

# AFIPS

CONFERENCE  
PROCEEDINGS

VOLUME 26

# 1964

FALL JOINT  
COMPUTER  
CONFERENCE

1964  
SPARTAN BOOKS, INC.  
Baltimore, Md.  
CLEAVER-HUME PRESS  
London

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1964 Fall Joint Computer Conference Committee or the American Federation of Information Processing Societies.

Library of Congress Catalog Card Number: 55-44701

Copyright © 1964 by American Federation of Information Processing Societies, P. O. Box 1196, Santa Monica, California. Printed in the United States of America. All rights reserved. This book or parts thereof, may not be reproduced in any form without permission of the publishers.

Sole Distributors in Great Britain, the British Commonwealth and the Continent of Europe:

CLEAVER-HUME PRESS  
10-15 St. Martins Street  
London W. C. 2

# CONTENTS

	<i>Page</i>
Preface	
PROGRAMMING TECHNIQUES AND SYSTEMS	
CPSS—A Common Programming Support System	D. BORETA 1
Error Correction in CORC	D. N. FREEMAN 15
The Compilation of Natural Language Text into Teaching Machine Programs	L. E. UHR 35
Method of Control for Re-Entrant Routines	G. P. BERGIN 45
XPOP: A Meta-Language Without Metaphysics	M. I. HALPERN 57
EXPANSION OF FUNCTION MEMORIES	
A 10Mc NDRO BIAX Memory of 1024 Word, 48 Bit per Word Capacity	W. I. PYLE 69
	R. M. MACINTYRE
	T. E. CHAVANNES
Associative Memory System Implementation and Characteristics	J. E. MCATEER 81
	J. A. CAPOBIANCO
	R. L. KOPPEL
2Mc, Magnetic Thin Film Memory	E. E. BITTMANN 93
A Semi-Permanent Memory Utilizing Correlation Addressing	G. G. PICK 107
A 10 <sup>5</sup> Bit High Speed Ferrite Memory System- Design and Operation	H. AMEMIYA 123
	T. R. MAYHEW
	R. L. PRYOR
NEW COMPUTER ORGANIZATIONS	
An Associative Processor	R. G. EWING 147
	P. M. DAVIES
A Hardware Integrated General Purpose Computer/Search Memory	R. G. GALL 159
A Bit-Access Computer in a Communication System	E. U. COHLER 175
	H. RUBINSTEIN
Very High Speed and Serial-Parallel Computers HITAC 5020 and 5020E	K. MURATA 187
	K. NAKAZAWA
IBM System 360 Engineering	J. L. BROWN 205
	P. FAGG
	D. T. DOODY
	J. W. FAIRCLOUGH
	J. GREENE
	J. A. HIPPI

	<i>Page</i>
<b>MANAGEMENT APPLICATIONS OF SIMULATION</b>	
UNISIM—A Simulation Program for Communications Networks	L. A. GIMPELSON 233 J. H. WEBER
The Data Processing System Simulator (DPSS)	D. D. RUDIE 251 M. I. YOUCHAH E. J. JOHNSON
The Use of a Job Shop Simulator in the Generation of Production Schedules	D. R. TRILLING 277
<b>DIGITAL SOFTWARE FOR ANALOG COMPUTATION</b>	
HYTRAN—A Software System to Aid the Analog Programmer	W. OCKER 291 S. TEGER
PACTOLUS—A Digital Analog Simulator Program for the IBM 1620	R. D. BRENNAN 299 H. SANO
MIDAS—How It Works and How It's Worked	H. E. PETERSEN 313 F. J. SANSOM R. T. HARNETT L. M. WARSHAWSKY
<b>INPUT AND OUTPUT OF GRAPHICS</b>	
The RAND Tablet: A Man-machine Communication Device	M. R. DAVIS 325 T. O. ELLIS
A System for Automatic Recognition of Handwritten Words	P. MERMELSTEIN 333 M. EDEN
A Laboratory for the Study of Graphical Man- Machine Communication	E. L. JACKS 343
Operational Software in a Disc-Oriented System	M. P. COLE 351 P. H. DORN C. R. LEWIS
Image Processing Hardware for a Man-Machine Graphical Communication System	B. HARGREAVES 363 J. D. JOYCE G. L. COLE E. D. FOSS R. G. GRAY R. A. THORPE E. M. SHARP R. J. SIPPEL T. M. SPELLMAN
Input/Output Software Capability for a Man- Machine Communication and Image Processing System	T. R. ALLEN 387 J. E. FOOTE
A Line Scanning System Controlled from an On-Line Console	F. N. KRULL 397 J. E. FOOTE
<b>MASS MEMORY</b>	
A Random Access Disk File with Interchangeable Disk Kits	E. C. SIMMONS
The Integrated Data Store—A General Purpose Programming System for Random Access Memories	C. W. BACKMAN 411 S. B. WILLIAMS
The IBM Hypertape System	B. E. CUNNINGHAM 423
Design Considerations of a Random Access Information Storage Device Using Magnetic Tape Loops	A. GABOR 435 J. T. BARANY L. G. METZGER E. POUMAKIS

TIME-SHARING SYSTEMS		<i>Page</i>
The Time-Sharing Monitor System	HOLLIS A. KINSLOW	443
JOSS: A Designer's View of an Experimental On-Line Computing System	J. C. SHAW	455
Consequent Procedures in Conventional Computers	D. R. FITZWATER	465
COMPUTATIONS IN SPACE PROGRAMS		
The Jet Propulsion Laboratory Ephemeris Tape System	E. G. OROZCO	477
JPTRAJ (The New JPL Trajectory Monitor)	N. S. NEWHALL	481
ACE-S/C Acceptance Checkout Equipment	R. W. LANZKRON	489
Saturn V Launch Vehicle Digital Computer and Data Adapter	M. M. DICKINSON	501
	J. B. JACKSON	
	G. C. RANDA	
The 4102-S Space Track Program	E. G. GARNER	517
	J. OSEAS	
HYBRID/ANALOG COMPUTATION—METHODS AND TECHNIQUES		
A Hybrid Computer for Adaptive Nonlinear Process Identification	B. W. NUTTING	527
	R. J. ROY	
The Negative Gradient Method Extended to the Computer Programming of Simultaneous Systems of Differential and Finite Equations	A. I. TALKIN	539
Quantizing and Sampling Errors in Hybrid Computation	C. R. WALLI	544
NON-NUMERICAL INFORMATION PROCESSING		
Real-Time Recognition of Hand-Drawn Characters	W. TEITELMAN	559
A Computer Program Which "Understands"	B. RAPHAEL	577
A Question-Answering System for High School Algebra Word Problems	D. G. BOBROW	591
The Unit Preference Strategy in Theorem Proving	L. WOS	615
Comments on Learning and Adaptive Machines for Pattern Classification	D. CARSON	
	G. ROBINSON	
	C. H. MAYS	623
HARDWARE DESIGNS AND DESIGN TECHNIQUES		
FLODAC—A Pure Fluid Digital Computer	R. S. GLUSKIN	631
	M. JACOBY	
	T. D. READER	
Design Automation Utilizing a Modified Polish Notation	W. K. ORR	643
Systematic Design of Cryotron Logic Circuits	J. M. SPITZE	
	C. C. YANG	651
	J. T. TOU	
Binary-Compatible Signed-Digit Arithmetic	A. AVIZIENIS	663
HYBRID/ANALOG COMPUTATION— APPLICATIONS AND HARDWARE		
A Transfluxor Analog Memory Using Frequency Modulation	W. J. KARPLUS	673
	J. A. HOWARD	
The Use of a Portable Analog Computer for Process Identification, Calculation and Control	L. H. FRICKE	685
	R. A. WALSH	
Progress of Hybrid Computation at United Aircraft Research Laboratories	G. A. PAQUETTE	695
A Strobed Analog Data Digitizer with Paper Tape Output	R. L. CARBREY	707
Hybrid Simulation of Lifting Re-Entry Vehicle	A. A. FREDERICKSON, JR.	717
	R. B. BAILEY	
	A. SAINT-PAUL	

# CPSS

## A COMMON PROGRAMMING SUPPORT SYSTEM

*Dushan Boreta*  
*System Development Corporation, Falls Church, Virginia*

### INTRODUCTION

Over the years many computer software systems have been developed to serve the program production process. These systems, variously known as "production" systems, "utility" systems, or "support" systems, are designed and produced for the same purpose: *to provide programmers the tools required to produce computer programs*. Beyond this common purpose these systems have little in common and, in fact, are unique systems individually tailored to a particular application. In each system much of the tailoring occurs because of the particular computer configuration, operational system support requirements, computer manufacturer's software characteristics, experience of the designers, schedule pressures, and style preferences of the programmers producing the system. The tailoring is reflected in the design of each program production system and is evident in many features, for example, the programming languages used, the computer operating procedures, the programmer's inputs, the outputs provided to the programmer, and the program organization in the system.

In examining program production systems, most are found to have functional capabilities for generating code, code-checking the object programs, and maintaining magnetic tapes containing programs.

In some instances these capabilities are of the most rudimentary sort. In other instances, very sophisticated and complete capabilities exist.

What this paper describes is a program production system, CPSS, that should assist programmers and managers in the performance of their tasks. The principle characteristics of CPSS provide for programmers an efficient and effective means for producing their programs. For managers, CPSS provides for the minimization of costs for producing programs, and a relatively inexpensive means for achieving an effective and efficient program production capability.

The CPSS characteristics that make these claims a reality are: first, it provides to programmers the attributes of higher order languages in each program production task; second, that both the functions of CPSS and its computer programs largely are transferable; and third, the totality of functions of a comprehensive program production system is provided in CPSS. Further, the design features embodied in CPSS should afford the minimization of its maintenance costs, reduction in the possibility of programmer errors, and simplification of the programming task itself.

Additionally, the design of CPSS provides for its "common" applicability. It may be used in "open-" or "closed-shop" operations in supporting the development and production of system, non-system, and "one-shot" programs.

Effectively, its design characteristics, language power, scope of applicability, and transferability make CPSS an *off-the-shelf* program production system.

CPSS is programmed in a subset of the JOVIAL language, and in design is compatible with the full JOVIAL language. Currently, CPSS is implemented on an IBM 7090 and is being used to support the development and production of a computer program system. This installation and continued testing will be the source for refinements to CPSS's design as the system continues under development.

### CPSS DESIGN CRITERIA AND REQUIREMENTS

Providing "off-the-shelf" capability is a different type of programming problem than normally is encountered. The problems in providing CPSS with the "off-the-shelf" capability stem from the class of computers on which it may be installed; the nature of the transferability task; some aspects of the programmer training tasks; the CPSS maintenance task; the programming language it provides; and the scope of its applicability to the operational system development process.

#### *Class of Computers*

CPSS is directly applicable to medium- and large-scale computers. The computer configuration should have, but need not be restricted to, a word size of 30 bits, a 32K one-instruction-per-word or a 16K two-instruction-per-word core memory, peripheral storage units consisting of four tape drives (or three tape drives plus drum or disc units), an on-line printing device, an on-line input device, and some external switches or keys.

The computer configuration need not be defined explicitly in that there are many possible trade-offs between the computer's characteristics, the programming conventions and techniques used in CPSS, and the capacity of the system. For example, by altering the labeling convention used in the coding of CPSS, the class of computers could be expanded to include machines with a word size of 24 bits.

#### *The Transferability Task*

The transferability of a program production system is important for many reasons. The cost of installing a program production system is minimized. For applications employing a variety of computers, there is a standard system

and methodology that contributes to programmer transferability. The difficulties and costs inherent in the transition from one computer to another are reduced. And, a bench mark is identifiable from which further technology development may progress.

The goal, transferability of programs, usually is interpreted as requiring a program coded and operating on one computer to be operable on a different computer and still retain the capability to perform its functions. The transfer should be completed at least semi-automatically, utilizing clerical or junior personnel and fixed procedures. The current state of the art does not afford 100 per cent transferability. Therefore, we have interpreted this goal to mean that CPSS is to be transferable with only a *minimum of known code change*. In order that CPSS be transferable, the functions and services provided by CPSS also must be transferable. Additionally, the CPSS documentation, program and system tests, operating procedures, transferability techniques, and transfer procedures are designed to be transferable.

It must be emphasized that the transferability task being discussed is that of getting CPSS to run on another computer, different from its current application (the IBM 7090). CPSS is designed to be transferred as a system. Although it is modular, the transferability of any module is a distinctly different task from that of transferring the whole of CPSS.

One natural design feature of a transferable system is its independence from machine characteristics. It must be noted that machine independence is a two-way street. Not only is the *code* of CPSS to be machine-independent, but the *functions* performed by the code also must be machine-independent. For example, programs making transfers to and from storage should not assume some given unit availability, transfer rate, segmentation of the transferred data, unit positioning, or even that it is the only user of a unit. In CPSS, this example of machine-independence (transferability) is provided by a central I/O program in the CPSS Computer Operations Subsystem (the design of which is discussed later in this paper).

It will be difficult to measure how transferable CPSS is until it has been transferred

across several computers. The system's transferability could be measurable in several dimensions, for example, in time elapsed from start to installation, in dollar costs for each economic factor involved in the transfer, in amounts and types of computer time required, in the amount of code to be altered per program and per function, and in the number of errors discovered in each phase, including installation and post-installation. Detailed records should be maintained that identify the transfer costs and the factors that influenced the cost. Some of these factors are: the differences in the machine instruction word format and addressing, the types and quality of programs available on the "new" machine, the frequency of occurrence and amount of down-time per occurrence of machine failure, and the location and availability of the staging and target computers.

Some of the principal technical problems that will arise in transferring CPSS to other computers lie in the sophistication of the design embodied in the system, the power of the language provided by the system, the systemization of the CPSS design, and the broad class of computers to which CPSS may be applied. For example, consider the problem of designing CPSS so that it will operate on a four-tape drive computer configuration. The task of compiling a JOVIAL program can use (1) an input device for the source program, (2) an input device for the library tape, (3) an input device for the system Compool, (4) a temporary storage device for the intermediate language, (5) a permanent storage device for the object program, (6) an output device for the listings, and (7) an output device to communicate with the computer operator (as will be noted later, a compilation may require other additional "storage devices"). Further complicate the task and allow the programmer to generate a test case and operate his program on the test case, and allow all this to occur in an uninterrupted single job. This problem is resolved in the design of the CPSS executive and I/O functions (discussed later in the paper). In essence, the I/O problem was resolved by constructing a central I/O program that provided machine-independent I/O functions for the remainder of CPSS. The control problem was resolved by allowing programmers the freedom of directing CPSS

via control card inputs (functionally oriented to the program production tasks).

All the tasks related to transferability are *not* involved in subsequent transfers of CPSS. Consider the input card; once we have coded routines to accept floating-field cards and have levied no special format requirements on the card, the card input processing functions are totally independent of the machine. The information processed from card inputs in one application need not appear on cards in another application but could be processed from other input media in a different input form, e.g. punched paper tape or teletypewriter.

The methods employed in achieving transferability, or machine-independence, vary depending on the function being performed in the program. Some of the more commonly applied techniques were: the parameterization of certain machine characteristics (word length, number of characters per print line, number of print lines per page, etc.); the establishment of programming conventions regarding the use of constants and tags; the use of floating-field card formats; and the use of "all-core" indexing to relocate data and to compute addresses. In many instances, special methods were required to achieve transferability. Some of these are discussed later in the paper during the discussions of the various CPSS subsystems.

#### *Programmer Training*

One of the principal benefits achieved by employing a higher order language and requiring transferability in CPSS is in the potential reduction of programmer training costs.

When a programmer is transferred from one application to another, a training or learning period is required to familiarize him with the particular computer and the program production system he will use. This retraining period varies from a week to a month-and-a-half or more. During this period a programmer's effectiveness is almost nil; and thereafter, it is less than it should be until the programmer becomes expert in the use of the "new" computer and system.

CPSS should afford a reduction of training and retraining costs by permitting programmers to code and test their programs in a higher



order language. If CPSS achieves a broad patronage, these costs are further reducible since CPSS is designed to reflect a stable form regarding its interfaces with the programmer. In effect, CPSS could become a means for achieving some level of *programmer transferability*.

#### *CPSS Maintenance*

In producing CPSS, a primary concern has been maintenance costs. These costs are related to error correction, program improvement, augmentation of the system's capability, program and system documentation, and product release. The design of CPSS provides for the minimization of such costs by isolating, identifying and documenting the program- and system-type functions that comprise CPSS.

The system and program documentation are designed to facilitate the maintenance task. A CPSS program's documentation consists of a heavily commented program manuscript, a detailed flow chart, a functionally organized flow chart, a program description document containing descriptions of data referenced, each routine coded, each procedure used, the input data formats and structures, the output tables, items and messages, and the function served by the program. Another document describes each machine dependency contained in the program. Also, a system design document describes each program function and the interfaces between programs.

CPSS is designed and documented to facilitate the maintenance task. Also, the system is capable of maintaining itself or of producing itself.

#### *The Programming Language*

Perhaps the most significant decision made in the CPSS project was the selection of a language for the programming of CPSS. The design of each function contained in a program production system is influenced by the language provided by the system. Therefore, certain design features are required to assure that the program production system is capable of responding to the operational system's programming needs. In a sense, a transferable program production system must be "overdesigned". The design must reflect the current capability of the

language being provided, and also needs to provide for logical extensions of the language. Consider the situation that exists with CPSS.

There are three levels of JOVIAL represented in CPSS which form a hierarchy of language that is upward compatible in language power and in the language processing algorithms. The formal JOVIAL, J-3, is subset into two levels: J-S, being a subset of J-3; and J-X, being a subset of J-S. The program generation subsystem is coded in J-X and processes programs that are written in J-S. All other subsystems are coded in J-S and perform their functions compatibly with J-3.

The decision range as to which level of language capability is to be provided in the program production system is bounded on the upper end by the formal definition of the programming language, and on the lower end by the language capability provided by the program generation subsystem.

In the program generation subsystem, the language capability to be provided is influenced by such factors as the subsequent use of the language, the design of the compiler, the level of transferability desired, and the expected characteristics of existing languages and compilers having the same generic name.

Other factors influencing the decision in CPSS were the transferring procedures and techniques, the testing techniques established to test the system, and the availability of computers with JOVIAL compilers.

#### *CPSS—Scope of Applicability*

CPSS serves programmers and managers in their performance of several tasks related to the system development process. Figure 1. shows a simplified representation of the system development process that has been employed for several systems, both large and small. The scope of CPSS is indicated by its applicability to the program production process, which encompasses parts of the program design, program generation, program test, and assembly test stages.

Figure 2. is a simplified representation of the program production process as served by program production systems. The programs of

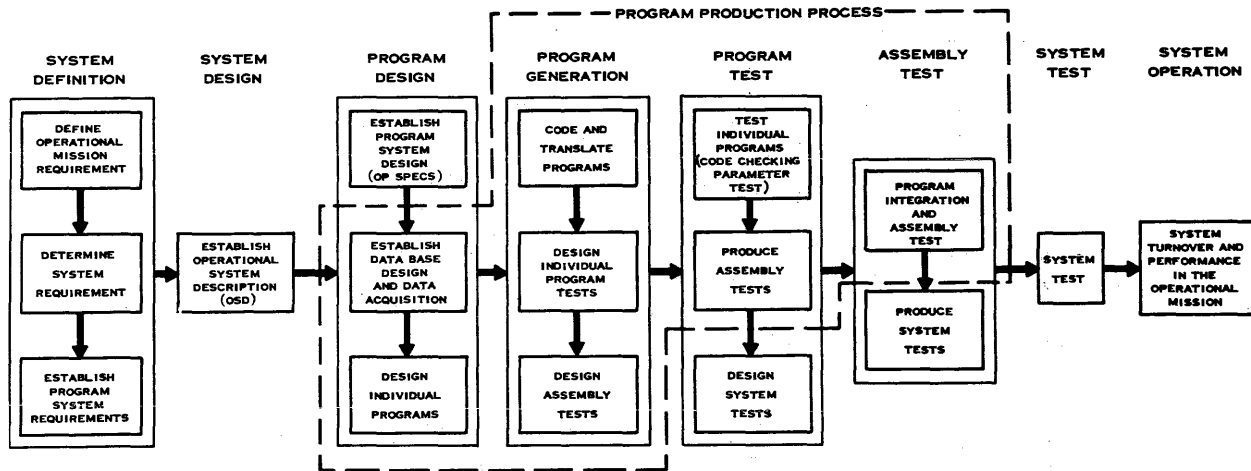


Figure 1. Program Production Process in the System Development Process

the production system are designed to assist programmers and managers in their performance of these four tasks: program generation, program test, system generation, and assembly test.

The principal product derived from the program production process is the operational program system master tape. Other products are a system data dictionary (referred to as a Compool) with its documentation and listings, the program system documents, listings, test plans, and test results, and the programs that comprise the program system with their documents, listings, test plans, and test results.

A major task, related to the system generation task, is the acquisition and management of a data base. This paper will not delve into the data base tasks except where such tasks directly interface with the program production system.

Figure 3. depicts the information and data flow provided for in CPSS. The flow of data

between the various functions is automatic. The execution of the functions is controlled by the programmer. The four program production tasks, program generation, program test, system generation, and assembly test are served by this data and information flow.

The preceding figures, Figure 1., Figure 2., and Figure 3., depict the scope of applicability for CPSS in the system development process.

*Program Generation.* The programmer, employing the JOVIAL language, encodes a program to satisfy the program design specifications. The code, the symbolic programming language statements (the source program), is input to the compiler which translates the code into machine instructions. During the compilation, the source program is appropriately augmented by routines from the procedure library tape and by system data descriptions from the Compool.

The principal output from the compiler is a binary program (object program). The re-

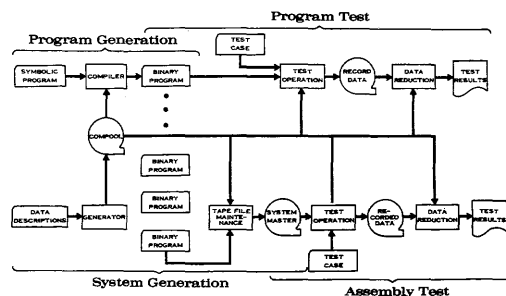


Figure 2. Program Production Process

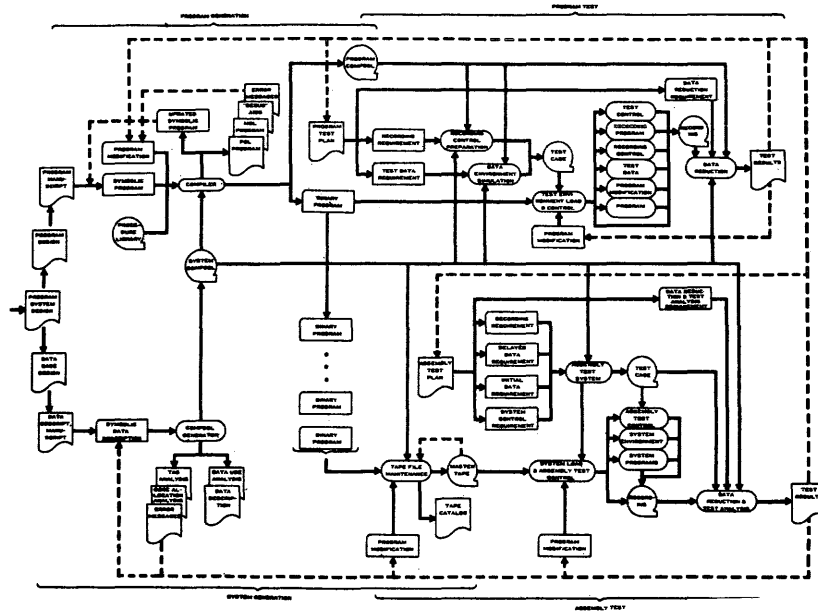


Figure 3. Program Production Process, Information and Data Flow

remainder of the outputs provide information to the programmer (and to other parts of CPSS) that facilitate the testing and correction of the program. The process of compilation in the early phases of program coding sometimes is referred to as "grammar-checking"—where the result of the grammar-checking is a "good" program; that is, one that is syntactically correct.

*Program Test.* In order to test a program—that is, validate that the program performs its functions correctly—the programmer must define a test case. A test case is comprised of a simulated data environment for the program, recording controls to retrieve data from the program's environment, and any program modifications required to correct the program as shown by previous tests.

The test case is input to CPSS which translates the programmer's inputs into a test environment. When requested, CPSS loads the test environment and the object program into the computer for operation. During the operation of the test, data is recorded as requested by the programmer in the test case. After the operation of the test, the recorded data is processed to provide, as outputs, the hard copy test results.

CPSS appropriately interprets data descriptions from the program Compool or the system

Compool to translate the test case inputs and process the recorded data. Essentially, the system Compool and the program Compool are the significant means through which CPSS affords the programmer the ability to test a program at a language level comparable to a higher order programming language.

This loop, the program test phase, is repeated for as many test cases as are required to satisfy that the program performs its functions correctly.

The two tasks discussed so far, program generation and program test, are common to all program production processes—whether the programs are system programs or independent programs. In this light, the applicability of CPSS is extended to include both system and non-system programming tasks.

*System Generation.* One of the principal tasks in building a system is to define the system data dictionary, more commonly known as a Compool. Essentially, the Compool is the means for defining the data that comprises a system's data base. A Compool can be thought of as being a central repository of data descriptions used both by programmers and programs. Usually, a Compool exists in two forms. The first is a document containing descriptions of the system's data environment, data structures,

data organizations, some commentary related to the reasons why the data exists in the system, and a description of the usage of the data. The second form is a binary tape containing information describing data structures, and data organizations. The binary Compool, in some cases (including CPSS), contains information that is usable in the constructing of the Compool document. The program production system itself is a principal user of the binary Compool in that it retrieves data descriptions from the Compool during the various phases of the program production process.

The Compool, being a central collecting point for system data descriptions, serves as an integrating device in the program production process. In this manner the Compool provides to program system managers a means for controlling a system's data environment.

CPSS provides both for the building of the binary Compool and for the Compool documentation. A programmer employing the appropriate data descriptors, encodes data description statements that describe the data comprising the data base. These statements are interpreted by the Compool generator which produces a binary Compool. Other outputs provided by CPSS are quality analysis aids, and data description listings.

The tape file maintenance function provides the means for building tapes containing programs. Further, the function provides for modifying, correcting, cataloguing, and in general maintaining computer tapes.

*Assembly Test.* The assembly test task, functionally, is similar to the program test task. The purpose of the assembly test task is to provide a means for testing a complex of programs that form a system or a logical subset of a system. In other words, the purpose served in assembly testing is to validate that a complex of programs acting in concert perform a system function correctly. Assembly testing can be thought of as a hierarchy of testing—ranging from simple program interface tests to complex full system tests.

Figure 3. depicts an assembly test as being performed in a controlled environment. The system control parameters, initializing data, simulated inputs, and recording controls are

prepared as a test case via an *assembly test system*. The test case is run against the appropriate complex of programs during which recording is performed. After operation on the test case, the recorded data is processed via a data reduction and test analysis subsystem which provides the hard copy test results. This loop, assembly testing, is performed for as many test cases and test levels as are required to validate that the program system performs its functions correctly.

Although CPSS is not designed explicitly to serve the assembly test task, it does contain programs that are usable in an assembly test system; for example, the data recording, data reduction, and data generation programs. With very minor modifications to the test environment load and data reduction programs, CPSS further could be used to provide a very sophisticated string test (program interface test) capability.

The reason for not explicitly providing an assembly test capability in CPSS is that the higher levels of assembly testing usually require programs that reflect the design of the operational program system (such as height reply message simulators, and radar correlation analysis programs).

## CPSS PROGRAM DESIGN

One of the principal design characteristics of CPSS is the functional modularity embodied both in CPSS and its programs. CPSS has been separated logically into subsystems, in general, corresponding to the common program production functions: program generation, data environment simulation, data recording, data reduction, test environment load, computer operation, Compool generation, and tape file maintenance. These subsystems are comprised of programs which further are partitioned into functional subroutines. An attempt was made to isolate each system function and each program function into an identifiable subpart of CPSS. Some of the common program-type functions have been programmed as JOVIAL procedures and loaded onto the CPSS procedure library tape. Additionally, the CPSS programs, tables, items and the Compool itself are defined in the Compool. Thus, CPSS is an integrated system constructed of modules, each of which

are program-type or system-type functions, which are organized as 22 major programs, 35 library procedures, 10 common executive entries, 25 system tables with 330 items, and 53 parameter items. The size of CPSS is approximately 20,000 JOVIAL statements that result in approximately 65,000 IBM 7090 machine instructions.

