memory are held in reserve to

allow for unanticipated working set expansions (it should be evident from the preceding discussion and from Eq. 4 that, if $Q(p) = M$, an unanticipated working set expansion can trigger thrashing). Eq. 18 represents a condition of *static balance* among the paging algorithm, the processor memory configuration, and the traverse time.

Eq. 16 and 17 show that the amount of memory $Q(p) = fM$ can increase (or decrease) if and only if p increases (or decreases), providing mT is constant. Thus, if $p' < p$ processors are available, then $Q(p') < Q(p) = fM$ and $fM - Q(p')$ memory pages stand idle (that is, they are in the working set of no active process). Similarly, if only $f'M < fM$ memory pages are available, then for some $p' < p$, $Q(p') = f'M$, and $(p-p')$ processors stand idle. A shortage in one resource type inevitably results in a surplus of another.

It must be emphasized that these arguments, being average-value arguments, are only an approximation to the actual behavior. They nevertheless reveal certain important properties of system behavior.

The cures for thrashing

It should be clear that thrashing is caused by the extreme sensitivity of the efficiency $e(m)$ to fluctuations in the missing-page probability m ; this sensitivity is directly traceable to the large value of the traverse time T . When the paging algorithm operates at or near saturation, the memory holdings of one program may interfere with those of others: hence paging strategies must be employed which make m small and indepen-

dent of other programs. The static balance relation $Q(p) = fM$ shows further that:

1. A shortage in memory resource, brought about the onset of thrashing or by the lack of equipment, results in idle processors.
2. A shortage in processor resources, brought about by excessive processor switching or by lack of equipment, results in wasted memory.

To prevent thrashing, we must do one or both of the following: first, we must prevent the missing-page probability m from fluctuating; and second, we must reduce the traverse time T .

In order to prevent m from fluctuating, we must be sure that the number n of programs residing in main memory satisfies $n \leq n_o$ (Figure 2); this is equivalent to the condition that

$$(19) \quad \sum_{i=1}^{n_o} \omega_i(t, \tau_i) \leq M$$

where $\omega_i(t, \tau_i)$ is the working set size of program i . In other words, there must be space in memory for every active process's working set. This strongly suggests that a working set strategy be used. In order to maximize n_o , we want to choose τ as small as possible and yet be sure that $W(t, \tau)$ contains a process's favored pages. If each programmer designs his algorithms to operate locally on data, each program's set of favored pages can be made surprisingly small; this in turn makes n_o larger. Such programmers will be rewarded for their extra care, because they not only attain better operating efficiency, but they also pay less for main store usage.

On the other hand, under paging algorithms (such as FIFO or LRU) which are applied globally across a multiprogrammed memory, it is very difficult to ascertain n_o , and therefore difficult to control m -fluctuations.

The problem of reducing the traverse time T is more difficult. Recall that T is the expectation of a random variable composed of queue waits, mechanical positioning times, and page transmission times. Using optimum scheduling techniques⁷ on disk and drum, together with parallel data channels, we can effectively remove the queue wait component from T ; accordingly, T can be made comparable to a disk arm seek time or to half a drum revolution time. To reduce T further would require reduction of the rotation time of the device (for example, a 40,000 rpm drum).

A much more promising solution is to dispense altogether with a rotating device as the second level of memory. A three-level memory system (Figure 7) would be a solution, where between the main level (level 0) and the drum or disk (level 2) we introduce a bulk core storage.

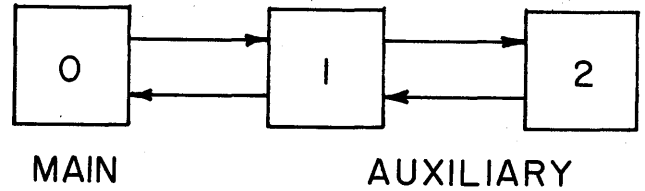


FIGURE 7—Three-level memory system

gests that it is possible, in today's systems, to reduce the traverse time T by a factor of 10 or more. There are two important reasons for this. First, since there is no mechanical access time between levels 0 and 1, the traverse time depends almost wholly on page transmission time; it is therefore economical to use small page sizes. Second, some bulk core storage devices are directly addressable,⁶ so that it is possible to execute directly from them without first moving information into level 0.

As a final note, the discussion surrounding Figures 4 and 5 suggests that speed ratios in the order of 1:100 between adjacent levels would lead to much less sensitivity to traverse times, and permit tighter control over thrashing. For example:

Level	Type of Memory Device	Access Time
0	thin film	100 ns.
1	slow-speed core	10 μ s.
2	very high-speed drum	1 ms.

CONCLUSIONS

The performance degradation or collapse brought about by excessive paging in computer systems, known as thrashing, can be traced to the very large speed difference between main and auxiliary storage. The large traverse time between these two levels of memory makes efficiency very sensitive to changes in the missing-page probability. Certain paging algorithms permit this probability to fluctuate in accordance with the total demand for memory, making it easy for attempted overuse of memory to trigger a collapse of service.

The notion of locality and, based on it, the working set model, can lead to a better understanding of the problem, and thence to solutions. If memory allocation strategies guarantee that the working set of every active process is present in main memory, it is possible to make programs independent one another in the sense that the demands of one program do not affect the memory acquisitions of another. Then the missing-page probability depends only on the choice of the working

set parameter τ and not on the vagaries of the paging algorithm or the memory holdings of other programs.

Other paging policies, such as FIFO or LRU, lead to unwanted interactions in the case of saturation: large programs tend to get less space than they require, and the space acquired by one program depends on its "aggressiveness" compared to that of the other programs with which it shares the memory. Algorithms such as these, which are applied globally to a collection of programs, cannot lead to the strict control of memory usage possible under a working-set algorithm, and they therefore display great susceptibility to thrashing.

The large value of traverse time can be reduced by using optimum scheduling techniques for rotating storage devices and by employing parallel data channels, but the rotation time implies a physical lower bound on the traverse time. A promising solution, deserving serious investigation, is to use a slow-speed core memory between the rotating device and the main store, in order to achieve better matching of the speeds of adjacent memory levels.

We cannot overemphasize, however, the importance of a sufficient supply of main memory, enough to contain the desired number of working sets. Paging is no

substitute for real memory. Without sufficient main memory, even the best-designed systems can be dragged by thrashing into dawdling languor.

REFERENCES

- 1 G H FINE et al
Dynamic program behavior under paging
Proc 21 Nat'l Conf. ACM 1966
- 2 P J DENNING
The working set model for program behavior
Comm ACM 11 5 May 1968 323-333
- 3 L A BELADY
A study of replacement algorithms for virtual storage computers
IBM Systems Journal 5 2 1966
- 4 J S LIPTAY
The cache
IBM Systems Journal 7 1 1968
- 5 J L SMITH
Multiprogramming under a page on demand strategy
Comm ACM 10 10 Oct 1967 636-646
- 6 R E FIKES H C LAUER A L VAREHA
Steps toward a general purpose time sharing system using large capacity core storage and TSS/360
Proc 23 Nat'l Conf ACM 1968
- 7 P J DENNING
Effects of scheduling on file memory operations
AFIPS Conf Proc 30 1967 SJCC