#### *Program Generation Subsystem*

The program generation function is provided in CPSS by the JOVIAL language and a JOVIAL compiler. With the development of CPSS, a powerful and comprehensive subset of the JOVIAL language was developed that should be sufficient to produce most computer software systems. This subset, the JOVIAL core-subset language, J-S, is the language employed in the programming of CPSS. The power of J-S is demonstrated by the fact that the programming of CPSS did not require the totality of J-S.

The principle reasons for developing J-S, and the goals achieved by this development were:

- (1) The definition of a "comprehensive minimum" JOVIAL language that is sufficient for producing most computer program systems.
- (2) The definition of a JOVIAL subset language that affords the production of transferable programs.
- (3) The design, development, and production of a JOVIAL compiler that can be produced on shorter schedules than more comprehensive J-3 compilers.
- (4) The improvement of the language processing speed of a JOVIAL compiler.
- (5) The retention of the significant language and compiler features normally expected of JOVIAL, for example: Compool sensitivity, procedure library capability, partitioning of programs into procedures and closed routines, memory allocation, packing of items into tables, processing of packed data, grammar checking, subscripting and indexing, bit and byte addressing, machine assembly language coding, logical and arithmetic operations, and program "debug" listings and aids.

In general, the differences between the J-S and J-3 languages should be more than offset by the improvements in the compiler design and its compatibility to CPSS. It should be noted that J-S is a proper subset of JOVIAL, i.e., that the programs coded in J-S are legal and valid inputs to J-3.

Some of the significant design features of the J-S compiler are:

- (a) The J-S compiler is a "two-pass" compiler. That is, a program is processed twice to produce a binary output. First, in the JOVIAL language form; and second, in an intermediate language form. The principal result of having only two passes is that compiling speed has been significantly increased.
- (b) The J-S compiler provides an "alter-mode" of recompilation. That is, the programmer can add modifications to the source program during compilation without altering the original source program. The compiler will produce an updated version of the source program as one of its selectable options.
- (c) The J-S compiler produces a program Compool. That is, the J-S compiler produces a Compool containing complete data descriptions of all data and labels referenced or declared by the program. The program Compool is usable interchangeably with the system Compool throughout CPSS and is compatible in form and structure with the system Compool.
- (d) The J-S compiler is capable of being expanded to incorporate additional language capability. The practical limitation on this expandability is the size of core memory.

Additionally, the programmer can select the outputs he wants, override the Compool, specify the Compool he wants used, and in general, exercise those options that specifically control the inputs to and the outputs from the compiler.

In general, the CPSS program generation subsystem provides the language power, compiler speed, and flexibility of use that affords a programmer the ability to generate almost any program conceivable.

### *Data Environment Simulation Subsystem*

The data environment simulation function is provided in CPSS by a computer program that processes data assignment statements. The program produces data records containing the programmer specified data, and control information that is used by the test environment load subsystem.

The programmer specifies his data environment requirements in a POL-type language. The program employs either a program Compool or a system Compool as selected by the programmer. The programmer also may identify the data being produced, thereby affording future selective use of the data. The program is compatible with the JOVIAL J-3 data forms and data structures.

The data itself can be coded as floating-point, fixed-point, integer, Hollerith, Standard Transmission Code, and status-variable, where such data is organized and structured as tables, items, strings, or arrays. Subscripting and indexing also is allowed.

The program allows the programmer to set values into any variable defined or referenced by his program if the program Compool is used. If a system Compool is used, the programmer may set values into any variable defined in the system Compool. The program imposes no limits on the volume of data it processes. It does perform legality checks on the programmer's inputs to determine the compatibility of the inputs with the defined data environment.

### *Data Recording Subsystem*

The data recording function in CPSS is separated into two parts, where the actual data recording is provided under the CPSS computer operation subsystem. The CPSS data recording preparation function is performed by a computer program that processes recording request statements. The program produces a data record containing control information for use by the computer operation subsystem, the test environment load subsystem, and the data reduction subsystem.

The programmer describes his recording requests in a fully symbolic language. The CPSS program employs either a program Compool or a system Compool as selected by the program-

mer. The programmer also may identify the recording that would be performed per his requests, thereby affording future selective use of the recording controls produced by the CPSS program.

The programmer may select the data to be recorded under any name defined in the program Compool or system Compool and/or any block of memory. The location in his program at which recording is to take place can be specified symbolically (if a program Compool is selected).

The programmer may request a memory register change survey, or dumps before, during, and/or after his program's operation. The dumps may be formatted as octal, machine language instructions, floating-point, and/or alphameric.

### *Data Reduction Subsystem*

The data reduction function is provided in CPSS by a subsystem of computer programs that process either CPSS recorded data or miscellaneous data formats. There are four general classes of printouts produced by CPSS: Compool-defined data, memory dumps, survey dumps, and tape dumps.

Compool-defined data is processed and appropriately formatted entirely dependent upon the Compool definition. CPSS interrogates either the system Compool or program Compool to determine the appropriate formatting. The information in a printout reflects the page number, table name, recording identity, recording location, table size, entry number, data name, data type, the converted data, and a security classification.

Memory dump processing is performed in any of four formats: octal, machine language instructions, floating-point, and/or alphameric. A printout contains the page number, security classification, recording location, recording identity, the beginning and ending locations of the dump, the contents of the addressable machine registers, and the contents of the computer words dumped. The page formatting is determined by the program and is printed as four or eight words per print line.

The survey dump processing is similar to the memory dump processing. The significant dif-

ference is that those memory locations which contain changed values are printed. The dumps to be compared are made by the CPSS recording program before and after the operation of a program in a test. A printout contains the page number, security classification, the recording identity, the beginning and ending locations of the survey area, the contents of the addressable machine registers in the "before" and "after" states, and a print-line for each changed computer word containing the address and both values.

In the tape dump function CPSS provides the means to print out any tape. The format of the printout is similar to the memory dump format, except that the addressable machine register print lines are not output.

The programmer can request that his data reduction be performed in either of three modes: recording, general, or binary.

The recording mode is used to process data recorded by CPSS. The programmer can select a subset of the data to be processed, or he can allow CPSS to process the data automatically per his recording requests. Processing selection is performed in exactly the same manner as specifying recording requests.

The binary mode is used to process tape dumps. The programmer can specify the "limits of processing" and the print formats (octal, floating-point, etc.). The limits of processing bounds the range of tape that is to be processed.

The general mode is used as a mixture of the recording and binary modes. All the controls available to the programmer under these two modes are available under the general mode. Further, if a tape containing records similar to CPSS-type records is to be processed, the general mode may be used to reduce the data per Compool definitions even though the tape was not built by the CPSS recording program.

#### *Test Environment Load Subsystem*

The test environment load function is provided in CPSS by a computer program that loads a test case into the computer for operation.

The CPSS test environment load subsystem provides for loading recording patches to a pro-

gram; loading a data environment; loading octal correctors to a program; and loading the program that is to be tested (currently, on the IBM 7090, CPSS is capable of loading and operating a 25,000 register test case).

#### *Computer Operation Subsystem*

The computer operation function is performed in CPSS by a subsystem of programs that provide for the uninterrupted operation of the computer.

The functions performed by the computer operation subsystem can be grouped into four classes: system control, operator communication, test control, and I/O monitor.

*System Control.* The system control function provides for the continuous operation of CPSS. It interrogates programmer supplied control "cards" to determine which function (or subsystem) is required. The system operates on stacks of jobs (usually prestored on tape in the sequence desired) where each job may be comprised of many dissimilar requests. For example, a job may be to compile a program, specify several sets of recording controls, specify several data environments, to load and execute the compiled program in various test environments, and to process the data recorded in the several program runs. The sequence of CPSS's operation is specified by the ordering of the programmer supplied control cards. In addition to the sequence control function, system control provides the normal control-type functions such as position tapes, clear core, job error recovery, loading of octals to cycling system programs, etc. Essentially, the system provides uninterrupted operation as long as there are jobs to be processed, and a test program does not loop or write into "permanent" core. Controls are provided through which the computer operator (or the programmer in one special case) may interrupt the computer's operation.

*Operator Communication.* The operator's communication function provides three methods for interrupting the system's operation, (1) at each I/O operation, (2) between control cards, and (3) recovery from test program loops, halts or other errors. When the operator has completed his tasks, he may recover the sys-

tem's operation, as appropriate, in any of five ways, (1) skip forward in the job to the next control "card", (2) skip forward to the next job, (3) skip forward to a specified job, (4) reinitialize the system, or (5) continue from the point of interruption. Recovery may be performed automatically by CPSS as a result of the operator's request, or the operator manually may enter the system as he desires. While the system is interrupted, the operator may reassign I/O units, list the I/O unit/file allocation, take dumps, position to a particular job, or perform other similar tasks. The operator may perform his tasks in response to programmer-supplied instructions (as printed by CPSS or otherwise), messages printed by CPSS relating to the system's needs, or recognizable error conditions requiring his actions.

CPSS provides a method for programmers to "simulate" certain computer operator actions. They may specify I/O allocations, list the I/O unit/file allocations, or perform other operator type tasks. By the judicious use of control cards the programmer may "directly" communicate with the computer operator to effect the job desired.

*Program Test Control.* The program test control provides for the operation of a test case loaded by the test environment load subsystem. Also, the program test control function executes the recording program for dumps and surveys, and loads system recovery type program modifications to the object program. The interfacing between the test environment load subsystem and other computer operation functions provides to programmers almost complete flexibility in the running of tests. CPSS allows any one of its functions to be run independently or in any sequence. Some of the types of computer runs a programmer might make are:

- (1) Compile only
- (2) Load and execute his program
- (3) Compile, load, and execute his program
- (4) Generate test data
- (5) Specify recording requirements
- (6) Load his program, recording controls, test data and execute his program
- (7) Reload data and re-execute his program
- (8) Load a different program, its recording controls, and execute the "new" program on the "old" data environment

(9) Re-execute his program

Effectively, CPSS imposes no operating restrictions on the programmer in the generation or testing of a program. In this manner the programmer is able to selectively test subparts of his program, his whole program, or strings of programs all as one job or independent jobs.

*I/O Monitor.* The principle function performed by the CPSS central I/O program is to provide machine-independent I/O operations for other programs. The I/O program performs all the I/O operations required by the programs comprising CPSS. The program provides a comprehensive set of I/O operators: Read, Read-search, Write, Position, Position-search, Close, Wait (for a specific file), Wait (on all files), Repeat (the preceding request), Rewind (initialize the file), and File-status (feed back the current status of the file). Additionally, the program provides some elementary data conversion and manipulation functions in conjunction with requested I/O transfers, i.e., transfer to or from packed or unpacked BCD data images, convert data to or from "standard transmission code", convert data to or from BCD, and/or combinations thereof. Also, the program will transfer data to or from specific locations or standard locations. The program will either wait for a transfer to be completed or return immediately as requested.

A program requests I/O operations by setting items in a CPSS communication table and transferring control to the I/O program. These items specify the name of the file on which the operation is requested, the operation to be performed, a wait or no wait condition, and other information related to the operation such as data conversion and manipulation, location of the data to be transferred, amount of data to be transferred, etc.

Upon completing the operation, the I/O program automatically enters information into the communication table relating to the requested operation and returns control to the requesting program. This information is usable to determine the status of the file, file addressing, status of the requested operation, amount of data transferred, etc.

In essence the I/O program determines the appropriate device, record fragmentation (or



accumulation), labeling, unit positioning, and other functions to effect a transfer of data to or from memory via an I/O device. The program monitors each transfer to determine the validity of the transfer and takes whatever corrective action is appropriate. The manner in which these functions are performed provides CPSS programs their independence from a machine's I/O and yet allows the referencing programs to perform "efficient" I/O.

#### *Compool Generation Subsystem*

The Compool generation function is provided in CPSS by a subsystem of computer programs that build and interpret a Compool. The CPSS Compool subsystem provides for a comprehensive definition of data. The inputs to the Compool assembler contain the normal type of data definitions, and a variety of supplementary data descriptive information (see Appendix B). Further the Compool assembler provides for assigning data addresses symbolically, and allocates core memory for data or program storage. Effectively, the Compool assembler program provides the ability to define data for system applications, normal utility type needs, and for the programmer's information needs. It facilitates the data description task by accepting a fully symbolic input. In that the contents of a Compool usually are operational system-dependent, the Compool assembler program provides for the definition of the Compool's content. The program interrogates a series of legality matrices to determine the acceptability of data, the validity of the input, and the completeness of the data definition. In this manner the Compool constructed by CPSS can be tailored to the operational system's needs.

Also, the Compool subsystem contains a program whose function is to retrieve the information contained in the Compool. This retrieval program provides two levels of information: first, that information which is required for the normal utility type functions, and second, all the information contained in the Compool. In the first case the data is printed in alphabetical order, and in the second case is alphabetized by data class. The third function provided by the CPSS Compool subsystem is a quality analysis of the information contained in the Compool.

The program performs a tag analysis function that checks for duplicated tags and ambiguous cross-reference tags. The program also determines the validity of the memory allocation by checking for violations of reserved areas, and overlapped allocations of data. In addition the program performs a capacity analysis by checking for unallocated addresses, and by tallying each data occurrence by data type and amount of memory required. Essentially, CPSS provides a Compool that is tailored to a system by its content and the tools needed to build, interrogate and provide quality control on a Compool.

#### *Tape File Maintenance Subsystem*

The tape file maintenance function is provided in CPSS by a computer program that performs those functions necessary to maintain tapes produced by or for CPSS. Further, the CPSS program is capable of performing the same set of functions on almost any tape regardless of format or structure.

Some of the more significant characteristics of the CPSS program are: it can duplicate, reformat, position, read, write, close, skip, backspace, rewind, compare, list the contents of, and load octals to tapes containing programs, Compools, files, records or any combinations thereof.

The CPSS program interrogates control cards containing information that describes the operation to be performed, the units on which the CPSS program will operate, and the structure of the data stored on the unit. The CPSS program provides for labeling of each transfer, and thereby can handle overlaid and interspersed files of varying structures.

The CPSS program is designed in modules such that each operator, and each modifier to the operator are procedures or closed routines. With this design the CPSS program easily can be modified to delete, add or modify the tape file maintenance functions as the particular application requires.

The tape file maintenance program establishes information in "dummy entries" for use by the CPSS central I/O program. In this manner the only machine-specifics in this program lie in its processing of binary cards .

## APPENDIX A

*CPSS Control*

The sequence of the tasks performed by CPSS is dictated by the ordering in the programmer's job deck. A job deck is comprised of system control cards, and data and/or function control cards. Data and function control cards im-

'ASSIGN, INPUT, unit \$  
 'CLEAR, area to be cleared \$  
 'COMMENT \$ commentary  
 'COMPILE \$  
 'COMPOOL, ANALYSIS, control information \$  
 'COMPOOL, ASSEMBLE \$  
 'COMPOOL, AUDIT \$  
 'COMPOOL, DISASSEMBLE, control information \$  
 'COMPOOL, LIST, control information \$  
 'ENDJOB \$  
 'GO, address \$  
 'JOB \$  
 'LOAD, control information \$  
 'OCTAL \$  
 'OPSIM \$  
 'POSN, file name, control information \$  
 'PROCESS, control information \$  
 'RECORD, control information \$  
 'RETURN, address \$  
 'TABSIM, control information \$  
 'UTILITY \$  
 'WAIT \$

mediately follow their related system control card in the job deck. The first card in a job deck is the 'JOB card and the last card is an 'ENDJOB card.

The system control cards acceptable to CPSS are listed below and summarize some of the system's capabilities.

Pairs the CPSS INPUT file to the given unit.  
 Clears the given area of core to plus zero.  
 The comment is printed.  
 Initialize the program generation subsystem.  
 Analyze a Compool as requested.  
 Assemble a Compool from the data cards.  
 Legality check the Compool data cards.  
 Format and print the binary Compool specified.  
 Format, order, and print the Compool specified with commentary added.  
 The end bracket for a job deck.  
 Transfer control to the given address.  
 The begin bracket for a job deck.  
 Load the environment specified.  
 Load and save octals for CPSS programs.  
 Initialize the operator "simulation" function.  
 Position the given file as directed.  
 Format and print the data as directed.  
 Prepare recording parameters as directed.  
 Load a transfer to the CPSS executive at the given address.  
 Prepare a data environment as directed.  
 Initialize the tape file maintenance subsystem.  
 Stop the system's operation.

## APPENDIX B

*CPSS Compool*

The CPSS Compool assembler builds a Compool from information coded on data declaration "cards". The program processes nine data declarator types which are used to define a system's data base, i.e., Program, Table, Item, String, Array, Free Item, Constant, File, and Task declarations. Also, the program processes four declarator types that are used in the building of a Compool, i.e., Ident, Locate, Reserve, and End declarations.

1. *Ident*. The Ident declaration is used to identify the Compool itself.

2. *Locate*. The Locate declaration is used to pair address labels to core memory addresses. These labels are usable in lieu of actual memory addresses. In this manner, the programmer is able to allocate memory and define data addresses in a completely symbolic method.

3. *Reserve*. The Reserve declaration is used to prevent the allocation of data to certain core memory areas.

4. *End*. The End declaration terminates the program's processing of declarations.

The type of information the programmer may use to describe data is quite comprehen-

sive. For example, a program description may contain the program name, mod, length, memory location status (absolute, relocatable, or dynamically relocatable), memory location, program type (closed or open, system program, parameterless subroutine, or parameterized subroutine), storage location (unit, label, unit addressing), subsystem name, title, related commentary, and input and output parameters.

## BIBLIOGRAPHY

The following represents a collection of general material on the subject of "program production" type systems and supplementary references for CPSS and JOVIAL.

1. BARNETT, N. L., FITZGERALD, A. K., "Operating System for the 1410/7010-360 philosophy", *Datamation*, Vol. 10, No. 5, pp. 39-42, May 1964.
2. BLATT, J. M., "Ye Indiscreet Monitor", *Communications of the ACM*, Vol. 6, No. 8, pp. 506-510, September 1963.
3. BORETA, D., "Introduction to CPSS", (soon to be published as a System Development Corporation tech memo, TM-WD-800/002/00).
4. BOUVARD, J., "Operating System for the 800/1800—admiral", *Datamation*, Vol. 10, No. 5, pp. 29-34, May 1964.
5. HOWELL, H. L., The Q-32 JOVIAL Operating System, System Development Corporation, TM-1588/000/00, November 1963.
6. OLIPHINT, C., "Operating System for the B5000—Master Control Program", *Datamation*, Vol. 10, No. 5, pp. 42-45, May 1964.
7. PERSTEIN, M. H., The JOVIAL Manual, Part 2, The JOVIAL Grammar and Lexicon, System Development Corporation, TM-555/002/02, March 1964.
8. SCHWARTZ, J. I., COFFMAN, E. G., WEISSMAN, C., A General-Purpose Time-Sharing System, Proceedings Spring Joint Computer Conference, Washington, D. C., pp. 397-411, April 21-23, 1964.
9. SHAW, C. J., "A Specification of JOVIAL", *Communications of ACM*, Vol. 6, No. 12, pp. 721-735, December 1963.
10. SHAW, C. J., The JOVIAL Manual, Part 1, Computers, Programming Languages and JOVIAL, System Development Corporation, TM-555, Part 1, December 1960.
11. SHAW, C. J., The JOVIAL Manual, Part 3, The JOVIAL Primer, System Development Corporation, TM-555/003/00, December 1961.
12. STEEL, T. B., Jr., "Operating Systems—boon or boondoggle", *Datamation*, Vol. 10, No. 5, pp. 26-28, May 1964.
13. SUTCLIFFE, W. G., Program Production System User's Manual (1604-A JOVIAL Compiler—OASIS Utility), System Development Corporation, TM-WD-402/000/00, January 1964.
14. SWANSON, R. W., SPASUR Automatic System Mark 1, Utility System Users Manual, System Development Corporation, TM-WD-28, July 1964.
15. VER STEEG, R. L., "TALK—A High-Level Source Language Debugging Technique with Real-Time Data Extraction", *Communications of the ACM*, Vol. 7, No. 7, pp. 418-419, July 1964.
16. CO-OP Manual, Control Data 1604 User's Group, Control Data Corporation, No. 067a, December 1960.
17. Cosmos IV Manual, System Development Corporation, TM-LX-81/001/00, October 1963.
18. H 800 Survey Guide, Honeywell, Electronic Data Processing Division.
19. IBM System/360 Special Support Utility Programs, IBM Corporation, File No. S360-32, Form C28-6505-0, 1964.
20. IBM System/360 Programming Systems Summary, IBM Corporation, File No. S360-30, Form C28-6510-0, 1964.
21. IBM 7090/7094 IBSYS Operating System, System Monitor (IBSYS), IBM Corporation, File No. 7090-36, Form C28-6248-1, 1963.
22. Philco 2000 Operating System SYS Version E, Philco Corporation, January 1963.
23. RCA 501 Electronic Data Processing System, EDP Methods, Radio Corporation of America, Technical Bulletin No. 16.
24. SCOPE/Reference Manual, CDC 3600, Control Data Corporation, August 1963.

# ERROR CORRECTION IN CORC, THE CORNELL COMPUTING LANGUAGE

*David N. Freeman*  
*IBM General Products Division Development Laboratory*  
*Endicott, New York*

## I. INTRODUCTION

CORC, the Cornell Computing Language, is an experimental compiler language developed at Cornell University. Although derived from FORTRAN and ALGOL, CORC has a radically simpler syntax than either of these, since it was designed to serve university students and faculty. Indeed, most of the users of CORC are "laymen programmers," who intermittently write small programs to solve scientific problems. Their programs contain many errors, as often chargeable to fundamental misunderstandings of the syntax as to "mechanical errors." A major objective of CORC is to reduce the volume of these errors. This objective has been achieved to the following extent: the average rate of re-runs for 4500 programs submitted during the fall semester of 1962 was less than 1.1 re-runs/program.

Three features of CORC have enabled it to achieve this low re-run rate:

- (1) Inherent simplicity of the syntax;
- (2) Closed-shop operation of the Cornell Computing Center on CORC programs, including keypunching, machine operation, and submission/return of card decks;
- (3) A novel and extensive set of error-correction procedures in the CORC compiler/monitors.

The CORC language is briefly described below; it is more fully documented elsewhere.<sup>1</sup>

The current paper describes the error-correction procedures in greater detail.

## II. THE CORC LANGUAGE

CORC was designed by a group of faculty and students in the Department of Industrial Engineering and Operations Research at Cornell. This group has coded and tested two similar compiler/monitor systems, one for a medium scale decimal computer and the other for a large binary computer.

During the definition of the language, the design group surrendered potency to simplicity whenever the choice arose. Certain redundancies have been included in CORC, serving two functions: to facilitate error-correction during source-deck scanning, and to aid novice programmers' grasp of compiler-language syntax. Excepting these redundancies, CORC is quite frugal with conventions. For example, all variables and arithmetic expressions are carried in floating-point form, avoiding the confusing notion of "mode." At the same time, programmers are spared all knowledge of floating-point arithmetic.

Each CORC card deck is divided into three required sub-decks plus an optional sub-deck of data cards:

- (a) The *preliminary-description cards* supply heading data for each page of the output listing.

- (b) The *dictionary cards* declare all variables used in the program, simple as well as subscripted.
- (c) Each *statement card* may have an indefinite number of continuation cards. Statements may bear *labels* having the same formation rules as variables. Continuation cards may not be labelled.

Variables, labels, numbers, reserved words, and special characters comprise the *symbols* of CORC. Each symbol is a certain string of at most eight, non-blank characters. *Numbers* may have up to twelve digits; decimal points may be leading, trailing, or imbedded in the numbers. There are forty-three *reserved words* in CORC, e.g., LET, and ten special characters: + - \* / \$ = ( ) . , The character string defining each label, variable, or reserved word is terminated by the first blank space or special character. The character string defining each number is terminated by the first character that is neither a digit nor a decimal point. Each special character is a distinct symbol. There are forty-six *legal characters* in CORC: letters, digits, and special characters.

A subset of the reserved words is the set of fifteen *first-words*: LET, INCREASE, INC, DECREASE, DEC, GO, STOP, IF, REPEAT, READ, WRITE, TITLE, NOTE, BEGIN, and END. The first symbol in each statement should, if correct, be one of these first-words.

There are eight executable-statement types, plus a NOTE statement for editorial comments on the source-program listing. (NOTE statements may be labelled; in this case, they are compiled like FORTRAN "CONTINUE" statements.) To simplify the description of the statement types, single letters denote entities of the CORC language:

- V . . . . . a variable, simple or subscripted
- E . . . . . an arithmetic expression, as defined in FORTRAN
- L . . . . . a statement label
- B . . . . . a repeatable-block label (see below)
- R . . . . . one of the six relational operators: EQL, NEQ, LSS, LEQ, GTR, and GEQ. A *relational expression* is a predicate comprising two arith-

metic expressions separated by a relational operator, e.g., 2\*X NEQ 0.9.

The statement types are as follows:

- (1) LET V = E, and two variants INCREASE V BY E and DECREASE V BY E. (INCREASE may be abbreviated to INC, DECREASE to DEC.)
- (2) IF E<sub>1</sub> R E<sub>2</sub>  
     THEN GO TO L<sub>1</sub>  
     ELSE GO TO L<sub>2</sub>, and two variants  
     IF E<sub>11</sub> R<sub>1</sub> E<sub>12</sub>      IF E<sub>11</sub> R<sub>1</sub> E<sub>12</sub>  
     AND E<sub>21</sub> R<sub>2</sub> E<sub>22</sub>      OR E<sub>21</sub> R<sub>2</sub> E<sub>22</sub>  
     .  
     .  
     AND E<sub>N1</sub> R<sub>N</sub> E<sub>N2</sub>      OR E<sub>N1</sub> R<sub>N</sub> E<sub>N2</sub>  
     THEN GO TO L<sub>1</sub>      THEN GO TO L<sub>1</sub>  
     ELSE GO TO L<sub>2</sub>      ELSE GO TO L<sub>2</sub>.
- (3) GO TO L.
- (4) STOP, terminating execution of a program.
- (5) READ V<sub>1</sub>, V<sub>2</sub>, . . . , bringing in data cards during the execution phase. Each data card bears a single new value for the corresponding variable.
- (6) WRITE V<sub>1</sub>, V<sub>2</sub>, . . . , printing out the variable names, the numerical values of their subscripts for each execution of the WRITE statement, and the numerical values of these variables.
- (7) TITLE (message), printing out the remainder of the card and the entire statement fields of any continuation cards.
- (8) REPEAT B . . . , comprising four variants
  - (8a) REPEAT B E TIMES,
  - (8b) REPEAT B UNTIL E<sub>11</sub> R<sub>1</sub> E<sub>12</sub>  
     AND E<sub>21</sub> R<sub>2</sub> E<sub>22</sub>  
     .  
     .  
     AND E<sub>N1</sub> R<sub>N</sub> E<sub>N2</sub>,
  - (8c) REPEAT B UNTIL E<sub>11</sub> R<sub>1</sub> E<sub>12</sub>  
     OR E<sub>21</sub> R<sub>2</sub> E<sub>22</sub>  
     .  
     .  
     OR E<sub>N1</sub> R<sub>N</sub> E<sub>N2</sub>,

- (8d) REPEAT B FOR V =  $E_1, E_2, \dots, E_i, E_j, E_k, \dots$ , where  $(E_i, E_j, E_k)$  is an *iteration triple* as in ALGOL.

Closed subroutines—called *repeatable blocks* in CORC—are defined by two *pseudo-statements* as follows:

```
B   BEGIN
      .
      .
      .
B   END,
```

where the “B” labels appear in the normal label field. A repeatable block can be inserted anywhere in the sub-deck of statement cards; its physical location has no influence on its usage. It can only be entered under control of a REPEAT statement (with a few erroneous-usage exceptions).

Repeatable blocks may be nested to any reasonable depth. Any number of REPEAT statements can call the same block, although the blocks have no dummy-variable calling sequences. All CORC variables are “free variables” in the logical sense, which avoids confusing the novice programmer no less than it hampers the expert programmer.

### III. ERROR ANALYSIS IN CORC

In the CORC compiler/monitor, the author and his colleagues have attempted to raise the number of intelligible error messages and error-repair procedures to a level far above the current state-of-art for similar systems. The success of these messages and procedures is measured by three economies:

- (a) reduced re-run loads,
- (b) reduced costs of card preparation, and
- (c) less faculty/student time devoted to tedious analyses of errors.

The detection of each error invokes a message describing the relevant variables, labels, numbers, etc.; why they are erroneous; and what remedial actions are taken by CORC. Exhibiting errors in detail has improved student comprehension of the CORC syntax. Of course, certain errors defy detection, e.g., incorrect numerical constants.

A principal tenet of the CORC philosophy is to detect errors as early as possible in:

- (1) characters within symbols,
- (2) symbols within expressions,
- (3) expressions within statements, e.g., the left and right sides of an assignment statement, and
- (4) statements within the sequencing of each program.

An explicit message for each error is printed on the output listing. This listing is the only output document from a CORC program; all programs are compiled and executed, and machine code is never saved on tape or punched cards.

After detecting a statement-card error, CORC always “repairs” the error by one of the two following actions:

- (a) CORC refuses to compile a “badly garbled” statement. Instead, CORC replaces it with a source-program “message statement” reminding the programmer of the omitted statement.
- (b) CORC edits the contents of a “less badly garbled” statement into intelligible source language. The edited statement is subsequently compiled into machine code.

Errors in cards other than statement cards are repaired by similar techniques.

Thus, the machine code produced by CORC is always executable, and compilation-phase and execution-phase error messages are provided for every program. By continuing compilation in the presence of errors, CORC provides diagnostic data simultaneously on structural levels (1)–(4) cited above. By also executing these programs, CORC detects additional errors in program flow, subscript usage, improper function arguments, etc.

The *correction* of a programming error is defined to be the alteration of relevant source-language symbols to what the programmer truly intended. Under this operational definition, many errors are incapable of “correction,” e.g., the programmer may have intended a statement or expression not even offered in CORC. Other errors are capable of “correc-

tion” by the programmer himself but by no critic unfamiliar with the complete problem-definition; an incorrect numerical constant is again an example.

A third class of errors can be corrected by an intelligent critic after scanning the source-deck listing, without recourse to the problem definition. Some errors in this class require a profound use of context to elicit the programmer’s true intention. Other errors in this class can be detected and corrected with little use of context, e.g., the omission of a terminal right parenthesis.

The author defines a *corrigible error* to be one whose correction is automatically attempted by the CORC compiler/monitor. Thus, this definition is by cases, for a specific version of CORC. CORC may correct one error and fail to correct a second, nearly-identical error. Error correction is a fundamentally probabilistic phenomenon; the CORC error-correction procedures attempt to maximize the “expected useful yield” of each program by strategies based on *a priori* probabilities associated with the different errors.<sup>2\*</sup>

The majority of corrigible errors are detected during the scanning of source decks by the CORC compiler. A few corrigible errors are detected during the execution of object programs. For each error, one or more correction procedures have been added to CORC, representing certain investments in core memory and operating speed.

The following paragraph discusses the selection of corrigible errors, and section IV catalogues these errors. The catalogue will be somewhat peculiar to the structure of CORC, a population of novice programmers, and the operation of a university computing center. However, the discussion of control-statement errors, arithmetic-expression errors, and misspellings is relevant to most compiler languages.

The author has roughly ranked various error conditions by two criteria: *a priori* probabilities<sup>†</sup> of their occurrence, and *a priori* probabilities of their correction (if correction is at-

\* References 2 and 5 also propose probabilistic correction of misspellings.

† Probabilities in the sequel are estimates based on human scrutiny of several hundred student programs.

tempted). Correction procedures were designed for some errors, while other chronic errors had such low *a priori* probabilities of correction that only explicit error-detection messages were printed out. For example, omission of a subscript is a common error which is difficult to correct, although easy to detect and “repair.” CORC “repairs” a subscript-omission error by supplying a value of 1.

On the other hand, misspellings are common errors whose *a priori* probabilities of correction are high if sophisticated procedures are used. The author hopes to achieve at least 75 percent correction of misspellings with the current procedures; many have not yet been tested in high-volume operation.<sup>3‡</sup>

#### IV. ERROR CORRECTION DURING SCANNING

First, the general procedures for card scanning will be described. The second, third, and fourth subsections deal with dictionary cards, data cards, and statement cards, respectively. The last subsection describes the error-correction phase which follows scanning, i.e., after the last statement card has been read but before machine code is generated by the compiler.

##### A. CARD SCANNING

Each CORC source deck should have all cards of one type in a single sub-deck:

- (1) Type 1, preliminary description cards
- (2) Type 5, dictionary cards
- (3) Type 0, statement cards
- (4) Type 4, data cards (if used).

The *type* of each card is defined by the punch in column 1 (although CORC may attempt to correct the type of a stray source card).

At the beginning of each new source program, CORC scans the card images (usually on magnetic tape) for the next type 1 card, normally a tab card bearing any non-standard time limit and page limit for this program. (The tab cards are used to divide the decks, facilitating batch processing and other handling.) This scanning procedure skips any extraneous data cards from the previous pro-

‡ Damereau has achieved over 95% correction of misspellings in an information-retrieval application.

gram deck. If the preceding deck was badly shuffled, misplaced dictionary cards and statement cards will also be skipped.

An indefinite number of type 1 cards may be supplied: CORC inserts data from the first two cards into the page headings of the output listing. This serves to label all output with the processing date and programmer name, avoiding losses in subsequent handling.

The problem identification should be duplicated into each deck; any deviations from this identification generate warning messages. The serialization of cards is checked, although no corrective action is taken if the cards are out of sequence. If the serialization is entirely omitted, CORC inserts serial numbers into the print-line image of each card, so that subsequent error messages can reference these print lines without exception.

The general procedure on extraneous or illegal punches is as follows: illegal punches are uniformly converted to the non-standard character " $\neq$ "; extraneous punches are ignored except in non-compact variable/label fields and in the statement field of type 0 cards, where all single punches are potentially meaningful. Rather than discard illegal punches, CORC reserves the possibility of treating them as misspellings. Likewise, any non-alphabetic first character of a variable/label field must be erroneous and is changed to " $\neq$ ," furnishing a later opportunity to treat this as a misspelling. All hyphen punches are converted to minus signs during card reading; the keyboard confusion of these two characters is so chronic—and harmless—that CORC even refrains from a warning message.

## B. DICTIONARY CARDS

Although the dictionary and data cards are processed in entirely different phases of a CORC program, their formats are identical—with the exception of column 1—and common procedures are used to scan them. As mentioned in the preceding subsection, non-alphabetic first characters are changed to " $\neq$ ." Embedded special characters are similarly changed with the following exception: character strings of the form "(I)" or "(I,J)" are omitted. Fixed-column subscript fields have al-

ready been provided and students consistently and correctly use them. However, a common student error is to supply redundant parenthesized subscripts in the label field; these are ignored by CORC, although a warning message is supplied.

Non-numeric characters in the subscript fields and the exponent field are changed to "I"s. Vector subscripts can appear in either the first-subscript field or the second-subscript field. These subscripts need not be right-justified in their respective fields. After an array has been defined, subsequent subscripts of excessive magnitude are not used; the corresponding data entries are put into the highest legal cell of the array.

## C. DATA CARDS

All of the foregoing procedures apply with these exceptions: if a data card has its variable field blank or, in the case of subscripted variables, its subscript fields blank, the data can still be entered with a high probability of correcting the omission. Information in the READ statement overrides incorrect or missing entries on the corresponding data cards. CORC insists on exact agreement of the variables and subscripts if warning messages are to be avoided. Symbolic subscripts may be used in READ statements, but their execution-phase values must agree with the numeric subscripts on the type 4 cards.

## D. STATEMENT CARDS

Correction of erroneous statement cards is a complex technique—and the most fruitful of those currently implemented in CORC. Statement cards comprise over 80% of student source decks, on the average. Students commit the overwhelming majority of their errors in communicating *imperative* statements to a compiler, rather than *header* statements, *declarative* statements, or *data* cards. Statement-card errors fall into two major categories: those detectable at compilation time and those detectable only at execution time. The second category is discussed in section V. Some of the most useful correction techniques for the first category—tested and modified during the past two years of CORC usage—are described in the following eight sub-sections.



(1) Misspellings<sup>4,5</sup>

At the end of Section III, misspellings were cited as a class of errors that both occur frequently and have attractively high *a priori* probabilities for correction. Accordingly, CORC now contains a subroutine that compares any *test word* to any list of words (each entry being denoted a *list word*), determining a "figure of merit" for the match of each list word to the test word. Each figure of merit can be considered as the *a posteriori* probability that the test word is a misspelling of this particular list word. The list word with the highest figure of merit is selected as the spelling of the test word "most likely" to be correct.

Various categories of misspelling are defined in CORC; to each category is assigned an *a priori* probability of occurrence. When the test word and a list word match within the scope of a category, i.e., the test word is some particular misspelling of the list word, the *a priori* probability for this category is added to the figure of merit for this list word. Actually, the figures of merit are integers rather than probabilities; they can be converted to probabilities by the usual normalization, but this is unnecessary—they are used merely to rank the possible misspellings.

All increments used in misspelling analyses reflect the number  $N$  of non-blank characters in the test word, as follows: a certain base-value increment is specified for each misspelling; if a match is found, this base value is multiplied by the ratio  $N/8$ , then added to the corresponding figure of merit.

(a) A *concatenation misspelling* occurs when a delimiting blank is omitted between two symbols, e.g., "LET X . . ." is a concatenation misspelling of "LET X . . ." When such a misspelling is detected, any relevant list of words is compared against the concatenated symbol. The increment to the figure of merit for each list word is computed as follows:

- (i) If the list word and the test word do not have at least their initial two characters in common, the increment is 0.
- (ii) For every consecutive character in common with the list word (after

the first character), an increment of 2 is added to the figure of merit.

Example: Assume that the test word is ENTRYA and that two of the list words are ENT and ENTRY. The corresponding figures of merit are 6 and 10, respectively. The higher figure reflects the more exact agreement of ENTRY to ENTRYA.

(b) Single-character misspellings provide four different increments to the figure of merit, corresponding to mutually exclusive possibilities:

(i) A *keypunch-shift misspelling* occurs when the IBM 026 keypunch is improperly shifted for the proper keystroke, e.g., a "1"-"U" error. There are fourteen possible misspellings of this type, corresponding to the seven letter-number pairs on the keyboard. The special character row, including "0," does not seem susceptible to misspelling analysis, since special characters are always segregated, never imbedded in symbols.

For each list word which agrees within a single keypunch-shift misspelling with the test word, an increment of  $(20N/8)$  is added to the corresponding figure of merit, where  $N$  is the number of non-blank characters in the test word.

(ii) An *illegal-character misspelling* occurs either (a) when a variable/label has previously required a "single-letter perturbation" using the character "≠" or (b) when an illegal punch in the card is changed to "≠." Single-letter perturbations are used when the same symbol occurs at both a variable and a label, or when a reserved word is used as a variable or label. In either case, conflicting usage cannot be tolerated, and CORC appends "≠" to the symbol for the current usage. In subsequent searches of the symbol dictionary, one may wish to recognize the orig-

- inal spelling. Thus, for each list word which agrees within a single illegal-character misspelling with the test word, an increment of  $(20N/8)$  is added to the corresponding figure of merit, where  $N$  is as above. This increment is higher than that for a random misspelling, reflecting the peculiar origins of the character "≠."
- (iii) A *resemblance misspelling* occurs whenever any of the following character pairs is confused: "I"- "1," "O" (the letter)-"0" (the number) and "Z"- "2." For each list word which agrees within a single resemblance misspelling with the test word, an increment of  $(40N/8)$  is added to the corresponding figure of merit, where  $N$  is as above.
- (iv) A *random misspelling* occurs when any other single character is mispunched in a symbol. For each list word which agrees within a single random misspelling with the test word, an increment of  $(10N/8)$  is added to the corresponding figure of merit, where  $N$  is as above.
- (c) A *permutation misspelling* provides a single increment to a figure of merit whenever the test word matches the corresponding list word within a pair of adjacent characters, this pair being the same but permuted in the two words, e.g., LTE is a permutation misspelling of LET. For each list word which agrees within a single permutation misspelling with the test word, an increment of  $(20N/8)$  is added to the corresponding figure of merit, where  $N$  is as above. Other permutations may deserve consideration at some future date, but adjacent-pair permutations seem to have the highest *a priori* occurrence probabilities.
- (d) Simple misspellings of the foregoing types have high probabilities of successful correction insofar as the following conditions are met:
- (i) The list of words does not contain many nearly-identical entries. Otherwise, there will be many reasonable misspelling possibilities from which the program may select only one.
  - (ii) Neither test words nor list words are single-character symbols. The program excludes such list words from consideration during a misspelling analysis; experience has shown that only a small proportion—perhaps 10 percent—of single-character symbols are successfully corrected.
  - (iii) Context can be extraordinarily helpful. Associated with each list word is a set of attributes such as the count of its usage in the current program, its function (variable, label, constant, reserved word, etc.), and any peculiar usages already detected (such as being an undeclared variable). Certain misspelling possibilities can be immediately discarded if the context associated with the corresponding list words does not match the context of the test word. For example, if an arithmetic statement is being analyzed, any test for misspelled variables can immediately discard all misspelled label possibilities.
- The first two of these three conditions are controlled by the vocabulary of the source-deck programmer; CORC gives far better assistance to programs using only a few variables and labels of highly distinctive spelling with at least three characters apiece.
- (e) The increments corresponding to different misspellings were arbitrarily selected; they can be readily raised or lowered as experience indicates. The current values reflect the following observations:
- (i) The weakest communication link is between the handwritten coding sheets and their interpretation by the keypunch operator. Hence, the

largest increment is assigned to resemblance misspellings.

- (ii) In lieu of exact information, permutation misspellings and key-punch-shift misspellings have been judged equally probable.
- (iii) Illegal punches in a card image arise from three sources: illegal hole patterns, improper use of a character (e.g., non-alphabetic character beginning a first word, or the duplicate use of a symbol as two entities), and card-reading failures. Lacking other evidence, the author considered the increment to be approximately the same as in (ii).
- (iv) Other single-character misspellings seem only half as likely to occur.

Examples of the current CORC misspelling analyses may be found at the end of subsection E on *Post-Scanning Spelling Corrections*.

## (2) Subscripts

Correction attempts for subscript errors have low success probabilities, on the whole. Isolated omission of one or both subscripts seems almost hopeless. CORC edits such an omission by appending "(1)" to a vector variable and "(1,1)" to a matrix variable. Likewise, if a matrix variable has other than two subscripts, CORC uses primitive editing techniques to produce executable machine code. Excessive commas are changed to "+" signs, and "(E)" is changed to "(E, 1)," where "E" is the arithmetic expression for the first subscript of a matrix variable.

Missing right parentheses are supplied and extra right parentheses are deleted as necessary, although not always correctly.

Definition of new array variables after the dictionary is complete (i.e., after all type 5 cards have been processed) is an attractive—if difficult—error-correction procedure. Most algebraic compilers scan source decks several times; they have a leisurely opportunity to accumulate evidence for undeclared array variables. If such evidence is overwhelming, i.e., if every usage of a certain variable is immediately

followed by a parenthesized expression, these compilers could change the status of this variable before the final code-generation scan.

To reduce compilation time, the current version of CORC scans each source statement once and must make an immediate decision when it finds a left parenthesis juxtaposed to a supposedly simple variable: should "V(. . .)" be changed to "V\*(. . .)," i.e., implied multiplication, or should it be treated as a subscript (and re-designate "V" as an array variable)? The present error-correction procedure is to encode "V(. . .)" into the intermediate language without change; special counters for usage as a vector/matrix variable are incremented, depending on one/two parenthesized arguments. At the conclusion of scanning, these usage counters are tested for all "simple" variables. Any variable used preponderantly as a vector variable causes CORC to test for the misspelling of some declared vector variable. Failing this, CORC changes the status of the variable to a vector of 100 cells. Any variable used preponderantly as a matrix variable causes CORC to test for the misspelling of some declared matrix variable. Failing this, CORC changes the status of the variable to a matrix of 2500 cells, comprising a  $50 \times 50$  array.

If a variable is infrequently juxtaposed to parenthesized expressions, CORC treats these juxtapositions as implied multiplications. Deferral of this decision necessitates a procedure for inserting the multiplication operator during the conversion of intermediate language to machine code, together with the appropriate message. This error-correction procedure is one of the few in the code-generation phase. The message appears at the end of the source-deck listing rather than adjacent to the offending card image; the gain in error-correcting power seems to justify deferring the message.

The *a priori* probabilities of omitted array-variable declarations and implied multiplications are both high. Since the two possibilities are mutually exclusive, CORC bases its choice on the percentage occurrence of the ambiguous usage. If the usage is chronic, i.e., comprising more than 50 percent of the total usage of some variable, an undeclared array variable seems more probable. If the ambiguous usage is a

small percentage of the total usage, implied multiplication seems more probable.

### (3) Arithmetic and relational expressions

The rules for analyzing and correcting arithmetic expressions are as follows:

- (a) Extraneous preceding plus signs are deleted, and preceding minus signs are prefixed by zero, i.e., “-E” becomes “0-E.”
- (b) Thereafter, “+,” “-,” “\*,” and “/” are all binary operators. If an operand is missing before or after a binary operator, the value “1” is inserted. This merely preserves the coherence of the syntax; to correct this error seems hopeless.
- (c) If an expression using two binary operators might be ambiguous (irrespective of the formal syntax), CORC prints out its resolution of the ambiguity, e.g., “A/B\*C IS INTERPRETED AS (A/B)\*C.”

### (4) LET, INCREASE-BY, and DECREASE-BY

Four components are essential to each correct statement in this category: the first-word, the assigned variable, the middle symbol, and the right-hand-side (RHS) arithmetic expression.

- (a) The first-word of the statement has been identified by a generalized pre-scan of the statement. If “LET” has been omitted but “=” has been found, CORC furnishes the former symbol.
- (b) The assigned variable may be subscripted; if so, CORC supplies any missing arguments, commas, and right parentheses when “=” or “BY” terminates the left-hand-side (LHS) of the statement. If other symbols follow the assigned variable but precede “=” or “BY,” they are ignored.
- (c) “EQU,” “EQL,” and “EQ” are erroneous but recognizable substitutes for “=.”
- (d) Any arithmetic expression is legal for the RHS.

### (5) GO TO, STOP, and IF

- (a) With one exception—(b) just below—all unconditional branches begin with “GO,” followed by an optional “TO.”
- (b) STOP is a complete one-word statement. Also, it may be used in the conditional-branch statement, e.g., “IF . . . THEN STOP ELSE GO TO . . .”
- (c) A conditional branch always follows one or more relational expressions in an IF or REPEAT statement. For IF statements, the first incidence of “THEN,” “ELSE,” “GO,” “TO,” or “STOP” terminates the last relational expression; missing operands, commas, and right parentheses are then inserted as needed. Thereafter, the two labels are retrieved from any “reasonable” arrangement with two or more of the above five words.

Missing labels are replaced by dummy “next statement” labels, which later inhibit the compilation of machine-code branches. Thus, if an IF statement lacks its second label, the falsity of its predicate during execution will cause no branch. At the end of scanning, certain labels may remain undefined; here also, CORC inhibits the compilation of machine-code branches.

### (6) REPEAT

- (a) If the repeated label is omitted, e.g., in the statement REPEAT FOR ARG = 2, CORC scans the label field of the following source card. Programmers often place repeatable blocks directly after REPEAT statements using these blocks: Hence, any label on this following card is likely to be the missing repeated label: it is inserted into the REPEAT statement. If no such label is found, CORC creates a dummy label for the repeatable block. During the execution of the program, usage of this erroneous REPEAT statement can be monitored by this dummy label.
- (b) If the REPEAT-FOR variant is used, CORC tests for three components in addition to the repeated label:
  - (i) The bound variable, i.e. ARG in the example in 6(a).

- (ii) The character “=” or its erroneous variants “EQU,” “EQL,” and “EQ.”
  - (iii) Any collection of iteration triples and single arithmetic expressions, separated by commas. In any iteration triple, CORC will supply a single missing argument with value “1.”
- (c) As in IF statements, an indefinite number of relational expressions can be used in REPEAT-UNTIL statements.

### (7) BEGIN and END

REPEAT statements and repeatable blocks require consistent spelling of labels and matching BEGIN/END pseudo-statements. Through misunderstanding or carelessness, novice programmers commit grievous errors in using REPEAT statements and their blocks. CORC attempts to correct a certain subset of errors whose correction probabilities are attractively high:

- (a) If the label of a BEGIN pseudo-statement is missing, the preceding and following cards are tested for clues:
  - (i) if the preceding card was a REPEAT statement using a yet-undefined label, this label is supplied to the BEGIN pseudo-statement.
  - (ii) If (i) fails to hold and if the following card is labelled, this label is shifted to the BEGIN pseudo-statement.
  - (iii) Otherwise, a dummy label is supplied, awaiting further clues to the identity of the repeatable block. If such clues never appear, the block is closed by a CORC-supplied END pseudo-statement after the last statement card of the deck.

Should an unpaired END pseudo-statement be subsequently found, the dummy label (on the BEGIN pseudo-statement) is changed to match this unpaired END label.

- (b) If the label for an END pseudo-statement is missing, CORC tests for the

existence of a “nest” of unclosed blocks. If so, the label of the innermost unclosed block is used in the current END pseudo-statement. Otherwise, the card is ignored.

- (c) If the label in an END pseudo-statement does not match the label of the innermost unclosed block, the current label is tested against the labels of the entire nest of blocks. If a “crisscross” has occurred, i.e.,

```
A . BEGIN
.
.
B . BEGIN
.
.
A   END,
```

CORC inserts the END pseudo-statement for block B before the current END pseudo-statement for block A.

- (d) If the preceding test fails, CORC again tests the current label against the nest, looking for a misspelling. If the current label is misspelled, procedure (c) is used. If the misspelling tests fail, CORC ignores the END pseudo-statement.
- (e) If the student has programmed an apparent recursion, CORC prints a warning message but takes no further action. Although unlikely, there may be a legitimate use for the construction:

```
A   BEGIN
.
.
.
REPEAT A ...
.
.
.
A   END.
```

In this situation, CORC makes no attempt to preserve the address linkages as a truly recursive routine would require. Thus, the program is likely to terminate in an endless loop.

### (8) READ and WRITE

Only simple or subscripted variables can appear in READ statements. The subscripts can

be any arithmetic expressions. If a label appears in the argument list of a WRITE statement, the current count of the label usage will be printed. Constants, reserved words, and special characters are deleted from the argument lists of READ/WRITE statements.

#### E. POST-SCANNING SPELLING CORRECTIONS

The misspelling of labels and variables is corrected—insofar as CORC is capable—after scanning an entire deck, with the exceptions mentioned in section D. After scanning, much usage and context data have been accumulated. CORC attempts to resolve suspicious usages by equating two or more symbols to the same entity.

When the implementation of CORC was originally under study, heavy weight was given to the potential benefits from correcting misspellings. Efficient correction of misspellings seemed to require one of the following similar strategies:

- (a) Two or more complete scans of the source deck, the first serving primarily for the collection of data on suspicious usages such as possible misspellings.
- (b) Encoding of the source deck into an intermediate language which is tightly packed and substantially irredundant but which also permits re-designation of labels and variables after misspelling analyses.

A third alternative to these strategies was to compile the source deck directly into machine code, then attempt to repair this code after determining the set of corrigible misspellings. However, this procedure seemed less flexible to use and more difficult to program than the first two strategies; it was rejected from consideration.

The second alternative was selected and appears in both current implementations of CORC. Details of the strategy are as follows:

- (a) Each new simple variable entered into the dictionary is paralleled by a *pointer-cell* containing the address of a second cell. This address is ordinarily used during machine code-generation to rep-

resent the variable in question. Since any misspelled variable is equated to a properly-spelled variable *after* scanning but *before* code generation, CORC corrects the misspelling merely by giving the variables identical pointer-cell contents.

- (b) Each new array variable is paralleled by a pointer-cell containing the base address of the array. As for simple variables, only one pointer cell is changed if this variable is equated to another array variable.
- (c) To each label corresponds a pointer-cell containing a branch instruction to the appropriate machine location (when the latter becomes defined during the generation of machine code). For an undefined label equated to some other label, its cell is filled with a branch instruction to the pointer-cell for the other label. Thus, execution of GO TO LABELA, where LABELA is a defined label, requires two machine-language branch instructions; if LABELA is an undefined label equated to LABELB, three machine-language branch instructions are required.

The penalty in compilation speed for using the intermediate language is modest: the average time to complete compilation for CORC programs—after the last statement card has been read—is less than one second; few decks require more than two seconds.

#### (1) Correction of misspelled labels

If a label has been referenced but never defined in a label field, it is tested for being a possible misspelling of some defined label. The defined label with the highest figure of merit is selected and the following message is printed:

LABELA IS CHANGED TO LABELB,

where LABELA and LABELB are the undefined and defined labels, respectively. If no defined label has a non-zero figure of merit with respect to the undefined label, the following message is printed:

LABELA IS UNDEFINED

Subsequently, all references to this label during the generation of machine language are

treated as “next-statement” branches. At execution time, any GO TO or REPEAT statements referencing this label cause the following messages, respectively:

IN STATEMENT \_\_\_\_\_,  
GO-TO NOT EXECUTED.

IN STATEMENT \_\_\_\_\_,  
REPEAT NOT EXECUTED.

(2) Correction of misspelled simple variables

(a) If an undeclared variable is never used in suspicious juxtaposition to parenthesized expressions (cf. subsection D(2) above), CORC attempts to find a declared simple variable meeting the following criteria:

- (i) The undeclared variable is a potential misspelling of the declared variable.
- (ii) The LHS-RHS usage of the declared variable is complementary to that of the undeclared variable. By *LHS-RHS usage* is meant the following two frequencies:
  - (aa) Usage on the LHS of an assignment statement, in a READ statement, or in the initial dictionary. This usage corresponds to assigning the variable a new value.
  - (bb) Usage on the RHS of an assignment statement, in a relational expression, or in a WRITE statement. This usage corresponds to using the current value of the variable.

The motivation for LHS-RHS analysis is the following: if two variables are spelled almost identically, if one has a null RHS usage and the other a null LHS usage, then the *a priori* probability that the programmer intended a single entity is higher than the probabilities for most alternative misspellings.

CORC does not use LHS-RHS analysis alone to determine the best misspelling possibility. Instead, an increment of 5 is added to the figure of merit of each declared variable whose null usage complements any null usage of the current test word, i.e., undeclared variable. Un-

declared variables can be equated only to declared variables, not to other undeclared variables.

(b) If a declared variable has a null RHS usage, it may be an erroneous dictionary spelling of some variable which is thereafter consistently spelled. However, CORC will announce that the dictionary spelling is “correct” in this case, after it detects the misspelling; all “misspelled” incidences of the variable are equated to the declared variable.

(3) Examples

Four groups of nearly-matching symbols are illustrated in Table I. In the first group, the label ABC requires testing for misspelling. The label ABCDE is a concatenation misspelling, figure of merit (FOM) = 6. The label ABD is a random misspelling, FOM = 3. The label BAC is a permutation misspelling, FOM = 7. The label AB≠ is an illegal-character misspelling, FOM = 7. Thus, CORC would choose at random between BAC and AB≠ for the defined label to which ABC should be equated.

In the second group, the defined label DEI has FOM = 15 with respect to the undefined label DE1.

In the third group, three simple variables have not been declared in the dictionary and require testing for misspelling. One should remember that only declared simple variables, i.e., XYZ and XYU, are eligible for identification with the undeclared variables. With respect to XYV, XYZ has misspelling FOM = 3; to this must be added the null-RHS increment of 5, making a total FOM = 8. Since XYU has only the misspelling FOM of 3 with respect to XYV, XYV is equated to XYZ.

With respect to YXZ, XYZ has a misspelling FOM of 7, plus the null-RHS increment of 5, making a total FOM of 12: since XYU has a zero FOM for YXZ, CORC equates YXZ to XYZ.

With respect to YXW, neither XYZ nor XYU has a positive FOM; thus, YXW is not equated to a declared variable.

In the fourth group, GH1 was invariably used as a vector variable. Since it is a resem-

TABLE I. SAMPLE PROBLEMS IN POST-SCANNING SPELLING CORRECTIONS

Symbol	Type	Declared/ Defined?	LHS Usage	RHS Usage	Usage as Vector	Usage as Matrix	Total Usage
ABC	label	no					
ABCDE	label	yes					
ABD	label	yes					
BAC	label	yes					
AB≠	label	yes					
DE1	label	no					
DEI	label	yes					
XYZ	simple variable	yes	1	0	0	0	1
XYU	smp. var.	yes	2	1	0	0	3
XYV	smp. var.	no	1	1	0	0	2
YXZ	smp. var.	no	0	1	0	0	1
YXW	smp. var.	no	0	2	0	0	2
GHI	vector variable	yes	2	2	4	0	4
GH1	smp. var.	yes	2	2	4	0	4
GHJ	smp. var.	yes	1	2	2	0	3
GHK	smp. var.	no	2	2	0	0	4

blance misspelling of the declared vector variable GHI, it is equated to this variable and its status changed to a vector. GHJ was used 67 percent of the time as a vector variable; since it is a random misspelling of GHI, it is equated to the latter. GHK has a positive figure of merit with respect to each of the three preceding entries. However, GHK was never used as a vector variable. Since the GHJ and GHI have been set to vector status, GHK can no longer be equated to either of them; it thus remains a distinct, undeclared variable.

V. ERROR MONITORING DURING EXECUTION

CORC prefaces each compiled statement by a sequence of machine language instructions to monitor object-program flow. Additional "overhead" instructions for monitoring appear in four types of statements: labelled statements, statements containing subscripted variables, REPEAT statements, and READ statements. The monitoring effort has three objectives:

- (a) Prevent the object program from overwriting the CORC compiler/monitor or itself;

- (b) Continue the execution phase through untested code when the flow of the object program becomes confused (through misuse of REPEAT statements or incomplete GO TO, IF, and REPEAT statements);
- (c) Provide explicit diagnostic messages for each error detected at execution time, followed by an unconditional post-mortem dump of simple-variable values and other helpful data.<sup>6</sup> §

A. THE GENERAL MONITOR

- (1) CORC accumulates a count of all statements executed, *the statement count*. This count is printed in the post-mortem dump, together with the number of errors committed during the entire program and the total elapsed time for the program. The statement count has two minor functions: to aid debugging of

<sup>6</sup> Many debugging languages such as BUGTRAN (cf. 6) furnish trace and snapshot information *if requested by the programmer*. CORC furnishes such diagnostic information unconditionally; the overhead instructions cannot be suppressed after programs are debugged.



short programs in conjunction with the “label tallies” (see (3) below) and—looking towards future CORC research—to exhibit the different speeds of execution for various programs, e.g., with/without heavy subscript usage. The per-statement overhead of the statement count is 13.2 microseconds, comprising a single “tally” instruction.

- (2) Before executing each statement, its source-card serial number (converted to a binary integer) is loaded into an index register. Execution-phase messages resulting from this statement retrieve the serial number and print it as an introductory phrase to each message, e.g.,

IN STATEMENT 1234, THE PROGRAM IS STOPPED.

Each load-index instruction requires 3.3 microseconds. The percentage of execution time devoted to items (1) and (2) is usually less than 3 percent; see (5) below.

- (3) The execution of each labelled statement is tallied, by label. These tallies are printed in the post-mortem dump; they show the progress of the program, which branches were never taken, endless loops, etc. Each tally instruction requires 13.2 microseconds.
- (4) At each labelled statement, a two-position console switch is interrogated. In the normal position, the switch has no effect on program flow. If set, the switch causes the program to terminate at once, printing the message,

IN STATEMENT \_\_\_\_\_, THE PROGRAM IS MANUALLY INTERRUPTED,

followed by the usual post-mortem dump.

Thus, any endless loop can be manually interrupted without stopping the computer, although this is rarely necessary. (Cf. the subsequent section on *Terminations*.) The switch interrogation is required only at labelled statements, since endless loops must include at least one label. Each switch interro-

gation requires 7.2 microseconds. The percentage of execution time devoted to items (3) and (4) is usually less than 1 percent, as exhibited by the following analysis.

- (5) Assuming that 100,000 statements are executed per minute, an average statement requires some 600 microseconds. Since items (1) and (2) aggregate 16.5 microseconds per statement, the overhead for these items is 2.75 percent. Assuming that every fourth statement is labelled, items (3) and (4) are incurred once every 2400 microseconds on the average; since these times aggregate 20.4 microseconds, their overhead is approximately 0.8 percent.
- (6) No tracing features are offered in CORC. If a student requires more diagnostic data than is already furnished, he is encouraged to use WRITE and TITLE statements generously. However, he is also warned to print such data compactly:
- (a) If two consecutive pages print less than 30 percent of the 14,400 character spaces available (2 pages  $\times$  60 lines/page  $\times$  120 characters/line), CORC prints out the following message:

—TRY TO USE MORE EFFICIENT WRITE AND TITLE STATEMENTS AND AVOID WASTING SO MUCH PAPER—

- (b) A page-count limit is set for all normal programs; when this limit is reached, the program is terminated at once.

- (7) Each untranslatable source card has been replaced by a TITLE card during scanning, bearing the following message:

CARD NO. \_\_\_\_\_ NOT EXECUTED, SINCE UNTRANSLATABLE.

These messages remind the programmer of omitted actions during the execution phase.

**B. MONITORING ARITHMETIC ERRORS**

CORC uses conventional procedures for arithmetic overflow/underflow errors, but somewhat novel procedures for special-function argument errors. The machine traps of the computer detect overflow/underflow conditions, which are then interpreted into CORC messages:

- (1) IN STATEMENT \_\_\_\_\_, EXPONENT UNDERFLOW. (CORC zeros the accumulator and proceeds.)
- (2) IN STATEMENT \_\_\_\_\_, EXPONENT OVERFLOW. (CORC sets the accumulator to 1 rather than to some arbitrary, large number. This tends to avoid an immediate sequence of identical messages, allowing the execution phase to survive longer before termination from excessive errors.)
- (3) IN STATEMENT \_\_\_\_\_, DIVISION BY ZERO. ASSUME QUOTIENT OF 1.0.

For each special function error, CORC creates an acceptable argument and proceeds, instead of taking drastic action, e.g., immediate program termination, as many monitor systems do.

- (4) IN STATEMENT \_\_\_\_\_,  $\left\{ \begin{matrix} \$ \\ \text{EXP} \\ \text{SIN} \end{matrix} \right\}$  ARGUMENT TOO LARGE. THE RESULT IS SET TO 1.
- (5) IN STATEMENT \_\_\_\_\_, LN 0 YIELDS (or . . . LOG 0 YIELDS) 1.
- (6) IN STATEMENT \_\_\_\_\_,  $\left\{ \begin{matrix} \text{LN} \\ \text{LOG} \\ \text{SQRT} \end{matrix} \right\}$  OF NEGATIVE ARGUMENT. THE ABSOLUTE VALUE IS USED.
- (7) IN STATEMENT \_\_\_\_\_, ZERO TO NEGATIVE POWER—ASSUME 1.
- (8) IN STATEMENT \_\_\_\_\_, \$—NEGATIVE ARGUMENT. THE RESULT IS SET TO 1.

**C. TERMINATIONS**

Two abnormal terminations were discussed in the *General Monitor* section. Altogether,

there are five terminations, caused by the following events:

- (1) Console switch set.
- (2) Page count limit exceeded.
- (3) Time limit exceeded. Overflow of the real-time clock produces a machine trap which is intercepted by CORC. For each program, a time limit (ordinarily of sixty seconds) is set. (The tab cards separating the source decks can bear any non-standard page-count and time limits.||) When this time is exhausted, the program is terminated with the following message preceding the post-mortem dump:

IN STATEMENT \_\_\_\_\_, THE TIME IS EXHAUSTED.

Endless loops are terminated by this procedure, avoiding the necessity of operator intervention with the console switch.

- (4) Error count too high. After each program has been compiled, the total error count is interrogated. When it exceeds 100, then or thereafter, the program is terminated with the appropriate message.
- (5) Normal execution of STOP. The message  

IN STATEMENT \_\_\_\_\_, THE PROGRAM IS STOPPED

 identifies which STOP statement—possibly of several such statements—has been met. For all terminations, the post-mortem dump includes the following:
  - (a) The final values of all simple variables. Since arrays may comprise thousands of cells, CORC cannot afford paper or machine time to dump them too.
  - (b) The usage tallies for all labels.
  - (c) The first fifteen (or fewer) data card images.
  - (d) The error-count, statement-count, and elapsed-time figures.

|| Ordinarily the tab cards are blank. A special re-run drawer is used for programs which require unusual output volume or running time; the computing center inserts special tab cards with non-standard page-count and time limits before these decks.

#### D. MONITORING SUBSCRIPTED VARIABLES

One of CORC's most radical innovations is the universal monitoring of subscripts. CORC is attempting to trade execution efficiency for two other desiderata:

- (a) Protection of the in-core compiler/monitor against accidental overwriting by student programs.
- (b) Provision of complete diagnostics on all illegal subscripts: in which statements, for which variables, and the actual erroneous values of the subscripts.

CORC's excellent throughput speed has depended on infrequent destruction of the in-core compiler/monitor; in the author's opinion, subscript monitoring is CORC's most important protective feature.

Criterion (b)—full diagnostic information on subscript errors—is also of significance, since erroneous subscript usage comprises at least 30 percent of all execution-phase errors. Students quickly learn that these errors are among the easiest to commit—although they are spared the hardship of their detection and isolation.

Subscript usage is monitored as follows:

- (1) Each reference to a subscripted variable incurs a load-index instruction corresponding to the dictionary entry for this variable. If subsequent troubles arise in the subscripts, CORC can retrieve the name and other particulars of the variable by using this index register.
- (2) The subscript is an arithmetic expression, whose floating point value is transmitted in the machine accumulator to a closed subroutine for unfloating numbers.
- (3) The latter subroutine checks for a positive, integral subscript.
  - (a) 0 is changed to 1 with the following message:

IN STATEMENT \_\_\_\_\_, SUBSCRIPT FOR VARIABLE \_\_\_\_\_ IS 0. IT IS SET TO 1.

- (b) Negative numbers are also changed to 1:

IN STATEMENT \_\_\_\_\_, SUBSCRIPT FOR VARIABLE \_\_\_\_\_ IS NEGATIVE. IT IS SET TO 1.

- (c) If non-integral, the subscript is rounded to an integer. If the round-off error is less than  $10^{-9}$ , no error message is incurred; earlier calculations may have introduced small round-off errors into a theoretically exact subscript. If the round-off error exceeds  $10^{-9}$ , the following message appears:

IN STATEMENT \_\_\_\_\_, SUBSCRIPT FOR VARIABLE \_\_\_\_\_ IS NON-INTEGRAL. IT IS ROUNDED TO AN INTEGER.

- (d) After verifying (or changing to) a positive, integral subscript, the closed subroutine for unfloating subscripts returns control to the size test peculiar to this variable.
- (4) The subscript is tested for exceeding the appropriate dimension of the array variable. Thus, the first subscript of a matrix variable is tested against the declared maximum number of rows, and the second subscript is tested against the declared maximum number of columns; a vector subscript is tested against its declared maximum number of elements. An excessive value incurs one of the three following messages:

IN STATEMENT \_\_\_\_\_, \_\_\_\_\_ IS THE  $\left\{ \begin{array}{l} \text{FIRST} \\ \text{SECOND} \\ \text{VECTOR} \end{array} \right\}$  SUBSCRIPT FOR THE VARIABLE \_\_\_\_\_. SINCE IT IS EXCESSIVE, IT IS REPLACED BY THE VALUE \_\_\_\_\_.

The second blank in the message is filled with the current execution-phase value of the subscript. The third and fourth blanks are filled with the variable name and its maximum allowable subscript. This action serves to repair the erroneous subscript but hardly to correct it.

The overhead for each error-free usage of a subscript is 85 microseconds. With obvious waste of effort, this overhead is incurred six times for the statement:

```
LET A(I,J) = B(I,J) + C(I,J).
```

Future versions of CORC may treat such repeated usage of identical subscripts with more sophistication. However, one must remember that "A," "B," and "C" could have different maximum dimensions, in this example. A row subscript legal for "A" might be excessive for "B," etc. Also, in statements such as

```
LET A(I) = A(I + 1),
```

one must corroborate the legal size of "(I + 1)" as well as that of "I."

The per-program overhead of subscript monitoring varies between 0 percent and 90 percent of the execution time, as one might guess. An average overhead of 15 percent has been measured for a representative batch of programs.

## E. MONITORING REPEATED BLOCKS

- (1) Each repeatable block is legally used only as a closed subroutine. Hence, the exit instruction from the block—machine code generated by its END pseudo-statement—can be used to trap any illegal prior branch to an interior statement of the block. (One cannot enter a block by advancing sequentially through its BEGIN pseudo-statement. However, one can illegally branch to an interior statement of a repeatable block from a statement physically outside the block.) When the block is properly entered by a REPEAT statement, the address of the exit instruction is properly set; after the repetitions have been completed, a trap address is set into this exit instruction before the program advances beyond the REPEAT statement.

Thus, program flow can physically leave and re-enter a repeatable block in any complex pattern, as long as the block has been properly "opened" by a REPEAT statement and has not yet been

"closed" by completion of the repetitions. In this respect, CORC allows more complex branching than most compilers.

When the exit instruction traps an illegal prior entry, CORC prints the following message:

```
IN STATEMENT _____, AN IL-
LEGAL EXIT FROM BLOCK _____
HAS JUST BEEN DETECTED. IN
SOME PREVIOUS GO-TO STATE-
MENT, THE BLOCK WAS ILLE-
GALLY ENTERED. THE PROGRAM
CONTINUES AFTER THE END
STATEMENT OF THIS BLOCK.
```

- (2) To protect against various illegal usages of the bound variable in REPEAT-FOR statements, CORC pre-calculates the number of repetitions and conceals this count from the repeatable block; the count is fetched, decremented, and tested only by the REPEAT statement. This discussion is amplified in (d) below.

Consider the statement: REPEAT B FOR V = (E<sub>1</sub>, E<sub>2</sub>, E<sub>3</sub>):

- (a) If E<sub>1</sub> = E<sub>3</sub>, the block is executed once.
- (b) Otherwise, if E<sub>2</sub> is zero, CORC prints the following message: IN STATEMENT \_\_\_\_\_, IN REPEAT-FOR TRIPLE, SECOND ARGUMENT IS 0. THE REPEAT IS EXECUTED ONCE.
- (c) Otherwise, if (E<sub>3</sub> - E<sub>1</sub>)/E<sub>2</sub> is negative, CORC prints the following message:

```
IN STATEMENT _____, IN
REPEAT-FOR TRIPLE, SECOND
ARGUMENT HAS WRONG SIGN.
THE REPEAT IS EXECUTED
ONCE.
```

- (d) Otherwise, CORC uses the count  $\left[ \frac{E_3 - E_1}{E_2} \right]$  to determine the number of repetitions. This count is reduced by 1 for each iteration, irrespective of the subsequent values of "V," "E<sub>2</sub>," or "E<sub>3</sub>." Novice programmers often manipulate "V" inside repeatable blocks; CORC pre-

vents many potentially endless loops by ignoring this manipulation.

#### F. MONITORING DATA-CARD INPUT

The reading and checking of data cards was introduced in Section IV. In brief, a READ statement causes the following steps to occur.

- (1) A new card is read in; if it is of type 1, CORC assumes it to be the first card of the next source deck. Thereupon, the following messages appear:

THE INPUT DATA HAS BEEN EXHAUSTED. IN STATEMENT \_\_\_\_\_, CORC SUPPLIES A DATA CARD FOR THE VARIABLE \_\_\_\_\_ WITH VALUE 1.0.

Thus, CORC enters a value of 1 for the READ variable and proceeds with the program; subsequent READ statements incur only the second message above.

- (2) If the new card is neither type 1 nor type 4 (i.e., the correct type), CORC prints this message:

IN STATEMENT \_\_\_\_\_, THE CARD IS ASSUMED TO BE A DATA CARD.

- (3) If the new card is type 4—possibly as the result of (2) above—CORC checks the variable field against the variable name in the READ statement. If they disagree, CORC considers the name in the READ statement to be correct; the following message is printed:

IN STATEMENT \_\_\_\_\_, THE VARIABLE \_\_\_\_\_ WAS READ FROM THE CARD. THE VARIABLE IN THE READ STATEMENT WAS \_\_\_\_\_.

- (4) When the variable names have been reconciled CORC checks for none, one, or two subscripts on the card, as appropriate to the READ variable. Missing or erroneous subscripts incur the following message:

IN STATEMENT \_\_\_\_\_, THE SUBSCRIPT (\_\_\_\_\_,\_\_\_\_\_) WAS READ FROM THE CARD. THE

SUBSCRIPT IN THE READ STATEMENT WAS (\_\_\_\_\_,\_\_\_\_\_),

or

IN STATEMENT \_\_\_\_\_, THE SUBSCRIPT (\_\_\_\_\_) WAS READ FROM THE CARD. THE SUBSCRIPT IN THE READ STATEMENT WAS (\_\_\_\_\_).

In every case, CORC uses the value in the READ statement.

#### VI. CONCLUSIONS

##### A. EXPERIENCE IN PRACTICE

Throughout the 1962–63 academic year, CORC was in “pilot project” status; in 1963–64 CORC was established as the fundamental computing tool for undergraduate engineering courses at Cornell. In the spring semester of 1964, over 15,000 CORC programs were run, peaking at 2500 programs in one week.

The performance of CORC programmers far surpassed the preceding years’ performance by ALGOL programmers at Cornell in such respects as speed of language acquisition, average number of re-runs per program, and average completion time for classroom assignments.

Actual processing time can be evaluated from the following figures, which are rough estimates based on last year’s experience with CORC programs:

- (a) Average processing time (tape/tape configuration)—500 programs per hour.
- (b) Average machine-code execution rate—100,000 source-language statements per minute, for a random sample of twenty student programs.
- (c) Average compilation time for CORC programs—less than two seconds.
- (d) Turnaround time for programs—one day or less, with rare exceptions.

The author has automated the operation of the compiler/monitor to the following degree: only a random machine malfunction can cause the computer to halt. Since programming errors cannot produce object code that erroneously diverts control outside the CORC system, the

role of the machine operator is merely to mount input tape reels and remove output tapes: the computer console needs almost no attention.

A few error-detection procedures were altered during 1962–64, primarily to make diagnostic messages increasingly explicit. A new CORC manual was prepared for instructional use in 1963–64; this manual omitted any catalogue of errors, since the author expected that the compiler/monitor systems could describe the errors—and the corresponding remedial actions—in satisfactory detail.

CORC has imposed a modest load on the two computers at Cornell. The computing center is satisfied that neither FORTRAN nor ALGOL can lighten this load, which is rarely as much as two hours of CORC runs daily. (FORTRAN and ALGOL systems have greater capability but require more facility in programming. The class of problems for which CORC has been developed would not warrant the expenditure of time required to program in the advanced languages.) In the author's opinion, this small commitment of resources is well-justified by the educational value of the CORC project.

## B. POTENTIAL UTILITY OF CORC

The author feels that many universities and technical colleges can profitably utilize CORC for introductory instruction. The designers of CORC are convinced that a simple language is well suited for initial study; many Cornell students have already easily advanced to FORTRAN or ALGOL after mastering CORC.

With respect to the error-detection and error-correction features, CORC demonstrates the modest effort required to furnish intelligible messages and how little core memory and machine time are consumed. Many CORC error-monitoring procedures deserve consideration in future implementations of compiler languages: unconditional counts of statement labels (or statement numbers), source-program citations in diagnostic messages, and brief dumps following all program terminations. The monitoring of subscripts would not be burdensome if the latter were carried as integers—index registers are used in most current compiled codes. Ninety percent of the CORC subscript-

usage execution time is devoted to unfloating numbers, and only ten percent is devoted to testing these numbers for size.

## C. POTENTIAL IMPROVEMENTS IN CORC

Four areas for significant improvements in CORC are as follows:

- (1) Identification of integer-mode variables by their context. Index registers can then be used for arrays and loop counting as in FORTRAN.
- (2) A problem-grading mechanism. Each instructor can assign a scale of penalties for various errors. CORC will process his batch of student programs and assign the appropriate grades.
- (3) A permanent file for tabulating errors. Each time that CORC programs are run, an auxiliary output device—paper tape or punched cards—will record the serial number of each error committed. Periodically, these tapes or cards will be summarized. This data will furnish statistical estimates for the *a priori* occurrence probabilities of the errors.
- (4) Remote consoles. These are much discussed in current computer literature, and they hold unusual promise for high-volume university operation. Students would type in their programs from keyboards distributed around a campus covering hundreds of acres. Either these programs would interrupt a large computer programmed for real-time entry, or they would be stacked on tape/disk by a satellite computer. Perhaps results could be printed/typed at these remote stations by the satellite computer.

The author and his colleagues are well aware of shortcomings in the language. However, they intend to resist changes which increase the power of the syntax at the expense of linguistic simplicity. Changes on behalf of additional simplicity or clarity are willingly accepted. Continuing efforts will be made to improve the clarity and explicitness of the diagnostic messages, so that classroom instruction can be further integrated with output from the computer.

## VII. ACKNOWLEDGMENTS

The author is a former student of Professors Conway and Maxwell; he gratefully acknowledges their assistance to the error-correction project.

Other contributors were R. Bowen, J. Evans, C. Nugent, J. Rudan, and R. Sanderson.

## VIII. REFERENCES

1. CONWAY, R. W., and MAXWELL, W. L., "CORC: The Cornell Computing Language," *Comm. ACM*, 6, 317 (1963).
2. DAMEREAU, F. J., "A Technique for Computer Detection and Correction of Spelling Errors," *Comm. ACM*, 7, 171 (1964).
3. DAMEREAU, F. J., op. cit.
4. Ibid.
5. BLAIR, C. R., "A Program for Correcting Spelling Errors," *Inform. and Ctrl.*, March 1960, pp. 60-67.
6. FERGUSON, H. E., and BERNER, E., "Debugging Systems at the Source Language Level," *Comm. ACM*, 6, 430 (1963).

# THE COMPILATION OF NATURAL LANGUAGE TEXT INTO TEACHING MACHINE PROGRAMS\*

*Leonard Uhr*  
*University of Michigan*  
*Ann Arbor, Michigan*  
*Consultant, System Development Corporation*  
*Santa Monica, California*

Programmed instruction, via digital computers, must be made as painless as possible, both in the writing and the changing of programs, for the author of the programmed text. Otherwise we will only slowly accumulate a body of expensive programs that we will never succeed in testing adequately. It is crucial, given that we are investigating programmed instruction at all, that it become easy to write and rewrite the programs.

A great deal of research is needed as to the effectiveness of different types and sequences of items; therefore, programs must be flexible and easily changed. A large number of different programs will be needed, from many different content areas. These programs should be written by people whose competence is in these content areas. Such people cannot be expected to learn about computers, or about programming. Ideally, the problems of writing a program for computer teaching of a course in, for example, logic, French, botany, or computer programming should be no greater than the problems in writing a good book.

This paper describes a set of two programs that have been written to (1) allow someone to write a program in his content area without having to learn anything new other than what

appears to be an acceptable minimum of conventions, and then compile it (TMCOMPILE), and (2) interpret the compiled program, thus giving a running program that interacts with students (TEACH).

In effect, then, this is a compiler-interpreter for programs that are written in relatively unconstrained natural language (no matter which), so long as they are oriented toward the specific problem of programmed instruction, in that they conform to the format constraints described below. It is thus similar in spirit to problem-oriented compilers. Similar compilers have been coded at IBM (referred to in Maher<sup>3</sup>) and SDC (Estavan<sup>1</sup>). Despite what appear to be a significantly simpler logic and fewer conventions that must be learned, the present compiler, by means of its branching features, appears to handle a larger set of programs than IBM's, uses a somewhat simpler set of formatting rules, and offers the ability to make loose, partially ordered and/or unordered matches, to use synonyms, and to delete and insert questions conveniently. Estavan has written a program that assembles instructions telling a student where to look in a pre-assigned textbook. This program is restricted to multiple-choice questions.

---

\* The author would like to thank Ralph Gerard for bringing the magnitude of the practical need for such a compiler to his attention, William Uttal for discussions of some of the features that such a compiler should have, and Peter Reich for suggestions as to format.



*Description of the Program and the Inputs  
It will Accept*

If he wishes, the author of a programmed text might sit down at the keypunch or flexo-writer and compose in interaction with the computer. Or he might retire to his study and write down the set of information, questions, alternate possibilities for answers, and branches that he wants, and ask a keypunch operator to put these onto cards for compiling. In either of these two modes he must follow a few conventions, as described below. Or, if he insists upon his freedom, he might simply be asked to write his text in any way he desired, subject only to the restriction that it follow the very general format of containing only: (1) statements giving information to the student, and (2) questions about this information, either (a) multiple-choice, (b) true-false, or (c) correctly answerable in a concise way, with the various acceptable crucial parts of answers listed by the author right after the question and acceptable synonyms listed in a synonym dictionary. In addition, for each alternative answer (or set of alternatives), the author should say what question or statement of fact the program should ask of the student next. (Or, alternatively, if the author does not bother to specify this, the program will simply go to the next item—the next statement or question—according to the order that the author has given them.) A text written in such a way could easily be formatted by the keypunch operator who punched it onto cards.

In general, then, the type of text that the author must write must be a set of strings which are either statements (of information) or questions. The questions must be followed by the alternate possible answers, and each set of alternate possible answers must be followed by an explicit or implied branch to another string in the text. Figures 1A and 2A give examples of such texts.

If the author is willing to go to a little bit of trouble, he will produce the texts of Figures 1A and 2A in a form that will be compiled directly. Figures 1B and 2B show what these texts would look like then.

If the author makes use of the computer as he writes, he can delete strings that he would

like to change, by means of an instruction to "erase string *i*," and then, if he wishes, write in the new version of string *i*. He can also ask the program to begin teaching him (or others), to collect data on successes and failures, and to give him a feeling of the program from the student's point of view.

*Rules for Format*

- A. The peculiarities of this language that the user must learn are as follows:
  1. A new item must be identified by \*NAME.
  2. Items are composed of elements, and all elements are bounded by slashes (/).
  3. The following things are elements: (a) the entire statement giving information or advice, (b) the entire question, (c) each alternative possible answer to a question, (d) the branch to the next string to be presented to the subject.
  4. The branch element must start with an asterisk (\*).
- B. If he so desires, the author can gain a good bit of additional flexibility by using the following additional features of the language:
  5. The NAME is optional: if none is given, the program names this string with the integer one greater than the last integer name given. The name can either be an integer (in which case care must be taken that it is never automatically assigned by the program) or a string of alphanumeric character.
  6. An "otherwise" branch (\*\*) for the entire question is optional, and goes at the end of the answer portion of a question.
  7. Partial matches between a student's answer and an acceptable answer will be accepted if they fulfill the following criteria: (a) if a word in the answer is listed in a synonym dictionary that has been read into the program as equivalent to a word that the student uses, (b) if the correct answer is a connected substring of the student's answer, (c) if a correct answer is specified as a list of substrings separated by commas and pe-

## A. In Need of Pre-editing.

TO TELL WHETHER AN OBTAINED DIFFERENCE IS SIGNIFICANT, YOU MUST KNOW WHETHER IT IS LARGER THAN MIGHT ARISE FROM SAMPLING VARIABILITY. SAMPLING VARIABILITY IS DUE TO ACCIDENTAL OR CHANCE FACTORS THAT AFFECT THE SELECTION OF OBSERVATIONS INCLUDED IN THE SAMPLE. THESE CHANCE FACTORS OBEY THE LAWS OF PROBABILITY; FROM THESE LAWS YOU CAN CALCULATE HOW BIG A DIFFERENCE MIGHT BE EXPECTED BETWEEN TWO SAMPLES DRAWN FROM THE SAME POPULATION. THE LAWS OF PROBABILITY APPLY ONLY TO SAMPLES THAT CAN BE SHOWN TO BE RANDOM SAMPLES.

A RANDOM SAMPLE MUST BE SELECTED IN A WAY THAT GIVES EVERY OBSERVATION IN THE - BEING SAMPLED AN EQUAL CHANCE OF BEING INCLUDED.

ANSWER: POPULATION.

WHEN THE NAMES OF THE STUDENTS IN A COLLEGE ARE WRITTEN ON IDENTICAL SLIPS AND ARE DRAWN OUT OF A HAT BY A BLINDFOLDED PERSON, THE SAMPLE SO DRAWN IS A - SAMPLE BECAUSE EACH MEMBER OF THE POPULATION WOULD HAVE AN - - OF BEING INCLUDED. ANSWER: RANDOM...EQUAL CHANCE.

A SAMPLE THAT IS NOT RANDOM IS BIASED. IF SOME OF THE STUDENTS' NAMES WERE NOT IN THE HAT, THE SAMPLE DRAWN WOULD BE - .

ANSWER: BIASED.

## B. Prepared for Automatic Compilation.

\*/TO TELL WHETHER AN OBTAINED DIFFERENCE IS SIGNIFICANT, YOU MUST KNOW WHETHER IT IS LARGER THAN MIGHT ARISE FROM SAMPLING VARIABILITY. SAMPLING VARIABILITY IS DUE TO ACCIDENTAL OR CHANCE FACTORS THAT AFFECT THE SELECTION OF OBSERVATIONS INCLUDED IN THE SAMPLE. THESE CHANCE FACTORS OBEY THE LAWS OF PROBABILITY; FROM THESE LAWS YOU CAN CALCULATE HOW BIG A DIFFERENCE MIGHT BE EXPECTED BETWEEN TWO SAMPLES DRAWN FROM THE SAME POPULATION. THE LAWS OF PROBABILITY APPLY ONLY TO SAMPLES THAT CAN BE SHOWN TO BE RANDOM SAMPLES/

\*/A RANDOM SAMPLE MUST BE SELECTED IN A WAY THAT GIVES EVERY OBSERVATION IN THE - BEING SAMPLED AN EQUAL CHANCE OF BEING INCLUDED/POPULATION/

\*/WHEN THE NAMES OF THE STUDENTS IN A COLLEGE ARE WRITTEN ON IDENTICAL SLIPS AND ARE DRAWN OUT OF A HAT BY A BLINDFOLDED PERSON, THE SAMPLE SO DRAWN IS A - SAMPLE BECAUSE EACH MEMBER OF THE POPULATION WOULD HAVE AN - - OF BEING INCLUDED/RANDOM.EQUAL CHANCE.=1/

\*/A SAMPLE THAT IS NOT RANDOM IS BIASED. IF SOME OF THE STUDENTS' NAMES WERE NOT IN THE HAT, THE SAMPLE DRAWN WOULD BE - /BIASED/

Figure 1. A Sequence Typical of Those Found in Programmed Instruction Texts.

Figure 1a. In Need of Pre-editing.

Figure 1b. Prepared for Automatic Compilation.

A. In Need of Pre-editing.

JOHN LIKES MARY BROWN.

WHO DOES JOHN LIKE? MARY BROWN.B. MARY.A.

A. MARY WHO? BROWN.B; OTHERWISE TO 1ST

B. BUT MARY LIKES PHIL AND PHIL LIKES BETTY.

DOES MARY LIKE BETTY? YES OR NO.C. DON'T KNOW.D.

C. YOU REALLY CAN'T KNOW FROM WHAT YOU'VE BEEN TOLD. IF ONE PERSON LIKES A SECOND PERSON WHO LIKES A THIRD, IT'S NOT CERTAIN THAT THE FIRST PERSON LIKES THE THIRD.

D. JOHN LIKES BETTY TOO, ALONG WITH JANE, AND CAROL.

WHO DOES JOHN LIKE? BETTY, MARY, JANE, OR CAROL.E. GIRLS, OR WOMEN.F.

E. GENERALIZE. WHAT DO BETTY, MARY, JANE AND CAROL HAVE IN COMMON?

JOHN LIKES - . GIRLS, OR WOMEN.F. WHO JOHN LIKES.G.

F. RIGHT. BUT NOT NECESSARILY ALL. DO YOU THINK JOHN LIKES MOST GIRLS?

YES, OR MAYBE.H. NO, OR DON'T KNOW, OR NOT ENOUGH INFORMATION.I.

G. IT DOESN'T ADD MUCH TO SAY "JOHN LIKES THAT WHICH JOHN LIKES." SUCH A STATEMENT IS CALLED A TAUTOLOGY -- THERE'S NO POINT IN SAYING THE SECOND HALF ONCE YOU'VE SAID THE FIRST HALF.I.

H. NO. THIS IS A VERY FALLIBLE AND UNLIKELY SORT OF INFERENCE TO DRAW.

FOR INSTANCE, JOHN CERTAINLY DOESN'T EVEN KNOW MOST GIRLS. GENERALIZATIONS OF THIS SORT ARE RISKY AT BEST, BUT AT THE LEAST YOU MUST KNOW MUCH MORE ABOUT THE TOTAL GROUP -- GIRLS -- AND ITS RELATION TO JOHN AND HOW THE PARTICULAR EXAMPLES GIVEN WERE CHOSEN.

I. IT SO HAPPENS THAT JOHN DOES LIKE MOST OF THE GIRLS THAT HE KNOWS. MOST MEN AND BOYS DO. BUT THERE ARE ALWAYS EXCEPTIONS. FOR EXAMPLE, JOHN DOESN'T LIKE ALICE. ON THE OTHER HAND, HE USUALLY LIKES THE GIRLS THAT PHIL LIKES.

IS IT LIKELY THAT JOHN LIKES BETTY? YES.J. OTHERWISE.K.

J. RIGHT. SINCE PHIL LIKES BETTY AND JOHN TENDS TO LIKE GIRLS AND TENDS TO LIKE GIRLS THAT PHIL LIKES.L.

K. THERE IS SOME REASON TO THINK YES, SINCE PHIL LIKES BETTY.

L. READ PAGES 7-13 OF THE TEXT.

Figure 2. A Contrived Example Exhibiting Some Features of the Program.

\*/JOHN LIKES MARY BROWN/

\*/WHO DOES JOHN LIKE/MARY BROWN/\*B/MARY/\*A/\*\*1/

\*A/MARY WHO/BROWN/\*B/\*\*1/

\*B/BUT MARY LIKES PHIL AND PHIL LIKES BETTY/

\*/DOES MARY LIKE BETTY/YES/NO/\*C/N.T.KNOW.=2/\*D/

\*C/YOU REALLY CAN'T KNOW FROM WHAT YOU'VE BEEN TOLD. IF ONE PERSON LIKES A SECOND PERSON WHO LIKES A THIRD, IT'S NOT CERTAIN THAT THE FIRST PERSON LIKES THE THIRD/

\*D/JOHN LIKES BETTY TOO, ALONG WITH JANE, AND CAROL/

\*/WHO DOES JOHN LIKE/BETTY,MARY,JANE,CAROL,=0/\*E/GIRLS,WOMEN,=0/\*F/

\*E/GENERALIZE. WHAT DO BETTY, MARY, JANE, AND CAROL HAVE IN COMMON/

\*/JOHN LIKES -/GIRLS,WOMEN,=0/\*F/WHO.JOHN.LIKES.=2/\*G/\*\*1/

\*F/RIGHT. BUT NOT NECESSARILY ALL. DO YOU THINK JOHN LIKES MOST GIRLS/ YES/MAYBE/\*H/NO/DON'T KNOW/NOT ENOUGH INFORMATION/\*I/

\*G/IT DOESN'T ADD MUCH TO SAY "JOHN LIKES THAT WHICH JOHN LIKES." SUCH A STATEMENT IS CALLED A TAUTOLOGY -- THERE'S NO POINT IN SAYING THE SECOND HALF ONCE YOU'VE SAID THE FIRST HALF/\*I/

\*H/NO. THIS IS A VERY FALLIBLE AND UNLIKELY SORT OF INFERENCE TO DRAW. FOR INSTANCE, JOHN CERTAINLY DOESN'T EVEN KNOW MOST GIRLS. GENERALIZATIONS OF THIS SORT ARE RISKY AT BEST, BUT AT THE LEAST YOU MUST KNOW MUCH MORE ABOUT THE TOTAL GROUP -- GIRLS -- AND ITS RELATION TO JOHN AND HOW THE PARTICULAR EXAMPLES GIVEN WERE CHOSEN/

\*I/IT SO HAPPENS THAT JOHN DOES LIKE MOST OF THE GIRLS THAT HE KNOWS. MOST MEN AND BOYS DO. BUT THERE ARE ALWAYS EXCEPTIONS. FOR EXAMPLE, JOHN DOESN'T LIKE ALICE. ON THE OTHER HAND, HE USUALLY LIKES THE GIRLS THAT PHIL LIKES/

(Continued)

```

*/IS IT LIKELY THAT JOHN LIKES BETTY/YES/*J/**K/
*/J/RIGHT. SINCE PHIL LIKES BETTY AND JOHN TENDS TO LIKE GIRLS AND TENDS
TO LIKE GIRLS THAT PHIL LIKES/*L/
*/K/THERE IS SOME REASON TO THINK YES, SINCE PHIL LIKES BETTY/
*/L/READ PAGES 7-13 OF THE TEXT/

```

Figure 2b. Prepared for Automatic Compilation.

riods, and ending with a number, e.g., /XX,XX,XX.XX.=N/, the program will look for an *unordered* match of the substrings terminating in commas, and an *ordered* match (starting from the first ordered substring) of the substrings terminating in periods. It will count the number of such matches it gets, and, if this is greater than N, it will accept the student's answer.

To summarize briefly, a new item must start with an \*. Its elements (statement of fact, question, alternate answer, branch) must be bounded by /. An item with more than one element is treated as a question. An item can have an optional numerical or symbolic name. A branch for any set of alternate answers can be specified by \*, and an "otherwise" branch by \*\*.

The following is a short example:

```

*1/JOHN LIKES JANE,SALLY,JO,AND
BETTY./
*A|WHO DOES JOHN LIKE/JANE,SAL-
LY,BETTY,JO,=0/*B/GIRLS/*C/MA,SAN-
TA,MO,=0/**1/
*/DON'T BE IRRELEVANT/*A/
*/C/BE MORE SPECIFIC/*A/
*/B/BILL LIKES MARY,ANN,JANE,
RUTH,SALLY,AND JO./
*/NAME TWO GIRLS BOTH JOHN AND
BILL LIKE./JA,SA,JO,=1/**B/

```

## DISCUSSION

### *Optional Modes of Operation*

The program will automatically refrain from asking a question that has previously been answered correctly with a frequency above a tolerance parameter, *t*, or if the student, at the time he answered the question *correctly*, also said "\*EASY\*."

Several other features are optional, dependent upon whether special flags have been raised for the particular run. Thus, when desired, the program will print out any or all of the following in response to a student's answer when a set of answers is required: "YOU ARE RIGHT TO SAY—" followed by the correct elements of the student's answer, "YOU ARE WRONG TO SAY—" followed by the incorrect elements of the student's answer, and "YOU SHOULD HAVE SAID—" followed by those elements that the student left unsaid.

The compiler and interpreter programs were coded in SNOBOL (Farber<sup>2</sup>) for the IBM 7090. As presently coded, the interpreter program handles only one student, accumulating the frequency of his success and failure on each question. If many consoles were used, each console would have a name and the different students would time-share the program. It seemed futile to add this to the present program (although it would be trivial to do so), since SNOBOL has no provision for reading in from on-line sources.

Figure 3 gives examples of a compiled program and its interactions with a student.

*Some Examples of Types of Material That Can Be Handled*

The person writing the text to be compiled has a great deal of latitude in formatting his material. The present set of programs will handle a wide variety of question and statement formats, including multiple-choice, true-false, fill-ins (either connected or disconnected, ordered or unordered), short answer questions, and essays. The limitations of the short answer type of question lie in the ability of the people who specify the alternate acceptable answers and the synonym dictionary. The key parts to the answer might be very loosely stated. A statement might impart information, or make a comment about the student's performance, or it might command the student to read a certain section of a certain book or perform a certain series of exercises. A branch might be to a question that underlies, forms a part of, or supplements the question missed (or got). Separate branches can be established for different answers with different implications and for different partial answers.

With such programs the distinction between teaching, testing and controlling the student becomes an arbitrary one. Thus a compiled program might be used to train the student in some content area, to simultaneously train and test, to give a final examination, or to run an experiment that explored the student's abilities under some specified conditions and treatments.

*Possible Extensions to the Present Program*

The program that has been coded is a simple first attempt toward what might be done, such as the following.

A. Rather than branch to a single string, the strings could belong to one or more classes, and the branch could then be to a class that contained several strings. For each particular execution of the branch, a random choice could be made; or, better, this choice could be a function of the difficulty of the different members of the class.

B. Frequencies of successes and failures could be collected for (1) each student, (2) all

students, (3) given types of students (e.g., high IQ, impulsive). Then the choice of the particular branch could be a function of the appropriate individual and/or group information as to what is likely to benefit this student.

C. The decision as to what group to *put* a student into could be made by the program, if it compared the patterns of successes and failures across students, and put students with similar patterns into the same group (e.g., by Kendall's tau).

D. The decision as to what string to *branch to* after each string could be made by the program, by some rule such as the following: (1) branch to a string whose success-failure frequencies are similar to this string's, (2) branch to a string whose answer is a substring of the answer to this failed string, (3) branch to a string whose answer contains this correctly answered string.

E. Weights of specific strings can be not merely functions of success-failure of themselves, but also functions of success-failure of other strings that are related to them by, for example, (1) equivalence-relatedness as specified by the author in a simple equivalence dictionary, (2) connectedness in the sense of the graph formed by the branches cycling through the strings.

F. At least simple methods could be programmed for taking an ordinary book, breaking it up into a set of statements, interspersing questions *composed by the program*, and then, by pretesting with human experimental students, winnowing the questions down to a good set (e.g., (1) non-redundant, (2) suitably difficult, (3) reliable, (4) valid).

G. Answers could be recognized by additional partial and loose matches that would allow for a wider variety of alternate forms, for example, misspelled words, than can be recognized at present.

H. The program could systematically collect alternate answers (e.g., from students that it judges to be pretty good) and occasionally ask *its* teacher whether these would in fact be acceptable alternates. It would then add these to its memory. It could similarly augment its synonym dictionary.

```

*/SUPPOSE WE HAVE TWO SENTENCES, 'A' AND 'B'. THEN THE SENTENCE
'(A)V(B)' IS CALLED THE DISJUNCTION (OR ALTERNATION, OR LOGICAL
SUM) OF THE SENTENCES 'A' AND 'B'./
*/ A SENTENCE SUCH AS '(C)V(D)' IS CALLED THE LOGICAL SUM, OR
ALTERNATION, OR ----- ./DISJUNCTION/*/**1/
*A/ WE AGREE THAT THE DISJUNCTION '(A)V(B)' IS TRUE IF AND ONLY
IF AT LEAST ONE OF THE TWO SENTENCES 'A' AND 'B' IS TRUE, I.E.,
IF EITHER 'A' IS TRUE, OR 'B' IS TRUE, OR BOTH OF THEM ARE TRUE./
*B/ IF IT IS NOT KNOWN WHETHER 'A' IS TRUE, CAN '(A)V(B)' BE
TRUE/Y.E.S.=1/*/**A/
*C/ IF 'A' IS FALSE, CAN '(A)V(B)' BE TRUE/YES/*/**A/
*D/ IF 'A' AND 'B' ARE FALSE, CAN '(A)V(B)V(C)' BE
TRUE/(CA)N.T.SA(Y).=2/*F/**G/
*E/ YES, IN FACT IT CAN. BUT THIS DOES NOT YET FOLLOW FROM WHAT
YOU HAVE BEEN TOLD./
*/ YOU ARE RIGHT IN SAYING THAT YOU DONT KNOW IF YOU MEAN
THAT THIS IS NOT YET DECIDED./
*/ IN FACT IT CAN. BUT THIS HAS NOT YET BEEN STATED EXPLICITLY IN
THE SYSTEM BEING DEVELOPED FOR YOU./
*/ THE SIGN 'V' OF DISJUNCTION CORRESPONDS WITH FAIR EXACTNESS
TO THE ENGLISH WORD 'OR' IN THOSE CASES WHERE 'OR' STANDS BETWEEN
TWO SENTENCES AND IS USED (AS IT MOST FREQUENTLY IS) IN THE
NON-EXCLUSIVE SENSE./
*/ WITH WHAT COMMON ENGLISH WORD DOES THE SIGN 'V' CORRESPOND
MOST CLOSELY/OR/*/**H/
*/GOOD. 'OR' IS CORRECT. CONGRATULATIONS ON FINISHING
THIS LESSON./
*

```

Figure 3. A Short Example of a Computer Run That Demonstrates Some Simple Uses of the Partial Match Features.

Figure 3a. Listing of the Program to be Compiled.

INTERACTIONS WITH STUDENTS FOLLOW.

INFORMATION- SUPPOSE WE HAVE TWO SENTENCES, 'A' AND 'B'. '(A)V(B)' IS CALLED THE DISJUNCTION (OR ALTERNATION, OR LOGICAL SUM) OF THE SENTENCES 'A' AND 'B'./

QUESTION- A SENTENCE SUCH AS '(C)V(D)' IS CALLED THE LOGICAL SUM, OR ALTERNATION, OR ----- .  
STUDENT ANSWERED- 'SUM'  
NO, WRONG.

INFORMATION- SUPPOSE WE HAVE TWO SENTENCES, 'A' AND 'B'. '(A)V(B)' IS CALLED THE DISJUNCTION (OR ALTERNATION, OR LOGICAL SUM) OF THE SENTENCES 'A' AND 'B'./

QUESTION- A SENTENCE SUCH AS '(C)V(D)' IS CALLED THE LOGICAL SUM, OR ALTERNATION, OR ----- .  
STUDENT ANSWERED- 'DISJUNCTION'  
RIGHT. A GOOD ANSWER IS-- DISJUNCTION

INFORMATION- WE AGREE THAT THE DISJUNCTION '(A)V(B)' IS TRUE IF AND ONLY IF AT LEAST ONE OF THE TWO SENTENCES 'A' AND 'B' IS TRUE, I.E., IF EITHER 'A' IS TRUE, OR 'B' IS TRUE, OR BOTH OF THEM ARE TRUE./

QUESTION- IF IT IS NOT KNOWN WHETHER 'A' IS TRUE, CAN '(A)V(B)' BE TRUE  
STUDENT ANSWERED- 'NO'  
NO, WRONG.

INFORMATION- WE AGREE THAT THE DISJUNCTION '(A)V(B)' IS TRUE IF AND ONLY IF AT LEAST ONE OF THE TWO SENTENCES 'A' AND 'B' IS TRUE, I.E., IF EITHER 'A' IS TRUE, OR 'B' IS TRUE, OR BOTH OF THEM ARE TRUE./

QUESTION- IF IT IS NOT KNOWN WHETHER 'A' IS TRUE, CAN '(A)V(B)' BE TRUE  
STUDENT ANSWERED- 'YAS'  
RIGHT. A GOOD ANSWER IS-- Y E S

QUESTION- IF 'A' IS FALSE, CAN '(A)V(B)' BE TRUE  
STUDENT ANSWERED- 'WHY SHOULDN'T I SAY IT IN FRENCH- MAIS OUI, CERTAINEMENT'  
RIGHT. A GOOD ANSWER IS-- YES OUI

QUESTION- IF 'A' AND 'B' ARE FALSE, CAN '(A)V(B)V(C)' BE TRUE  
STUDENT ANSWERED- 'THAT CAN'T REALLY BE SAID'  
RIGHT. A GOOD ANSWER IS-- CAN T SAY

INFORMATION- YOU ARE RIGHT IN SAYING THAT YOU DONT KNOW IF YOU MEAN THAT THIS IS NOT YET DECIDED./

INFORMATION- IN FACT IT CAN. BUT THIS HAS NOT YET BEEN STATED EXPLICITLY IN THE SYSTEM BEING DEVELOPED FOR YOU./

INFORMATION- THE SIGN 'V' OF DISJUNCTION CORRESPONDS WITH FAIR EXACTNESS TO THE ENGLISH WORD 'OR' IN THOSE CASES WHERE 'OR' STANDS BETWEEN TWO SENTENCES AND IS USED (AS IT MOST FREQUENTLY IS) IN THE NON-EXCLUSIVE SENSE./

QUESTION- WITH WHAT COMMON ENGLISH WORD DOES THE SIGN 'V' CORRESPOND MOST CLOSELY  
STUDENT ANSWERED- 'AND'  
NO, WRONG.

INFORMATION- THE SIGN 'V' OF DISJUNCTION CORRESPONDS WITH FAIR EXACTNESS TO THE ENGLISH WORD 'OR' IN THOSE CASES WHERE 'OR' STANDS BETWEEN TWO SENTENCES AND IS USED (AS IT MOST FREQUENTLY IS) IN THE NON-EXCLUSIVE SENSE./

QUESTION- WITH WHAT COMMON ENGLISH WORD DOES THE SIGN 'V' CORRESPOND MOST CLOSELY  
STUDENT ANSWERED- 'NON-EXCLUSIVE 'OR''  
RIGHT. A GOOD ANSWER IS-- OR

INFORMATION- GOOD. 'OR' IS CORRECT. CONGRATULATIONS ON FINISHING THIS LESSON./

Figure 3b. Printout of Interactions with a Simulated Student.

I. It could further try to boil down sets of equivalent alternate answers, by finding things in common among them, composing a summarizing statement, and asking its teacher whether this new statement is equivalent to all the specific alternates it is presently storing. It could then substitute this new statement for the alternates that in fact were equivalent, and now look only for this common element in students' future answers.

J. It could have various methods for computing branches when appropriate to the problem domain; for example, (1) using a transform dictionary to analyze mistakes in logic or

arithmetic, (2) using similarity between substrings to analyze types of mistakes in spelling.

K. The program could itself compute the correct answer, rather than having this answer stored in memory. It might then also do such things as check the sequence of a student's answer (which it would get simply by commanding the student "GIVE YOUR ANSWER STEP BY STEP") and try to analyze at what point the student went astray. It could then generate a new question either on the basis of such an analysis or as a function of the student's present level of competence.



L. Some simplifications in the basic formatting rules could be implemented with relatively little trouble. For example, the program might accept several alternative identifications for questions; e.g., "\*" could be replaced by "\*Q" or "\*QUESTION" or "QUESTION"; "." could be replaced by "THEN"; "," could be replaced by "AND"; "/" in the answer section could be replaced by "OR"; the "\*" that marks the branch by "GOTO"; the "-" that designates erase by "ERASE". Experiments might be run to see which form is preferable. If, as seems likely, the presently implemented form is somewhat harder to learn at first, but slightly faster to use once learned (if only because fewer symbols need be typed), novices could be trained on the form that looks

more English-like and then given the option of using the shorter, more cryptic symbols.

#### REFERENCES

1. ESTAVAN, D. P. "Coding for the class lesson assembler." FN-5633. Santa Monica, Calif.: System Development Corp., 1961.
2. FARBER, D. J., GRISWOLD, R. E., and POLONSKY, I. P. "SNOBOL, a string manipulation language." J. Assoc. Comp. Machinery, Vol. 11, No. 1, Jan. 1964, 21-30.
3. MAHER, A. "Computer-based instruction: Introduction to the IBM research project." RC-1114. Yorktown Heights, N.Y.: IBM, 1964.

# METHOD OF CONTROL FOR RE-ENTRANT PROGRAMS

*Gerald P. Bergin  
Programming Systems  
International Business Machines Corporation  
New York, New York*

## INTRODUCTION

The use of multiprogramming and multiprocessing raises a question as to the number of copies of a routine needed in memory for multiple concurrent use. In the case where two or more scientific programs are in core at the same time, each needing the use of a SINE routine, a private copy can be provided for each program's own use, or one copy can be loaded for all to use. A message processing program that services multiple terminals can run into a situation where message A interrupts the processing of message B and because of priority considerations, message A must be processed immediately by the program. Again, the question of how many copies of the program are required in core occurs. Finally, a multiprocessing configuration with two or more computers sharing a common core memory may each be using the FORTRAN compiler. Each computer could have its own copy of the compiler or a single copy of the compiler could be executed by all computers concurrently. Intuitively, the provision of one copy of the routine or program appears more elegant.

Assuming the use of only one copy of each routine, the possibility that a commonly used routine may not run to completion before being entered again must now be considered. A routine which permits unlimited multiple entrances and executions before prior executions are complete is called a re-entrant routine. This paper describes a method of controlling these routines and sets forth conventions that must be fol-

lowed to produce a routine that satisfies the requirements of re-entrability.

The terms used in this paper are defined to eliminate possible misinterpretation.

routine—an ordered set of computer instructions which is entered by an explicit call  
program—a set of routines and associated data areas

context—the information which a routine needs to perform its functions

instance—the execution of a routine with a particular context

read-only routine—a sequence of machine instructions which is not self modifiable or modifiable by others

re-entrant routine — a read-only routine which accepts and uses the context associated with an instance of a routine, such that multiple entrances and executions can occur before prior executions are completed

subexecution—an instance of a re-entrant routine

task—a set of one or more routines which define a unit of work, and which can compete independently for computer time

job—a collection of tasks organized and submitted by a user under a single accounting number

LIFO—Abbreviation for Last In, First Out. This pertains to the retrieving of data in the reverse order in which it was stored. Also called a push-down, pop-up list

SCL—Abbreviation for Single Cell available space List

ATQ—Abbreviation for Active Task Queue

TCL—Abbreviation for Task Control List

SCB—Abbreviation for Subexecution Control Block

BAL—Abbreviation for Block Available space List

## RE-ENTRANT PROBLEMS

The biggest problem a re-entrant routine poses is that of referencing proper context. The routine can be made to conform to a well-defined set of conventions for its references to input, output, and working storage—this solves only part of the problem. The remainder must be resolved through the use of a monitor capable of associating context with each instance of the re-entrant routine, and of accepting responsibility for providing context reference during re-entrant executions.

The amount of control information which a monitor must create and maintain is a function of:

1. The number of unfinished instances of each re-entrant program
2. The number of unfinished sub-executions (or current levels down) for each program instance
3. The number of context pointers to data for each unfinished subexecution.

In addition, each routine requires working storage and data areas associated with a given instance. To pre-allocate all the space required for some maximum activity seems unreasonable in a dynamic environment. When activity is minimal, a large number of cells would be unusable for other purposes, and any change in the size of data blocks would require re-assembly of the system.

Dynamic space allocation will circumvent some of these problems; space can be allocated as needed. When the space for a subexecution is no longer required, it is returned to available space and can be used by other subexecutions.

To provide dynamic space allocation, both a single cell and block allocation scheme were considered essential. A small block of space is pre-linked and constitutes the Single Cell

available space List (SCL). This space is available for use by the monitor only. The Block Available space List (BAL) permits blocks of space of variable size to be allocated for both program and monitor storage needs. A description of the Space Allocation scheme is contained in Appendix A.

## RE-ENTRANT CONTROL

### *Functions*

The monitor functions discussed are not intended to be all inclusive even for re-entrant routines. Functions such as I/O and interrupt control are virtually ignored since they are of little concern in this paper.

The monitor functions which are of importance for re-entrant control include:

1. Obtaining and returning single cells and blocks of cells needed for control information
2. Determining priority of tasks and task queuing
3. Creating and terminating tasks
4. Maintaining context for each unfinished instance
5. Maintaining the data structures which reflect the activity of the re-entrant routines
6. Handling all inter-routine and intra-routine communication

### *Structure*

To perform these functions, the monitor must have control information organized in some manner. The following data structures are, therefore, the basis of achieving the required monitor control.

### *Job Description Block (JDB)*

Pertinent information about each job is contained in a set of contiguous locations called a Job Description Block. One JDB is created for each job. These blocks are the source of all activity to be done in regard to job processing, especially the sequence of tasks to be accomplished within each job. The set of all JDBs need not reside permanently in core although information pertaining to some tasks may be used frequently enough to dictate its presence.

The major concern this paper has with JDBs is that they exist.

#### *Active Task Queue (ATQ)*

The Active Task Queue is a list of the tasks which are in some phase of execution in the computer. There is a scheduling procedure applied to this list to determine the next task to be activated or reactivated when interrupts occur or when a subexecution relinquishes control.

The ATQ is a simple list structure composed of cells obtained from the SCL. The monitor adds or inserts an entry to the list when a new task is to become a candidate for processing in the multiprogram environment. An entry is deleted from the queue when a task is terminated, and the cell is returned to the SCL.

Each entry in the ATQ contains status information, task identification, priority number, pointer to the associated task control list, and a link to the next entry in the ATQ.

#### *Task Control List (TCL)*

Associated with each ATQ entry is a Task Control List. This list is used to establish and associate context for each level of subexecution within a task. When a task is added to the ATQ, a TCL is created for it. The first entry contains the name of the associated JDB and the second entry points to a Subexecution Control Block (SCB). This SCB contains the pointers to the context needed for the execution of the main control routine of the task. When the execution of the control routine is initiated, each level of descent into nested subexecutions causes an entry to be added to the TCL which points to the associated SCB. When a subexecution terminates, returning to the prior level of execution, its entry is removed from the TCL. It should be noted that all transfer of control to and from subexecution is through the monitor.

The TCL is a push-down, pop-up (LIFO) list with entry to the list through a header cell—the cell containing the name of the JDB. The header cell and push down cells are obtained from (and later returned to) the SCL. New entries to the LIFO list are added to the top of the list with prior entries pushed down. Termination of a subexecution results in the LIFO

list being popped up and the cell returned to the SCL (also returning the SCB space). When the control section terminates, its entry in the TCL, the TCL header, and the entry in the ATQ are deleted thereby terminating the task. The job description may get updated at this point.

The first entry of the TCL contains a pointer to and the name of the JDB, and a link to the top of the LIFO list. Each entry on the LIFO list points to a SCB, links to the previous entry on the list, and contains the name of the subexecution.

#### *Subexecution Control Block (SCB)*

Each Subexecution Control Block contains the context or references to context associated with its related unfinished subexecution. The monitor creates an SCB when a subexecution is called. The pointer to the SCB is pushed down on the TCL as explained earlier. As a subexecution requests more space or asks for data pointers, the monitor uses the proper SCB to store or fetch the necessary information. When space is returned, the monitor updates the proper SCB appropriately. Termination of a subexecution results in its SCB being returned to available space along with the space no longer needed by the calling subexecution.

An SCB is a block of contiguous cells obtained from the block-space pool. The number of cells per block may vary depending on the anticipated requirements of the subexecution. A minimum number of cells will always be allocated to contain immediate data and pointers to normal data requirements.

There are two types of entries in an SCB: immediate data entries and data pointers. Immediate data consists of "save console" information when a subexecution is entered, and "save console" and other program status information when an interrupt occurs that does not return to the interrupted code after its servicing. Data pointers are used to define the location of input, output, working storage blocks, extension of the SCB, etc. Each data pointer contains the name of the data block, the location of a cell which points to the cell containing the upper and lower boundaries of the block, the register to be loaded with a boundary, and information concerning the return of the space which is being pointed to.

## RE-ENTRANT CONTROL EXAMPLE

Table I shows five jobs which are known to the system. Each job has a Job Description Block (JDB) which was created when the job entered the system. Within each JDB is the list of Tasks to be done.

The monitor has scheduled three Tasks, which in this case are from different jobs. The ATQ shown in Table II indicates that Task "MSG Processor T" is waiting for I/O to complete, Task "MSG Processor T" (a different instance of the previous Task) is in execution, and Task "GO" is pending and will be executed when both instances of "MSG Processor T" Terminate or cannot proceed.

Each entry in the ATQ points to a TCL, shown in Table III. Task "MSG Processor T" points to the Header cell K which contains the name of its JDB (Terminal A) and points to the top of its LIFO list,  $k_1$ . The "T control" routine has "called" the routine "Update 3" which will resume when its wait condition Terminates.

Task "MSG Processor T" (2nd instance) points to its TCL entry S. The Header cell names the JDB (Terminal N) and points to the Top of the LIFO list  $s_1$ . The main routine "T control" is three levels down in routines "Interpret," SCAN, and SCAN (SCAN has re-entered itself once). The Tasks at locations  $E_1$  and  $E_2$  of the ATQ are both using program "MSG Processor T" with the first using the context at  $W_1$  and  $W_2$ , the second using the context at  $X_1$  through  $X_4$ . Task "GO" is associated with TCL U. Its Header cell names the JDB (Job—B) and points to the top of its LIFO list  $u_1$ . A program "Integrate" has been loaded and is now ready for execution with context at  $Y_1$ .

Each entry on a LIFO list points to an SCB which associates context with the instance of a routine. A minimum of  $m$  cells has been allocated for each SCB. Table IV shows the content of the SCB for one instance of a routine.

If another task can be accommodated by the computer, a task will be chosen from the Job Description area. The following example shows what occurs to this new contender for computer time. The scheduler selects Task "Load" of Job—C whose priority has changed to 4.

The monitor gets a cell from the SCL (cell  $E_4$ ). This cell is inserted in the ATQ by changing the link of cell  $E_1$  to  $E_4$  and putting  $E_2$  in the link of cell  $E_4$ . The console information associated with task "MSG Processor T" (of Terminal N instance) is saved in its SCB ( $X_1$ ), and its status in the ATQ is set to P. The name of the Task (Load), its priority (4), and the status (E) are inserted in cell  $E_4$ . Two more cells  $V$  and  $v_1$  are obtained from the SCL to establish the TCL at  $V$ . A block of space is obtained for the SCB of the main routine of "Load." The initial context for the task is then entered in the SCB beginning at  $Z_1$  and Load is then executed. Table V is a graphic representation of the structure created in the preceding example.

## RE-ENTRANT ROUTINE CONVENTIONS

By definition, a re-entrant routine is read-only in nature. Address calculations, internal indicators, subroutine parameters, and similar information must be stored and used external to the routine. The association of context to an instance of the routine is a function of the monitor and has already been treated. The following conventions are those considered im-

TABLE I. JDB'S (AUXILIARY OR MAIN STORAGE)

<div style="border: 1px solid black; padding: 5px; width: fit-content;"> <p>Job—B</p> <hr/> <p>Compile Load Go Print</p> </div>	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> <p>Terminal A</p> <hr/> <p>MSG Proc- essor T</p> </div>	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> <p>Terminal N</p> <hr/> <p>MSG Proc- essor T MSG Proc- essor R</p> </div>	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> <p>Job—C</p> <hr/> <p>Load Go Compile Go Print</p> </div>	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> <p>Terminal Q</p> <hr/> <p>MSG Proc- essor H MSG Proc- essor R</p> </div>
---	--	--	--	--

TABLE II. ACTIVE TASK QUEUE (ATQ)

Location	Link	Status	Priority	Pointer to TCL	Name
E <sub>1</sub>	E <sub>2</sub>	W	3	K	MSG Processor T
E <sub>2</sub>	E <sub>3</sub>	E	6	S	MSG Processor T
E <sub>3</sub>	0	P	7	U	Go-Integrate

W = waiting status due to I/O, etc.  
P = pending  
E = in execution

portant at present to get, use, and store the external data (context) required in any routine.

1. All routines must be called through the monitor.
2. Parameters required for inter-routine communication are contained in the calling routine's context. Return of control to the higher level routine is through the monitor also, so that return can be considered an implied call. To call a routine, the monitor is entered indicating:
  - a. The name of the routine being called (or returned to)
  - b. The name of the register which contains the pointer to the required context (if necessary).

TABLE III. TASK CONTROL LIST (TCL)

Location	Link	Routine Name	Pointer to SCB
K	k <sub>2</sub>	Terminal A	loc. of JDB
k <sub>1</sub>	0	T control	W <sub>1</sub>
k <sub>2</sub>	k <sub>1</sub>	Update	W <sub>2</sub>
S	s <sub>4</sub>	Terminal N	loc. of JDB
s <sub>1</sub>	0	T control	X <sub>1</sub>
s <sub>2</sub>	s <sub>1</sub>	Interpret	X <sub>2</sub>
s <sub>3</sub>	s <sub>2</sub>	Scan	X <sub>3</sub>
s <sub>4</sub>	s <sub>3</sub>	Scan	X <sub>4</sub>
U	u <sub>1</sub>	Job—B	loc. of JDB
u <sub>1</sub>	0	Integrate	Y <sub>1</sub>

3. To get a new block of cells for use, the monitor must be entered indicating:
  - a. Name to be assigned to the block allocated
  - b. Number of contiguous cells needed
  - c. Name of the register to be used (if any)
  - d. Value to be put in the register (either upper or lower boundary)
  - e. Return location or action if space is not available.
4. To re-establish a pointer in a register whose contents have been changed, the monitor must be entered indicating:
  - a. Name of the block
  - b. Value to be used (either upper or lower boundary)
  - c. Name of the register to be used if other than the previously associated register (if named, the previous association is lost).
5. To drop a register from use as a context pointer so that it can be used for other purposes, the monitor must be entered indicating the name of the pointer.
6. To return block space to the available space pool the monitor must be entered indicating the name of the block to be returned or the name of a list containing the blocks to be returned.
7. The responsibility of returning space rests with the routine which obtained the space. The termination of a subexecution will, however, result in all space requested for private use being returned to the block allocation pool.

TABLE IV. SUB-EXECUTION CONTROL BLOCK (SCB) FOR THE TASK "INTEGRATE"

Location	Control	Name	Register	Pointer to loc. pointing to Data
$Y_1-0$	1	IN 1	a	$\beta$
-1		IN 2		
-2	2	OU 1	c	$\delta$
-3		OU 2		
-4	1	WS 1	d	$\epsilon$
-5		WS 2		
-6		SCB		
-7		CONS		
.		.		} immediate data
.		.		
.		.		
$Y_1-7 + i$		CONI		} immediate data
.		.		
.		.		
-m				

Control 1 = can be returned  
 2 = Returned  
 3 = common data base

IN = INPUT  
 OU = OUTPUT  
 WS = Working storage  
 SCB = Additional SCB for additional space if required  
 CONS = Console save data  
 CONI = Console save data plus program status information at an interrupt

## SUMMARY

Association of context for an instance of a routine has been achieved through the use of control information created by the monitor or furnished to the monitor via program conventions. The organization of the control information is in the form of a list structure for ease of inserting and deleting new data.

The Active Task Queue is a list, ordered on

priority, used primarily for task sequencing. The Task Control Lists are LIFO lists which relate context in the Subexecution Control Blocks to routines which are associated with Tasks in the Active Task Queue.

This method of control, in conjunction with the conventions that a routine must follow, allows multiple entrances and executions of a routine before prior executions are completed.

TABLE V. GRAPHIC REPRESENTATION OF THE MONITOR DATA STRUCTURE

ATQ					TCL				SCB
Location	Link	Status	Priority	Pointer to TCL Name	Location	Link	Name	Pointer to SCB	Location
$E_1$	$E_4$	W	3	K MSG Processor T	K	$k_2$	Terminal A	Location of JDB	$W_1 \rightarrow \left. \begin{matrix} W_{1-0} \\ \vdots \\ W_{1-m} \end{matrix} \right\} \text{SCB Data}$
					$k_1$	0	T Control	$W_1$	
						$k_2$	Update 3	$W_2 \rightarrow \left. \begin{matrix} W_{2-0} \\ \vdots \\ W_{2-m} \end{matrix} \right\} \text{SCB Data}$	
$E_2$	$E_3$	P	6	S MSG Processor T	S	$s_4$	Terminal N	Location of JDB	$X_1 \rightarrow \left. \begin{matrix} X_{1-0} \\ \vdots \\ X_{1-m} \end{matrix} \right\} \text{SCB Data}$
					$s_1$	0	T Control	$X_1$	
						$s_2$	Interpret	$X_2 \rightarrow \left. \begin{matrix} X_{2-0} \\ \vdots \\ X_{2-m} \end{matrix} \right\} \text{SCB Data}$	
						$s_3$	Scan	$X_3 \rightarrow \left. \begin{matrix} X_{3-0} \\ \vdots \\ X_{3-m} \end{matrix} \right\} \text{SCB Data}$	
						$s_4$	Scan	$X_4 \rightarrow \left. \begin{matrix} X_{4-0} \\ \vdots \\ X_{4-m} \end{matrix} \right\} \text{SCB Data}$	
$E_3$	0	P	7	U Go Integrate	U	$u_1$	Job—B	Location of JDB	$Y_1 \rightarrow \left. \begin{matrix} Y_{1-0} \\ \vdots \\ Y_{1-m} \end{matrix} \right\} \text{SCB Data}$
					$u_1$	0	Integrate	$Y_1$	
$E_4$	$E_2$	E	4	V Load	V	$v_1$	Job—C	Location of JDB	$Z_1 \rightarrow \left. \begin{matrix} Z_{1-0} \\ \vdots \\ Z_{1-m} \end{matrix} \right\} \text{SCB Data}$
					$v_1$	0	Load	$Z_1$	



## APPENDIX A SPACE ALLOCATION

The structure of the control data needed for re-entrant and recursive routines is based on list-structure concepts. The use of a list structure approach requires being able to obtain and return space dynamically. Part of the space needed must be prestructured or linked in the IPL-V (Newell, Simon and Shaw)\* manner. Availability of space in blocks of contiguous cells is also required to gain a compromise for efficient use of core storage.

The following is a description of a single cell and block allocation scheme that was developed and implemented on the IBM 7094 by Mr. M. R. Needleman of WDPC-UCLA.†

### SINGLE CELL ALLOCATION

A relatively small number of contiguous cells are linked together to form the Single Cell available space List (SCL). A fixed cell is maintained which always points to the next available cell on the list. Table A-I shows this construction.

The allocation routine allocates a cell by giving the requestor the name of a cell (the address  $\alpha$ ) and updates the link of A to point to the next available cell on the list which is  $\beta$ . Table A-II shows the result of allocating 1 cell.

When a cell  $\tau$  is returned to the available space list, it is inserted at the top of the SCL as follows:

1. Cell A, which contains the pointer to the next available cell on the SCL, is modified to point to  $\tau$ .
2. The former pointer  $\beta$  is put into the link portion of the cell  $\tau$ .

Table A-III shows the results of this process.

\* The Rand Corp., Santa Monica, Calif., Newell A. Editor, "Information Processing Language—V Manual," Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1961.

† Western Data Processing Center, University of California, Los Angeles 24, California. The scheme was developed by WDPC under contract with the Advanced Research Projects Agency (Contract No. SD 184), Office of Director of Defense, Research and Engineering, Pentagon, Washington, D. C.

TABLE A-I. SINGLE CELL AVAILABLE SPACE LIST (SCL)

<u>Location</u>	<u>Link</u>	<u>Information</u>
A (fixed loc.)	$\alpha$	
$\alpha$	$\beta$	
$\beta$	$\gamma$	
.	.	
.	.	
.	.	
$\lambda$	0	

### BLOCK ALLOCATION

The second type of space allocation is called block allocation. A block of contiguous cells is reserved for this type of allocation. Two lists are used to identify the space available and space allocated. Cells for both lists are obtained from the Single Cell available space List.

Each entry in the block allocation list contains: a flag indicating whether or not the block is currently available; a pointer to the cell containing the addresses of the first and last locations of the block; and a link to the next cell on the Block Allocation List. The first cell on the list (header cell) always links to the last cell put on the list. Table A-IV shows the block allocation list after three requests for space.

Each cell on the block-limits list contains the address of the first and of the last cell allocated as a block by the block allocation routine and is in one-to-one correspondence to the Block

TABLE A-II. SINGLE CELL AVAILABLE SPACE LIST (SCL) AFTER ALLOCATING ONE CELL

<u>Location</u>	<u>Link</u>	<u>Information</u>
A (fixed loc.)	$\beta$	
$\beta$	$\gamma$	
$\gamma$	$\Delta$	
.	.	
.	.	
.	.	
$\lambda$	0	

TABLE A-III. SINGLE CELL AVAILABLE SPACE LIST (SCL) AFTER RETURN OF ONE CELL

Location	Link	Information
A (fixed loc.)	$\tau$	
$\beta$	$\gamma$	
$\gamma$	$\Delta$	
.	.	
.	.	
.	.	
$\lambda$	0	
$\tau$	$\beta$	

TABLE A-IV. BLOCK ALLOCATION LIST (BAL)

Location	Link	Flag	Pointer
$\alpha$	$\xi$	0	$\beta$
$\gamma$	0	1	$\Delta$
$\delta$	$\gamma$	1	$\epsilon$
$\xi$	$\delta$	1	$\eta$

Flag = 0, block is available  
 Flag = 1, block is being used

BLOCK LIMITS LISTS (BLL)

Location	First Location	Last Location	(Allocated to)
$\beta$	10K	20K	(available)
$\Delta$	40K + 1	45K	(Z1)
$\epsilon$	30K + 1	40K	(Z2)
$\eta$	20K + 1	30K	(Z3)

SPACE POOL MAP

10K	Available
15K	
20K	Z 3
25K	
30K	Z 2
35K	
40K	Z 1
45K	

Allocation List. An example of this list used in conjunction with the Block Allocation List (BAL) and a core map of the block allocation space pool is shown in Table A-IV.

The method of block space allocation is best illustrated by some examples. The first is a request for a block which is immediately available, the second is the return of a block, and finally a request for space which exceeds the length of any one available block. These examples will assume the previous state of the lists and blocks allocated and indicate only the allocation routine action. The monitor is the implied user. The first example starts with Table A-IV state.

In general, the user requests a block of N cells. The allocator assigns space and returns to the user via the monitor. The monitor sets the address of the cell containing the address of the first and of the last cell assigned in the SCB and places the base value in the register specified. To return space, the user indicates the name of the block, via the monitor, to be returned to the block space pool by the space allocation routine.

*Example 1*

A user (Z 4) requests 3000 cells of block storage. The block allocation routine goes through the following sequence.

1. From cell  $\alpha$  (in the BAL), get the limits cell  $\beta$  and the subsequent limits. In the rare case where the block is in use (Flag = 1) the coalescing of blocks, as outlined in Example 3 is done first.
2. Determine the number of cells available and determine if the request can be filled. (In this case assume the affirmative.)
3. Decrease the upper limit in cell  $\beta$  by the number of cells needed.
4. Get two cells ( $\theta$ ,  $\iota$ ) from the Single Cell available space List (SCL).
5. Insert cell  $\theta$  in the BAL with a link of  $\xi$  (obtained from cell  $\alpha$ ) and pointer to  $\iota$ . The address of  $\theta$  is put in the link field of  $\alpha$  and the flag of  $\theta$  set to 1.
6. Set the block limits in cell  $\iota$  equal to 17,001 and 20,000 respectively.
7. Return to the user with the address of cell  $\iota$ .

Table A-V shows the result of requesting a block of cells.

*Example 2*

User (Z 2) returns the block of space whose limits are found in cell  $\epsilon$ . The address of this cell is used to search the BAL for the pointer to this cell. As can be seen in Table A-VI, only the flag of the cell ( $\delta$ ) which contains the pointer to the block limits returned is changed (from 1 to 0). The block limits list is not altered.

*Example 3*

User (Z 5) requests 15,000 cells of block storage. The block allocation routine does the following.

1. Using cell  $\alpha$  (in the BAL) the contents of cell  $\beta$  are obtained. The number of

TABLE A-V. BLOCK ALLOCATION LIST (BAL)

Location	Link	Flag	Pointer
$\alpha$	$\theta$	0	$\beta$
$\gamma$	0	1	$\Delta$
$\delta$	$\gamma$	1	$\epsilon$
$\xi$	$\delta$	1	$\eta$
$\theta$	$\xi$	1	$\iota$

BLOCK LIMITS LIST (BLL)

Location	First Location	Last Location	(Allocated to)
$\beta$	10K	17K	(available)
$\Delta$	40K + 1	45K	(Z 1)
$\epsilon$	30K + 1	40K	(Z 2)
$\eta$	20K + 1	30K	(Z 3)
$\iota$	17K + 1	20K	(Z 4)

SPACE POOL MAP

10K	Available
15K	
20K	Z 4
25K	
30K	Z 3
35K	
40K	Z 2
45K	
	Z 1

TABLE A-VI. BLOCK ALLOCATION LIST (BAL)

Location	Link	Flag	Pointer
$\alpha$	$\theta$	0	$\beta$
$\gamma$	0	1	$\Delta$
$\delta$	$\gamma$	0	$\epsilon$
$\xi$	$\delta$	1	$\eta$
$\theta$	$\xi$	1	$\iota$

BLOCK LIMITS LIST (BLL)

Location	First Location	Last Location	(Allocated to)
$\beta$	10K	17K	(available)
$\Delta$	40K + 1	45K	(Z 1)
$\epsilon$	30K + 1	40K	(available)
$\eta$	20K + 1	30K	(Z 3)
$\iota$	17K + 1	20K	(Z 4)

SPACE POOL MAP

10K	Available
15K	
20K	Z 4
25K	Z 3
30K	
35K	Available
40K	
45K	Z 1

cells available is determined to be less than the number requested.

2. Link through BAL putting the limits pointer of each "in use" entry (Flag = 1) on a push down list. Each entry with a Flag = 0 (space returned) is returned to the single cell available space list along with its associated limits cell. The BAL cell returned is also deleted from the BAL list. The push down list cells are obtained from the Single Cell available space List. The entries in the list are now ordered such that the name of the cell containing the highest block limits is last on the list (therefore 1st off) and the name of the cell containing the lowest block limits is first on the list (therefore last off).

3. The method to coalesce available blocks is as follows. Move each used block up in core so as to pack them to the upper boundary of the space pool. This will push any scattered available space further and further down in core until it is engulfed by the limits of  $\beta$ ; i.e., all unused space is in one block at the lower boundary of the space pool. To implement the coalescing, the pointers to the used space limits are popped-up and limits are changed to reflect data movement

which is done when each new block of unused space is encountered. Table A-VII shows the results of coalescing space. Since the user points to a pointer to the block, the block can be moved and the pointer to it changed without concern by the user.

4. If the request for space can now be filled from the space available limits at  $\beta$ , the method of allocating the block is the same outlined in Example 1.

TABLE A-VIII. BLOCK ALLOCATION LIST (BAL)

<u>Location</u>	<u>Link</u>	<u>Flag</u>	<u>Pointer</u>
$\alpha$	$\theta$	0	$\beta$
$\gamma$	0	1	$\Delta$
$\xi$	$\gamma$	1	$\eta$
$\theta$	$\xi$	1	$\iota$

BLOCK LIMITS LIST (BLL)

<u>Location</u>	<u>First Location</u>	<u>Last Location</u>	<u>(Allocated to)</u>
$\beta$	10K	27K	(available)
$\Delta$	40K + 1	45K	(Z 1)
$\eta$	30K + 1	40K	(Z 3)
$\iota$	27K + 1	30K	(Z 4)

SPACE POOL MAP

10K	Available
15K	
20K	
25K	
30K	Z 4
35K	Z 3
40K	Z 1
45K	



# XPOP: A META-LANGUAGE WITHOUT METAPHYSICS

*Mark I. Halpern  
Research Laboratories  
Lockheed Missiles & Space Co.  
Palo Alto, California*

## INTRODUCTION

The XPOP programming system is a straightforward and practical means of implementing on a computer a great variety of languages—in other words, of writing a variety of compilers. The class of languages it can handle is not easy to characterize by syntactic form, since the system permits syntax specification to be varied freely from statement to statement in a program being scanned; the permitted class includes the best-known programming languages, as well as something closely approaching natural language. We believe that this distinguishes the XPOP processor from the syntax-directed compilers,<sup>1,2,3</sup> although it shares with them the fundamental idea that the process of programming-language translation can be usefully generalized by a compiler to which source-language syntax is specified as a parameter.

This paper describes only the more novel features of XPOP; a fuller treatment is available elsewhere.<sup>4</sup>

## DISCUSSION

XPOP consists of two major parts: (1) a generalized skeleton-compiler that performs those functions common to all compilers, and (2) a battery of pseudo-operations for specifying the notation, operation repertoire, and compiling peculiarities of a desired programming language. The programmer creates the compiler for such a language not by programming it from scratch but by using the XPOP

pseudo-operations to modify and extend XPOP itself, which then becomes the desired compiler.

The use of these facilities involves the creation by the programmer of functional units that superficially resemble the programmer-defined macro-instructions of, for example, IBCMAP (and in fact include such macros as a subset), but whose effects may be radically different from those obtained by use of conventional macros.\* An XPOP macro does not necessarily generate coding; its possible effects are so varied that it can best be defined simply as an element of the source program that, when identified, causes the processor to take some specified action. That action may be any of the following:

- (1) The parameterization of XPOP's scanning routine to make it recognize, either for the remainder of the source program or within some more limited domain, a new notation
- (2) The compilation of coding for immediate or remote insertion into the object program
- (3) The immediate assembly and execution of any of the instructions compiled from a source-language statement.

---

\* By "conventional macros" we mean the user-defined operators that some programming systems allow. The definition of a macro consists essentially of the assignment of a name to a block of coding, after which every appearance of that name as an operator causes the system to insert a copy of that coding into the object program.

- (4) The preservation on cards and/or tape of the language description currently in use, in a condensed format that can be redigested by XPOP at tape speed when read back in; also the reading-in of such a language from a tape file created earlier in the same machine run or during an earlier run
- (5) The production by XPOP of a bug-finding tool called an XRAY—a highly specialized core-and-tape dump giving the programmer the tables and strings produced by the system in structured, interpreted, and captioned form

In the illustrations of these features, some conventions that require explanation will be used. All programming examples offered are exact transcripts of the symbolic parts of actual XPOP listings. Lines prefixed by a dollar sign are records output by the processor as comments; these originate either as source-program statements printed out as comments for documentary purposes or as processor-generated messages notifying the programmer of errors or other conditions he should be aware of. No attempt is made to illustrate XPOP facilities by coding examples of any intrinsic value. The examples used are merely vehicles for the exhibition of those facilities and are therefore generally trivial in size and effect. The discussion that follows takes up the chief features of the system in the order of the five-point outline given earlier.

#### *Notation-Defining Pseudo-Operations*

Consider a macro, LOGSUM, created to store the logical sum of two boolean variables, A and B, in location C.

```

$LOGSUM  MACRO A, B, C
$          CAL    A
$          ORA    B
$          SLW    C
$          END

```

Having been defined, this macro may at once be called upon in XPOP's standard form, which requires that the macro's name be im-

mediately followed by the required parameters with commas separating these elements and the first blank terminating the statement. A standard-form call on LOGSUM would have this appearance and effect:

```

$ LOGSUM,ALPHA,BETA,GAMMA
CAL    ALPHA
ORA    BETA
SLW    GAMMA

```

Suppose we find standard-form notation unsatisfactory and want to call upon the function LOGSUM in the following form:

```

STORE INTO CELL 'C' THE LOGICAL
SUM FORMED BY 'OR'ING THE BOOLE-
EAN VARIABLES 'A' AND 'B'.

```

There are, from the XPOP programmer's viewpoint, four differences between the standard and the desired form:

- (1) The name of the function is no longer LOGSUM, but STORE.
- (2) The order in which parameters are expected by STORE differs from that of LOGSUM.
- (3) The punctuation required by the two forms differs; in standard form, the comma is the sole separator, blank the sole terminator. In the desired form, three kinds of separator are used:
  - (a) The one-character string 'blank'
  - (b) The two-character string 'blank-apostrophe'
  - (c) The two-character string 'apostrophe-blank'
and one terminator
  - (a) The two-character string 'apostrophe-period'
- (4) The desired form contains several "noise words"—that is, character strings present for human convenience but which XPOP is to ignore.

In the following illustration, we use its pseudo-ops to teach XPOP the new statement form, then demonstrate that the lesson has been learned by offering it the new form as input and verifying that it produces the correct coding. An explanation of each pseudo-op used follows the illustration.

```

$STORE  MACRO  A,B,C
$      LOGSUM  B,C,A
$      END
$
$      CHPUNC
$NEW PARAMETER-STRING PUNCTUATION ADOPTED AT THIS POINT
$
$
$      CHPUNC  3S1 2 '2' 1T2'.
$NEW PARAMETER-STRING PUNCTUATION ADOPTED AT THIS POINT
$
$      CHPUNC  1S2.. 1T1.
$NEW PARAMETER-STRING PUNCTUATION ADOPTED AT THIS POINT
$
$      NOISE   4INTO 4CELL 3THE 6LOGICA 6FORMED 2BY 6OR'ING 6BOOLEA
$
$      NOISE   6VARIAB 3AND 3SUM
$
$      STORE INTO CELL 'GAMMA' THE LOGICAL SUM FORMED BY 'OR'ING THE...
$      BOOLEAN VARIABLES 'ALPHA' AND 'BETA'.
$      CAL    ALPHA
$      ORA    BETA
$      SLW    GAMMA

```

The definition of STORE with which the above illustration begins deals with the first two of the four differences noted between the desired and the standard statements. It causes XPOP to recognize STORE as an operator identical in effect to LOGSUM, and specifies that the parameter expected as the third by LOGSUM will be expected as the first by STORE. The pseudo-op CHPUNC (CHange PUNCTuation) deals with the third difference. Its first use, with blank variable field, erases all punctuation conventions from the system; the comma is no longer a separator nor is the blank a terminator. Having thus wiped the slate clean, CHPUNC is used again to specify the required punctuation. The variable field that follows this second CHPUNC may be read: "Three separators—the one-character string *blank*, the two-character string *blank-apostrophe*, and the two-character string *apostrophe-blank*; also one terminator—the two-

character string *apostrophe-period*." (The additional punctuation specified by the third CHPUNC was introduced because the signal to XPOP that a statement is continued on the next card is the occurrence, at the end of each card's worth, of a separator immediately followed by a terminator; here the programmer wanted to use the string *'...'* for this purpose. A separate CHPUNC was necessary simply because the additional punctuation came as an afterthought.) The fourth and last difference is dealt with by means of the pseudo-op NOISE, which permits the programmer to specify character strings to be ignored by the processor. Since strings longer than six characters are taken as noise words if their first six characters are identical to any noise word, such strings as VARIABLE, VARIABLES, and VARIABILITY are effectively made noise words by the definition of 6VARIAB as an explicit noise word.



With these pseudo-ops given, XPOP has been taught the desired statement form, as proof of which it generates correctly parameterized coding when used as input. That statement was created, of course, only for illustrative purposes; few programmers would care to use so many words to generate three lines of machine-language coding. For an application in which documentation was an unusually important requirement, however, so elaborate a statement might serve a useful purpose—and real macros would average closer to 100 instructions than to 3.

The most important property of this technique for describing a notation to a processor, though, is the flexibility with which a notation so specified may be used. All that the XPOP programmer has explicitly defined is a number of individual words and punctuation marks, with no constraints on their combination; they may be used to form any statement that makes sense and conveys the necessary information to the processor. The programmer will often have a particular model statement in mind when specifying the vocabulary he wishes to use in calling for some function, but he will find that in implementing the model he has incidentally implemented an enormous number and variety of alternative forms.

If we add to our list of noise words the two strings OF and AS, we can use any of the following to generate the required coding:

- (a) STORE INTO GAMMA THE SUM OF ALPHA AND BETA.
- (b) STORE AS GAMMA THE LOGICAL SUM OF ALPHA AND BETA.
- (c) STORE AS LOGICAL GAMMA THE SUM OF THE VARIABLES ALPHA AND BETA.
- (d) STORE LOGICALLY INTO GAMMA 'ALPHA' AND 'BETA.'
- (e) STORE GAMMA ALPHA BETA.
- (f) LOGICALLY STORE INTO 'GAMMA' THE VARIABLES 'ALPHA' AND 'BETA.'
- (g) INTO GAMMA STORE THE SUM OF ALPHA AND BETA.

As (f) and (g) indicate, both noise words and operands may precede the operator, provided only that they are not themselves mistakable for operators. If, for example, INTO were an operator as well as a noise word (such multiple roles are possible and sometimes useful), statement (g) would be misunderstood as a call on INTO. Excepting such uncommon cases, the operator and operands in a statement may float freely with respect to noise words, and the operator may float freely with respect to its operands; the sole constraint is that the operands must be given in the order specified when the operator was defined. Even this last constraint will be relaxed when the QWORD feature is fully implemented. A QWORD is a noise word that, like an English preposition, identifies the syntactic role of the word it precedes; its use enables the programmer to offer operands in an order independent of that specified when the operator is defined. Applied to the statement type dealt with so far, the QWORD feature might be used thus:

```
STORE MACRO $INTO$C,A,B
      CAL   A
      ORA   B
      SLW   C
      END
```

The string \$INTO\$C informs the system that if the QWORD "INTO" appears in a call on STORE, the first operand following it is to be taken as corresponding to the dummy variable C. The use of the QWORD would override the normal C,A,B order and enable the user of STORE to write, as another alternative:

- (h) LOGICALLY STORE THE SUM OF ALPHA AND BETA INTO GAMMA.

Practically all notation-defining pseudo-ops may be used within macros as well as outside them, and the difference in location determines whether the conventions thereby established are 'local' or 'global.' If such pseudo-ops are given at the beginning of a macro definition that includes some non-pseudo-op lines as well, they are taken as local in effect. They will temporarily augment or supersede any notational conventions already established, and be

nullified when the macro within which they were found has been fully expanded. 'Local' notation-defining pseudo-ops will be put into effect in time to govern the scan of the very statement that calls on their containing macro. Such internally defined statements need respect the earlier conventions only to the extent necessary to permit their operators to be isolated. When pseudo-ops constitute the sole contents of a macro, they are taken as applying to the rest of the program in which they appear; the effect of calling on such a macro-ful of pseudo-ops is as if each pseudo-op were given as a separate input statement. Insofar as the notation a programmer requires is regular and self-consistent, then, it may be described in a single macro whose name might well be that of the language itself, and which would be called on at the beginning of any program written in that language. Statement forms that have special notational requirements in conflict with any global conventions would include the necessary local conventions within the bodies of their macro definitions. The local-notation feature will be illustrated in the next section.

As should be evident at this point, it is possible to teach XPOP to recognize an enormous number of logically identical but notationally different statements by means of a few uses of just those pseudo-ops introduced so far. It should be possible, in fact, to define a programming language empirically—that is, to treat a language as a cumulative, open-ended corpus of those statement forms that experience shows to be desirable. The full set of notation-defining pseudo-ops, of which about one-third is exhibited here, permits the description of the notations of FORTRAN, COBOL, and most other existing compiler languages.

#### *Compilation-Control Pseudo-Operations*

The compiler designer also needs, of course, various kinds of control over the compilation process. One requirement is for the ability to call for remote compilation. To meet this need XPOP provides the pseudo-ops WAIT and WAITIN. Both signify that the part of any macro lying within their range is to be ex-

panded as usual—that is, parameters substituted for dummy variables, system-generated symbols inserted where called for, and so on—but that the resulting coding is not to be inserted into the object program yet. Instead, these instructions are put aside, to be inserted into the object program only when a source-program statement is found bearing the statement label specified by the WAIT or WAITIN. (The label to wait for is specified in the pseudo-op's variable field, where it may be given as a literal constant or—more likely—represented by a dummy to be replaced by a parameter.) In any case, all instructions waiting for such a label will appear just after those resulting from the translation of the statement so labeled.

The instructions waiting for a label may have come originally from several various macros, or several uses of the same macro; if so, the one difference between WAIT and WAITIN will make itself felt. If, for example, a group of instructions lay within range of WAIT ALPHA, they would be *appended* to the threaded list of those already waiting for ALPHA; if the pseudo-op were WAITIN, they would be prefixed to it. Those groups of instructions made to wait by WAITIN's will, therefore, appear in the object program in the inverse of the order in which they occurred in the source program—hence "WAITIN" (WAIT INverse). If the label for which a batch of instructions is waiting never appears, the instructions do not appear in the object program. If no label is specified, they appear at the very end of the object program.

The following example shows the use of WAITIN in a simplified version of FORTRAN's "DO"—one that permits only the special case of subscripting that is formally identical to indexing. First, the source program that defines "DO" to XPOP, and then uses it in a two-level-deep DO nest: \*

\* Note that XPOP can process algebraic expressions. These may be used as source-language statements or within macros; when used within macros, they may contain dummy variables to be replaced by parameters when the macros are used, and those parameters may be arbitrarily long subexpressions. Subscripts, not now allowed, are being provided for.

```

J   EQU      2
K   EQU      4
    CHPUNC   4S1=1 1,2, 1T2
DO  MACRO    A,B,C,D,01
)A  AXT      C,B
    WAITIN   A
    TXI      *+1,B,01
    TXL      )A+1,B,D
    END

    DO 15 J=1,3
    DO 15 K=2,20,2
    PHI,J=RHO,J+BETA,J
15  TAU,K=PHI,J+4
    END

```

} definition  
of "DO"

} "DO" nest

And below, the object program produced by the above:

```

J   EQU      2
K   EQU      4
$
$   CHPUNC   4S1 1=1,2, 1T2
$NEW PARAMETER-STRING PUNCTUATION ADOPTED AT THIS POINT
$
$DO  MACRO    A,B,C,D,01
$)A  AXT      C,B
$   WAITIN   A
$   TXI      *+1,B,01
$   TXL      )A+1,B,D
$   END
$
$
$   DO 15 J=1,3
)0001 AXT      1,J
$
$   DO 15 K=2,20,2
)0002 AXT      2,K
$   PHI,J=RHO,J+BETA,J
    CLA      BETA,J
    FAD      RHO,J
    STO      PHI,J
$15  TAU,K=PHI,J+4
15   CLA      =4
    FAD      PHI,J
    STO      TAU,K
    TXI      *+1,K,2
    TXL      )0002+1,K,20
    TXI      *+1,J,01
    TXL      )0001+1,J,3
    END

```

Another obvious use for WAIT or WAITIN is the handling of closed subroutines. The programmer will frequently want a macro to generate only a calling sequence to a closed subroutine, with the subroutine itself appearing only once in the object program, at the end. To secure this effect, the programmer would define the macro in question as starting with the calling sequence; then he would incorporate a WAIT with blank variable field, a ONCE pseudo-op, and then the subroutine. If the macro were not used in a given source program, the subroutine would not be made part

of the object program. If used, the first such use would output the calling sequence normally, and the subroutine as waiting instructions to be put into the object program at its end. Subsequent uses of the macro in that program would cause the compilation of the calling sequence only, the ONCE pseudo-op reminding XPOP that it had already compiled the subroutine. The following examples will illustrate uses of WAITIN, ONCE and local notation-defining pseudo-ops. The first is the pseudo-DO with its punctuation defined within its own body:

```

$DO      MACRO      A,B,C,D,01
$        CHPUNC    4S1 1=1,2,      1T2
$)A      AXT        C,B
$        WAITIN    A
$        TXI       *+1,B,01
$        TXL       )A+'1,B,D
$        END
$
$
$        DO 15 J=1,3
$        CHPUNC 4S1 1=1,2,      1T2
$NEW PARAMETER-STRING PUNCTUATION ADOPTED AT THIS POINT
)0001    AXT        1,J
$
$        DO 15 K=4,48,TWO
)0002    AXT        4,K
$        PHI,J=RHO,J+BETA,J
        CLA        BETA,J
        FAD        RHO,J
        STO        PHI,J
$15     TAU,K=PHI,J+4.
15      CLA        =4.
        FAD        PHI,J
        STO        TAU,K
        TXI       *+1,K,TWO
        TXL       )0002+1,K,48
        TXI       *+1,J,01
        TXL       )0001+1,J,3
        END

```

A use of ONCE is shown next. ONCE may be used in either of two ways, depending upon whether its variable field is blank or not. When the macro in which it occurs is being expanded and a ONCE with blank variable field is encountered, the name of the macro is searched for in a table. If it is found, the rest of that macro is skipped; if not, it is entered

in the table to be found on later searches and expansion continues. The procedure followed if a symbol *is* found in the variable field differs only in that the symbol found is used rather than the name of the macro being expanded. This type of use permits copies of a subroutine, a set of constants, or a storage reservation to be incorporated into the definitions of many

different macros, with assurance that they will appear in the object program if and only if one of the macros is used, and not more than once no matter how many of them are used. It is this second type of use that is now shown:

```

$FIRST  MACRO  A,B,C
$        CLA    A
$        ADD    B
$        ONCE   M
$        STO    C
$        END
$
$
$
$
$SECOND MACRO  X,Y,C
$        LDQ    X
$        MPY    Y
$        ONCE   M
$        STO    C
$        END
$
$        FIRST,ALPHA,BETA,GAMMA
$        CLA    ALPHA
$        ADD    BETA
$        STO    GAMMA
$
$        SECOND,PHI,RHO,GAMMA
$        LDQ    PHI
$        MPY    RHO
$
$        END

```

Last among the compilation-control pseudo-ops that will be discussed here is XPIFF, which permits the programmer to specify conditions whose satisfaction is a prerequisite to the compilation of the next line of coding. (It is, of course, a direct development of the IFF familiar to users of the FAP-IBMAP family of assemblers.) The IFF is almost entirely restricted to testing conditions involving source-program symbols; the direction in which XPIFF is being developed is that of greater range of reference. The conditions upon which XPOP compilation may be made contingent will include many referring not to source-program symbols but to the system itself. When fully developed, this facility should bring within the compiler-writer's reach the means

of specifying as much object-program optimization as he wishes, short of that which, like FORTRAN's, depends on a flow-analysis of the entire compiled program.

The kind of optimization available through XPIFF in its present state is indicated by the following illustration, where it is used to avoid compiling loop-initializing and -testing instructions where they are unnecessary.

```

$MOVE   MACRO  A,B,O
$        XPIFF  O,X,X
$        MOV MOR A,B,O
$        XPIFF  O,X,Y
$        MOV EL A,B
$        END
$
$
$MOV MOR MACRO  Q,E,D
$        AX T   D,4
$)A      CLA    Q+1,4
$        STO    E+1,4
$        TIX    )A,4,1
$        END
$
$
$MOV EL   MACRO  L,M
$        CLA    L
$        STO    M
$        END
$
$
$        MOVE,ALPHA,BETA
$        XPIFF  O,X,X
$        XPIFF  O,X,Y
$        CLA    ALPHA
$        STO    BETA
$
$
$        MOVE,ALPHA,BETA,5
$        XPIFF  5,X,X
$        AX T   5,4
$)0002   CLA    ALPHA+1,4
$        STO    BETA+1,4
$        TIX    )0002,4,1
$        XPIFF  5,X,Y
$        END

```

#### *XECUTE Mode—A Compile-Time Execution Facility*

The XPOP processor may at any point in a source program be switched into XECUTE mode, in which succeeding source-language

statements are not only compiled but assembled and executed. The programmer switches into this mode by using the pseudo-op XECUTE, and reverts to normal processing by using the pseudo-op COMPYL; the coding between each such pair is assembled as a batch, then executed. XECUTE mode may be used with great freedom. The programmer may enter and depart it within a macro; while in the mode he may use macros (with full notational flexibility), algebraic expressions, and everything else that XPOP normally processes except certain pseudo-ops that would be meaningless at compile time. XECUTE mode was originally implemented by those working on the XPOP processor for their own use in maintaining and developing that program, and has proved itself better for such tasks than any other method we know. It enables us to patch XPOP in a symbolic language practically identical to the FAP language in which the processor itself is written, and to cause these patches to become effective at such points during a compilation as we choose—not necessarily at load time. The effectiveness of any such patch can be made contingent on results of program execution thus far, so that tests otherwise requiring several machine runs can be accomplished in one. A FAP-like assembly listing is produced by XPOP while in XECUTE mode, and the symbolic language employed is so nearly identical to FAP that the very cards used for XECUTE-mode patches can later be used for FAP assembly-updating.\*

But this facility is by no means usable only by those working on the processor itself. It has the further role of giving the compiler-designer working with XPOP the ability to specify pseudo-ops for his compiler, and make it perform any compile-time functions it requires that are not built into XPOP—building special tables, setting flags, and so on. It enables the designer to make his system, to any extent he wishes, an interpreter rather than a compiler, or a monitor/operating system rather than a language processor.

Compile-time execution makes a great variety of special effects readily available to the

---

\* We have produced a subroutine, entirely independent of XPOP, equivalent to XECUTE mode, and hope soon to announce its general availability.

programmer. For example, it allows any macro to be used recursively: just before calling on itself, such a macro switches into XECUTE mode, makes whatever test is required to determine whether further recursion is indicated, then switches back to compile normally either at or just after the internal call, depending on the outcome of that test. Another useful facility it affords is that of trapping any source-language statement type for such purposes as counting the number of uses made of it, taking snapshots of its variables before their values are changed, or debugging by testing the values of a procedure's variables just before exiting from it. Such trapping could be done even at the machine-language level. If the programmer wanted to trap all TRA instructions, for example, he would define TRA to be a macro, enter XECUTE mode within that macro to take the desired compile-time action, then return to normal processing. (The pseudo-op ULTLEV—ULTimate LEVel of expansion—would be used within such an op-code/macro to prevent the taking of a TRA instruction within the TRA macro as a recursive call, with resulting infinite regress.)

One purpose of replacing op codes by macros of the same name might be to cause each such extended operator to step a programmed clock at execution time (as well as executing the original op code, of course), so that the programmer can learn exactly how long his routines take to run—a critically important matter in real-time applications, which require that programmed procedures fit into time slots of fixed size. This capability, together with its notational-flexibility and immediate-execution features, makes XPOP particularly suitable for command and control programming.<sup>6</sup>

#### *Language-Preserving Pseudo-Operations*

XPOP provides the programmer with a group of three pseudo-ops that enable him to order, at any points in his program, that all macros so far defined be punched onto binary cards, written onto tape, or both. The use of any of these pseudo-ops preserves all macros then in the system in a highly compact form (binary-card representation takes about one-sixth the number of cards that symbolic takes) and, more important, a form that can be read

into the system at tape speed on any later XPOP run, without the time-consuming process of scanning and compressing the symbolic-language definitions. Notation-defining macros may, of course, be preserved on cards and/or tape along with code-generating macros. The tape and/or card deck produced may thus contain a complete programming language of the programmer's own design in both vocabulary and notation. This language may then be changed in any respect during the course of any ordinary production or debugging run. Functions may be added or deleted, notation elaborated or simplified. Because any of these pseudo-ops can be used as often as desired in a single program, it is possible to preserve successively larger sets of macros, each set containing its predecessors as subsets, as well as any macros defined since. Each time macros are punched or written out by means of any of these pseudo-ops, a report is generated, giving an alphabetized list of the macros preserved and the percentage of the system's macro capacity they occupy.

Another two pseudo-ops are available for ordering, either during a later XPOP run, or later in the same run, that predefined macros be read in either from the input tape (if they had been preserved on cards) or a reserved tape (if they had been preserved on tape). Sets of preserved macros may be read into the

```

$                WMDT      TEST,A6
$THE FOLLOWING MACROS HAVE BEEN OUTPUT ON TAPE
$      TEST2
$      TESTER
$      TESTXC
$ 00 PER CENT OF AVAILABLE SPACE HAS BEEN USED
$                RMDT      TEST,11
$ALL PREVIOUS MACROS HAVE BEEN DESTROYED BY THE USE OF RMDT

```

#### *Debugging Tools—The XRAY*

XPOP provides one unconventional tool for finding bugs that our experience has shown to be highly useful, and which might readily be incorporated into other systems. This is the XRAY—a structured, interpreted, and captioned dump of core memory and the output tape. It prints out the chief buffers, tables, and character-strings in the system in meaningful format and (where one exists) external representation, as well as all the program com-

system at any point in any program, making it possible to switch languages in midprogram. This greatly facilitates the consolidation into one program of sections written by several programmers using different XPOP-based languages—each section simply begins by reading into the system the language in which it was written.

The five pseudo-ops, and their exact effects, are given in Table 1.

As is shown in the following example, the programmer may override XPOP's built-in assumptions about the tapes that WMDT, WAPMD, and RMDT refer to. He does so simply by specifying, either by logical or by FORTRAN tape designation, the unit he wishes to address. He may also assign a name to each file when he creates it, and later retrieve it by name; this permits many languages to be stacked on a tape while sparing the programmer any concern over the position of the one of interest to him. In the example below, the programmer has used WMDT to write his language onto logical tape A6 under the name 'TEST'. His language consists of three macros, whose names are then listed by XPOP. (Since the amount of available core storage used by these three was less than one-half percent, it is given as zero percent.) He then read this language back in again, this time addressing the tape by its FORTRAN designation, 11.

piled so far (whether still in core or already on tape), and a standard octal dump of as much of core memory as the programmer may require. In case of system trouble or source-program trouble not covered by one of XPOP's 50-odd error messages, the first thing the XPOP programmer will want to check is that the macro definitions were properly accepted and packed away, and these definitions are accordingly converted back to original input form and exhibited first. Because these defini-

TABLE 1 THE LANGUAGE-PRESERVING PSEUDO-OPERATIONS

Pseudo-op	Meaning	Action Caused
WMDT	Write Macro-Definition tape	Writes definitions and associated information in binary on logical tape A5
PMDC	Punch Macro-Definition Cards	Writes definitions and associated information in card-image format on system punch tape
WAPMD	Write and Punch Macro Definitions	Causes both tape files described above to be written
RMDT	Read Macro-Definition Tape	Reads in from logical tape B5 a binary file created by a 'WMDT' or 'WAPMD'
RMDC	Read Macro-Definition Cards	Reads in from the input tape binary records representing a deck produced by a 'PMDC' or a 'WAPMD'

tions, as seen in an XRAY, have undergone both compression into internal form and expansion back into input form, the programmer who can recognize his macros there can feel some assurance that they were properly digested by XPOP. He will next want to see how the system has scanned the last statement it saw; for this purpose he is given a print-out of the table that shows what symbols XPOP extracted from that statement as the parameters, and how it paired them off with dummy variables. Following this he is shown that part of the compiled program still in the system's output buffer, then that part already written out onto tape. Finally, the XRAY will present as much of core memory in standard octal dump format as the programmer may have specified in the variable field of the XRAY pseudo-op that triggers this output.

XRAYs can be obtained in two ways. One is to use the pseudo-op explicitly at whatever points trouble has shown up in a previous run, or is to be feared; the other is to order compilation in XPER (eXPERimental) mode, which may be started at any point in the program by use of the pseudo-op XPER. In this mode, the detection by XPOP of any error in the source program or the system itself causes the generation of an XRAY—and one will be generated at the end of the program in any case. The two methods may be combined, the programmer calling explicitly for XRAYs at some points as well as compiling all or parts of his

program in XPER mode. The information presented by an XRAY as presently constituted is not fully adequate (hence the selective octal dump as a backup), and additions to it are being made, but experience indicates that the gain in intelligibility of information presented in XRAY form over that given in octal dumps is great enough to mark a step forward in bug-finding methods, as we think that XECUTE mode does in bug-correction. The joint power of the two source language facilities suggests the possibility of some experiments in on-line debugging; we hope to report on these later.

#### ACKNOWLEDGMENTS

The general macro-instruction concept, as well as many details of format, are derived from the BE-SYS systems created at Bell Telephone Laboratories and generally associated with the names D. E. Eastwood and M. D. McIlroy.

Three projects similar at least in spirit to XPOP but which came to the author's attention too late to play any part in the XPOP design are the Generalized Assembly System (GAS) of G. H. Mealy,<sup>7</sup> the Self-Extending Translator (SET) of R. K. Bennett and A. H. Kvilekval,<sup>8</sup> and the Meta-Assembly Language of (presumably) D. E. Ferguson.<sup>9</sup>

At Lockheed Missiles & Space Company the author's principal debt is to B. D. Rudin, W. F.



Main, and C. E. Duncan for steady faith and support over long bleak stretches. Much of the coding of the processor and most of the daily problems fell to W. H. Mead, Marion Miller, M. Roger Stark, and A. D. Stiegler.

The author is grateful to C. J. Shaw of System Development Corporation for an acute critique of XPOP that has helped to improve the presentation and to pinpoint the areas in which further development is most needed.<sup>10</sup>

Thanks are also due to P. Z. Ingerman of Westinghouse Electric Corporation for useful discussions on the relationship between XPOP and syntax-driven compilers, and for the opportunity to read part of his forthcoming book on such compilers.<sup>3</sup>

#### REFERENCES

1. IRONS, E. T., "A Syntax Directed Compiler for ALGOL 60," *Communications of the ACM*, January 1961, pp. 51-55.
2. FLOYD, R. W., "The Syntax of Programming Languages — A Survey," *IEEE Transactions on Electronic Computers*, EC-13, August 1964, pp. 346-353.
3. INGERMAN, P. Z., *A Syntax Oriented Translator* (New York: Academic Press, to be published).
4. HALPERN, M. I., *An Introduction to the XPOP Programming System*, Lockheed Missiles & Space Co., Electronic Sciences Laboratory, January 1964.
5. ———, "Computers and False Economics," *Datamation*, April 1964, pp. 26-28.
6. ———, *A Programming System for Command and Control Applications*, Technical Report 5-10-63-26, Lockheed Missiles & Space Co., July 25, 1963.
7. MEALY, G. H., *A Generalized Assembly System*, Memorandum RM-3646-PR, The RAND Corporation, August 1963 (2nd printing).
8. BENNETT, R. K., and A. H. KVILEKVAL, *SET: Self-Extending Translator*, Memo TM-2, Data Processing, Inc., March 3, 1964.
9. FERGUSON, D. E., "The Meta-Assembly Language," address presented before the Special Interest Group on Programming Languages, Los Angeles Chapter of ACM, July 21, 1964 [information taken from the announcement].
10. SHAW, C. J., "On Halpern's XPOP," System Development Corporation, unpublished, undated [early 1964].

# A 10 Mc NDRO BIAX MEMORY OF 1024 WORD, 48 BIT PER WORD CAPACITY

*William I. Pyle  
Theodore E. Chavannes  
Robert M. MacIntyre  
Philco Corporation  
Ford Road, Newport Beach, California*

## INTRODUCTION

Most of the approaches to fast read access memories in the past have been centered about the achievement of either faster conventional destructive switching, or the use of various non-destructive readout techniques and storage devices. Many of these techniques have inherent drawbacks for very fast read operation, such as the necessity for rewriting, in the case of conventional switching approaches, or the lack of truly non-destructive properties. The memory system described in this paper solves these problems by utilizing the BIAX memory element, with its inherently non-destructive readout properties, in a system organized to minimize circuit delays and utilize transmission line properties for the various signal paths. In this manner it is possible to achieve random read access times of 85 nanoseconds maximum since most inductive components are incorporated into the various transmission lines with the lines being terminated in their characteristic impedance. Not only is the memory designed for very high readout rates in the non-destructive mode, but it is electrically alterable with conventional linear select methods in five microseconds or less.

The sections which follow will describe the system design concepts, operation of the BIAX memory system, and the circuit and packaging designs which were used to achieve the system performance.

## SYSTEM DESCRIPTION

### *System Design Goals*

The basic goals of the memory program were to design and construct an operating model of a 1024 word, 48 bit per word memory capable of 10 Mc. random access non-destructive readout (NDRO) while being electrically alterable with a write cycle time of five microseconds. Although the performance requirements were of prime concern it was nevertheless necessary to utilize state-of-the-art components to insure that a practical system would ultimately result. Table I outlines the system characteristics which resulted.

### *System Organization*

The organization of any memory system is, in general, related to the desired speed of operation. If the primary design goal is the achievement of very short read access time it is usually mandatory that parallel operation of

- CAPACITY: 1024 WORDS, 48 BITS PER WORD
- REPETITIVE READ CYCLE TIME: 100 NANoseconds
- READ ACCESS TIME: 85 NANoseconds (MAXIMUM)  
(RANDOM ACCESS)
- REPETITIVE WRITE CYCLE TIME: 5 MICROSECONDS
- REPETITIVE WRITE/READ CYCLE TIME: 10 MICROSECONDS

Table I. Memory System Characteristics.

many parts of the memory be employed. The block diagram of Figure 1 shows how this type of parallel organization is employed to achieve 10 Mc. NDRO operation. In this diagram it is seen that the flow of information for a typical readout operation is through the input buffer, read decoder, interrogate drivers, BIAx array, sense amplifiers, and output register. To achieve the goal of 85 nanoseconds read access time, the propagation delays through the functional parts of the system as shown in Figure 2<sup>1</sup> were necessary.

The achievement of these propagation delays necessitated the use of certain specific organizations of the circuitry within the memory read system. These organizational factors and how they were applied to the 10 Mc. NDRO memory are listed below.

- 1) Every signal path involved in the read operation, including interconnections, must be considered in terms of its transmission line characteristic impedance and propagation delay. This is especially true in the BIAx array where the inductance of wires passing through many elements is substantial.
- 2) When the array signal transmission paths are portions of transmission lines, the total array delay is approximately the sum of the interrogate line delay plus sense line delay. Therefore the minimum total array delay usually results when the number of array words is approximately equal to the number of bits per word. In this memory, the 256 word by 192 bit per word array organization permits achievement of near minimum delay within the array.

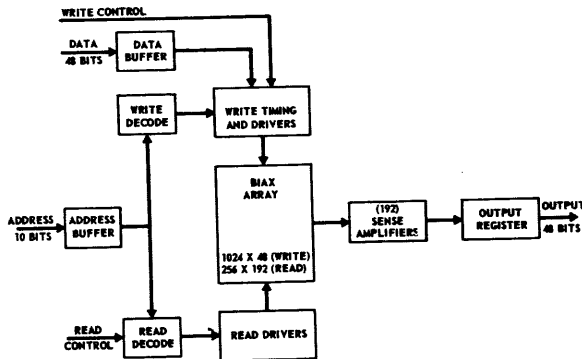


Figure 1. 10 Mc. NDRO BIAx Memory Organization.

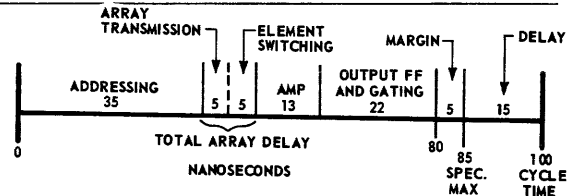


Figure 2. 10 Mc. BIAx NDRO Cycle.

- 3) Read address decoding must be accomplished at as low a signal level as practical, with high level gating kept to a minimum. In order to effectively accomplish this end it is necessary to use one interrogate driver per array word and a 1 of 256 decoder. Although 256 interrogate driver circuits are used each circuit is simple since it drives a transmission line terminated in its characteristic impedance.
- 4) The BIAx output signals produced by the interrogation of a word must be strobed at the earliest possible time following the interrogation. In this memory it is accomplished by strobing the sense amplifier with timing pulses derived from the array itself. By using these array-derived strobing pulses each sense amplifier output is strobed at an optimum time and variation in signal delays due to physical location of the word within the array or degradation of the interrogate pulse rise time is automatically compensated for.
- 5) All circuits associated with the NDRO portion of the memory must be located as close as possible to the array to minimize interconnection delays. In the memory this is accomplished by arranging the read circuits on two sides of the array, and making interconnections via twisted pair lines.

### Memory System Design and Operation

The memory described here has two basic modes of operation, non-destructive readout and, a writing mode, both of which utilize linear or word select techniques for address selections.

### NDRO Mode

The basic concept employed to achieve non-destructive readout in the BIAx element is

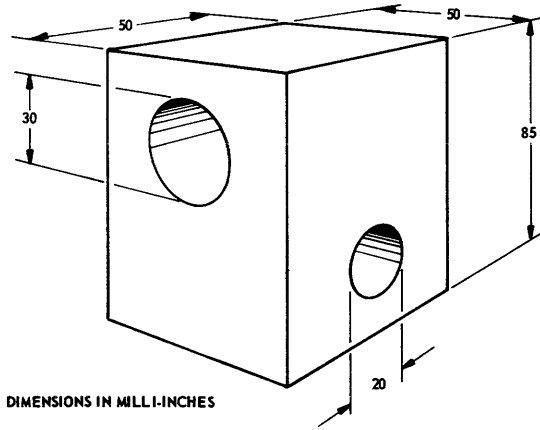


Figure 3. Nominal BIAX Physical Characteristics.

one involving crossed or quadrature magnetic fields in a common volume of square loop magnetic material.<sup>2-5</sup> The BIAX element used in the 10 Mc. memory is a pressed block of ferrite material having two non-intersecting orthogonal holes. The physical dimensions are approximately 50 x 50 x 85 milli-inches (mils) with two circular holes, one 30 mils in diameter, the other 20 mils in diameter (Fig. 3). Information is stored by saturating the magnetic material around the 30 mil hole (the storage hole). The storage hole contains the windings necessary to write into the memory element and to sense signal output. The inter-

rogate hole contains a single conductor for interrogation of the memory element.

Interrogation of the element is accomplished by applying a current producing flux in the same direction as flux already established around the interrogate hole. The current causes the domains in the common volume to be re-oriented toward the direction of the flux linking the interrogate hole. This reorientation decreases the flux linking the storage hole and thereby gives rise to a  $d\phi/dt$  voltage on the sense winding passing through the storage hole. The polarity of this voltage is dependent on the orientation of the flux linking the storage hole, consequently, a selected polarity of element output voltage will be observed for a ONE and the opposite polarity for a ZERO (See Figure 4C). Upon termination of the interrogate pulse, the domains in the common volume revert back to their original permanent flux condition and a true non-destructive read-out is achieved. Several advantages result from the use of this principle as employed in the BIAX memory element. First, the interrogate process introduces no measurable delay in the read operation and is therefore quite applicable to very high speed reading. Secondly, since the interrogation process involves only shuttling of flux around the interrogate hole, the inductance of the wires passing through a

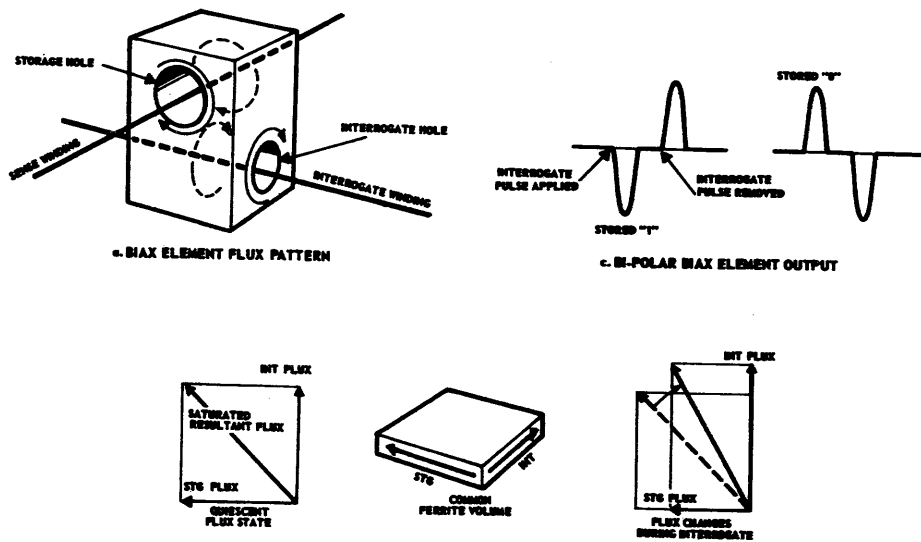


Figure 4. The BIAX Principle.

number of elements is sufficiently linear to permit low loss wide bandwidth transmission lines to be constructed using the BIAx element inductance and the associated array capacitance. By using the array construction techniques described later in this paper, it was possible to achieve transmission line impedance as low as 200 ohms while propagating pulse rise times less than 5 nsec.

NDRO operation of the memory is initiated upon receipt of a clocked read command after the address levels have stabilized. The ten address bits and their complements are converted by the input buffer to levels required by the read address decoder. The decoder selects one unique path of the possible 256 and activates the interrogate driver connected to the decoder output. The actual decoding process starts with the occurrence of the clocked read command and proceeds through the various levels of the decoder at a rate limited only by the response of the circuits in the path corresponding to that address. Figure 5 shows the functional breakdown of the input portion of the memory. To accomplish the required 1 of 256 decoding, it is seen that two decoders, a 64 place, and a 4 place, are used. The primary advantage of this method is that it minimizes the number of gating levels since the decoders operate in parallel. The 4 place decoder is a clocked unit, while the 64 place is unclocked and the outputs of the two decoders are combined at the input of the interrogate driver

circuits with another level of gating. Since the memory array is organized as 256 words of 192 bits, only eight of the ten address bits are required for decoding at the input to the memory, with the remaining two address bits being employed to select the desired 48 bits (of the 192 available) to be transferred to the 48 bit memory output register.

When a particular interrogate driver has been activated, it is necessary to extract the stored information from the array within 10 nanoseconds if the total memory read access time of 85 nanoseconds is to be achieved. To understand the difficulty of achieving the 10 nanosecond array delay with conventional constant current techniques, consider the following calculations: Assume that each interrogate line consists of approximately 200 elements, each exhibiting an inductance of 30 nanohenries. By lumping all the inductances, a total inductance of 6 microhenries would result. Using conventional constant current drive techniques, to achieve 80 ma. within 10 nanoseconds would require the following voltage:

$$E = \frac{L \Delta I}{\Delta T} \quad (1)$$

$$E = \frac{6 \cdot 10^{-6} (80 \cdot 10^{-3})}{10^{-8}} \quad (2)$$

$$E = 48 \text{ V} \quad (3)$$

It was felt that not only would a 48 volt current source be impractical since 256 were re-

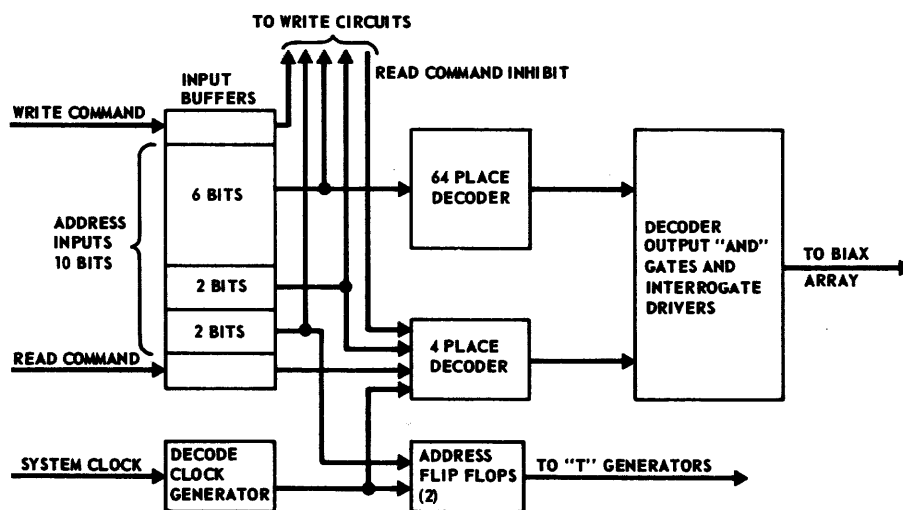


Figure 5. 10 Mc. BIAx NDRO Memory Input Block Diagram.

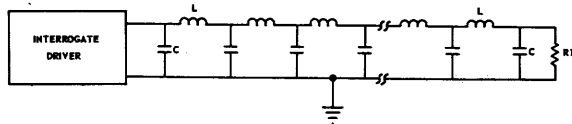


Figure 6. Simple BIAx Interrogate Line Equivalent Circuit.

quired, but it also would introduce reactive transients which would seriously limit the maximum interrogation rate. In order to bring the required interrogate drive voltage within practical limits, and to minimize transients, the terminated transmission line concept of operation was employed in the memory array. Figure 6 shows the schematic representation of a simple BIAx transmission line. In this figure, each lumped inductance is represented by one BIAx element through which an interrogate wire passes, and the capacitance is that between the wire and the ground plane.

If one calculates the properties of the transmission line<sup>6</sup>, assuming an element inductance of 30 nH per element, with elements spaced at approximately 0.125 inch intervals and located above the ground plane, the line will exhibit a characteristic impedance of approximately 500 ohms. Were a line with such a high characteristic impedance to be used for the memory interrogate line, certain problems would be encountered. First the drive voltage required to achieve 80 ma. interrogate current would be 40V, and even with a constant voltage driver, it is excessive from a practical circuit standpoint. Secondly, such a large excursion in voltage on the interrogate line introduces noise onto the sense line by capacitive coupling through the element, even though this capacitance is only about 0.01 pf. per element. Third, if this transmission line consisted of 200 sections, corresponding to the required word length in the array, the delay would be approximately 12 nsec. (Fig. 7). In order to alleviate these problems, several steps were taken to alter the electrical length, impedance and driving characteristics of the lines. These steps are described briefly below.

To reduce the driving voltage requirements, the line impedance was reduced by two means. First, the elements were offset as shown in

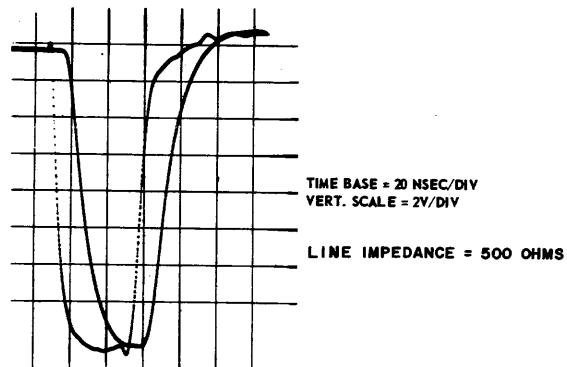


Figure 7. Interrogate Pulse Propagation Through 200 Element Transmission Line.

Fig. 8 and treated essentially as two transmission lines in parallel, and further split into two additional parallel lines. Since each wire passes through only half as many elements (and inductance) per unit of capacitance as for the single line, the impedance is reduced to 0.7 of the single line value. It should be noted that the delay per section of line is actually greater for the offset placement by a factor of 1.4, but since only half as many sections are employed (by driving in parallel), the net propagation delay is reduced to 0.7 of the single line value. The second means employed to reduce the impedance of the interrogate line is also shown in Fig. 8. This method consists of introducing a perforated metallic shielding mask around each element between the two holes. This increases the capacitance per section by approximately another factor of three, and brings the characteristic impedance down to approximately 200 ohms.

The methods described above, employed to reduce the transmission line characteristic impedance, did reduce the drive voltage requirements to about 12 V and as a result, the capacitive coupling to the sense line through the BIAx element was reduced accordingly. Even so, an objectionable amount of noise was still observed due to the coupling. Two measures were taken to eliminate this problem. The offsetting of the elements as shown in Fig. 8 necessitated driving the two lines in parallel. Because of the inherent properties of the BIAx element, interrogation can be accomplished with either polarity pulse, if it is in the

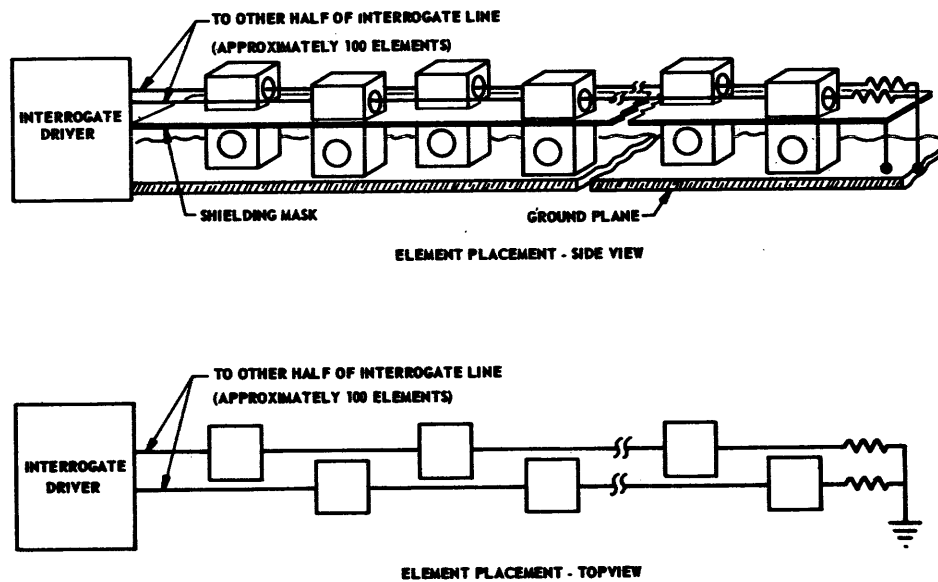


Figure 8. Dual BIAX Interrogate Line—Physical Configuration.

same direction as the previously established flux around the interrogate hole. This property of the BIAX element was used to reduce the capacitively coupled noise to the sense line by using pulses of equal amplitude and simultaneous rise times but of opposite polarity applied to the offset lines. Since a given sense line crosses both of these offset interrogate lines, the total capacitive coupling is reduced to a value proportional to the algebraic sum of the opposite polarity interrogate voltages during the rise time. Since this method did not provide perfect cancellation of the capacitively

coupled noise, an additional method was employed to provide partial cancellation of remaining noise on the sense line. In Fig. 9 it is seen that the sense line is divided into eight segments of 32 bits each. Within each segment, the electrical length of the line is short compared to the interrogate pulse rise time, and one end of each segment is returned to ground. When capacitive noise is introduced into the segment, it propagates to the grounded end and is reflected back to the source inverted, providing effective cancellation of the noise pulse.

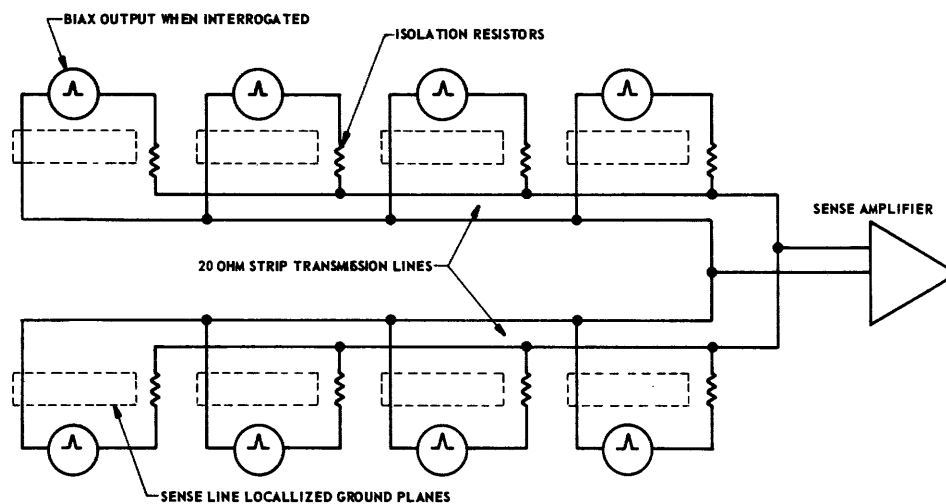


Figure 9. Sense Line Summing Equivalent Circuit.

When an output is produced from an element by interrogation, the isolation resistors shown in Fig. 9 create, in effect, a constant current source. The sense amplifier is then designed with a very low input impedance to provide compatibility with the sense line summing method. In the present memory, the sense amplifier has an input impedance of approximately 15 ohms, and receives its input from the array not more than 10 nsec after the 50% point of the interrogate pulse.

When the signals are observed at the output of the sense amplifier the time delay (relative to the interrogate pulse) depends both upon the physical word location relative to the sense amplifier and the location of the bit in the word relative to the interrogate driver. In the present memory, this delay ranges from a minimum of essentially zero to a maximum of ten nanoseconds, not including the delay through the sense amplifier. This variation, added to the variation in decoding delay, renders it difficult, if not impossible to strobe all 192 sense amplifiers reliably with a pulse fixed in time while still maintaining the required access time. To avoid this problem, the pulse used to strobe the sense amplifier is derived from the same region of the array as is the information. This can best be understood by considering Figure 10. Each group of 48

sense amplifiers is accompanied by a 49th bit, identical to the other 48, which provides the input to a pulse generator ("T" pulse generator) the output of which is used to strobe the 48 sense amplifiers in that group. In so doing the inherent time variations in signal output due to word and bit location within the array, degradation of the interrogate pulse rise time, and variations in decoding time are automatically compensated for. Figure 10 also shows a function block called "S" clock generator. The pulse from this generator, which is also derived from the array, is used to set those output register flip flops which do not receive a reset input from the "T" gate. This technique permits the use of simple one input or "D" flip flops in the output register.

The final operation which occurs in NDRO is selection, by the two most significant address bits, of the proper group of 48 sense amplifiers whose strobed outputs are to establish the state of the memory output register. This selection is accomplished by permitting only one of the four "T" generators to be activated at any time thus producing an output on only one of the four "OR" inputs to each of the memory output flip flops.

*Write Mode*

It will be recalled that the organization of the array for reading is as 256 words of 192

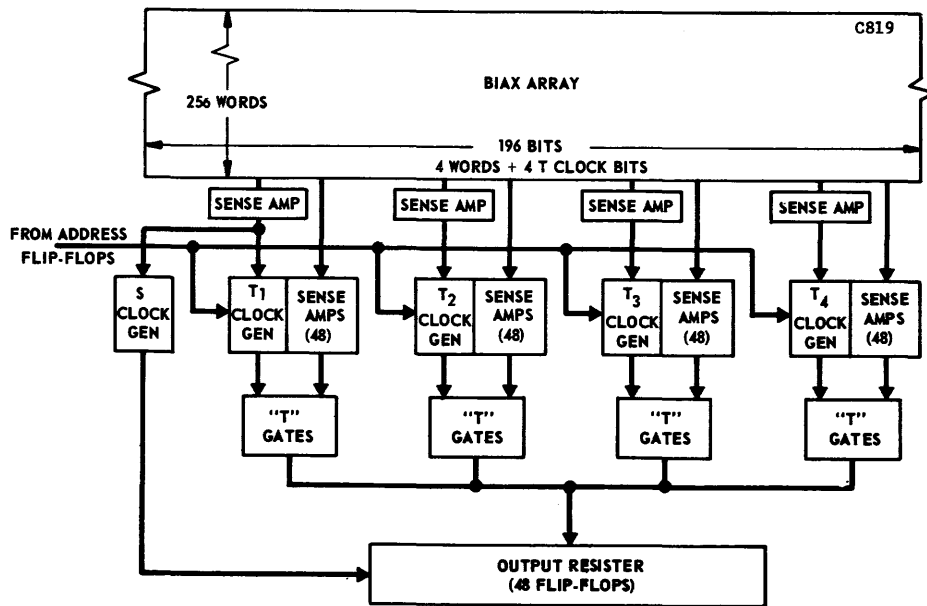


Figure 10. 10 Mc. NDRO BIAx Memory Output Block Diagram.



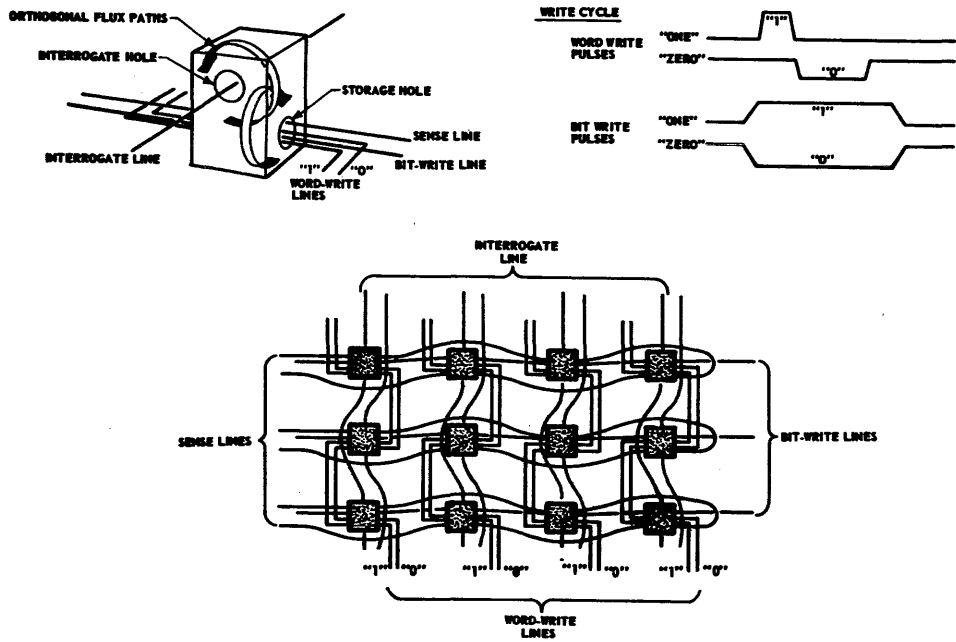


Figure 11. BIAx Array Element Orientation and Write Current Program.

bits per word. For writing however, the array is organized as 1024 words of 48 bits per word, and conventional linear select techniques are used. The elements are oriented and wired in the array such that current pulses pass in both the word and bit directions and selective writing is accomplished by the coincidence of a word oriented word write pulse and bit oriented write pulse. The orientations of the BIAx elements within the array and the

write current program are shown in Figure 11. The word write pulse currents consists of a fixed sequence of two opposite polarity word write currents, and a time-overlapping bit current whose polarity depends upon the binary state of the information to be stored. In order to select the appropriate word of the 1024 for writing, a matrix of 16 word drivers and 64 word switches, organized as shown in Figure 12, is used.

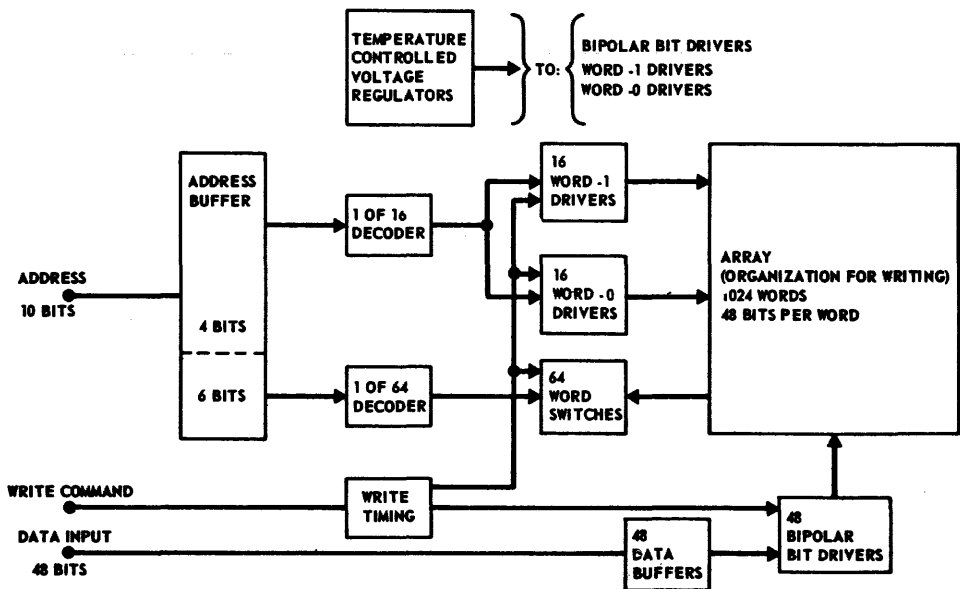


Figure 12. Memory Write System Block Diagram.

Receipt of the write command activates the write cycle by starting the fixed sequence of word current pulses and permitting the 48 bipolar bit drivers to generate currents in a direction dictated by the data inputs (Figure 11). The portion of the write cycle during which element switching occurs, if it occurs at all, is determined by the information carried by the bit current and the previous history of the element. Both the bit and word currents are temperature compensated to permit optimum switching of the elements with minimum disturb over a temperature range of 0°C to 50°C. Nominal write system operating parameters for the memory are given in Table II.

## MEMORY SYSTEM FABRICATION AND PERFORMANCE

### *Fabrication and Packaging*

The complete 1024 x 48 memory described in the preceding portion of this paper was designed, fabricated and tested. The completed memory is shown in Figure 13. In this photograph are identified the following essential parts of the memory.

#### 1. Array and Read Circuits

One of the four memory array planes is visible in Figure 13. Note that the decoder and interrogate driver circuits, located above the array, and the sense circuits and memory output circuits to the right of the array, both are mounted as physical extensions of the main array ground plane. This was done primarily to minimize propagation times and to avoid ground noise problems. Each of the four array planes is divided into eight sections as shown in the photograph. This results from the required segmenting of the sense lines (refer to Fig. 9), and because word write lines must be terminated at 48 bit intervals. The allocation

<b>TOTAL WRITE CYCLE TIME</b>	<b>5<math>\mu</math> SEC MAX</b>
<b>WRITE - READ CYCLE TIME</b>	<b>10<math>\mu</math> SEC MAX</b>
<b>WORD CURRENT AMPLITUDE FIRST ("1")</b>	<b>+ 200 ma</b>
(25°C) <b>SECOND ("0")</b>	<b>- 200 ma</b>
<b>BIT CURRENT AMPLITUDE</b>	<b><math>\pm</math> 95 ma</b>
(25°C)	

Table II. Write System Parameters.

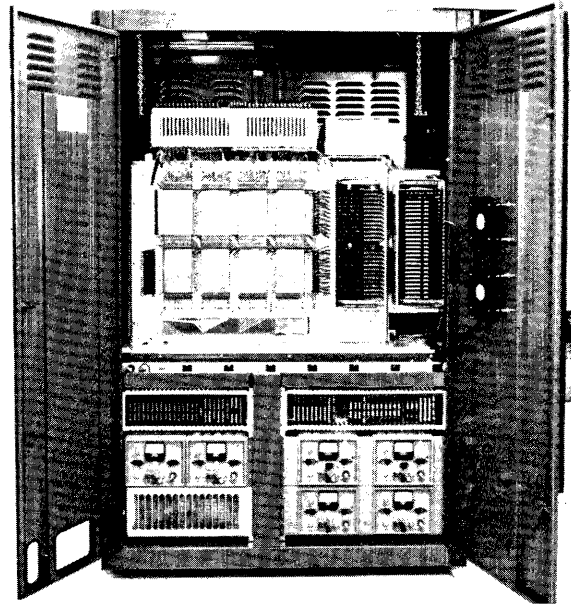


Figure 13. 10 Mc. NDRO BIAX Memory.

of spare word and bit lines is made so that each of the eight sections contains two word spares and two bit spares and a spare for the "T" line. Therefore each section contains 34 x 52 or 1768 elements. Since each section is identical, each array plane contains 14,144 BIAX elements and the entire memory array consists of 56,576 elements. Figure 14 shows

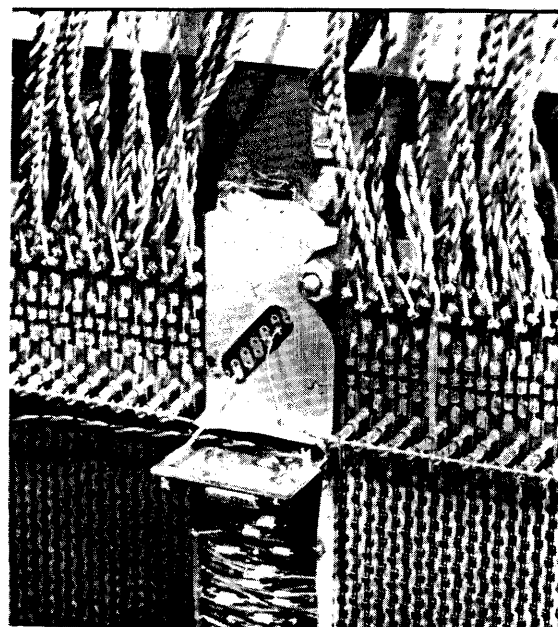


Figure 14. Detailed View of 10 Mc. NDRO Array.

a detailed view of one section of the array. From this photograph can be seen the dual interrogate lines, offset to permit straight wire looming. The element-to-element spacing in both the horizontal and vertical directions is 0.125 inches. A shielding mask can be seen, positioned between the holes of the BIA $\bar{X}$  element forming the ground plane for the interrogate transmission line. In the lower center of Figure 14 in the gap between the shielding masks are the sense lines with their isolation resistors, and the bit lines. Seen near the top of the picture are the twisted pairs which connect to the interrogate drivers, the word write lines and word select diodes.

## 2. Write Circuits

The write circuits used in the memory can be seen in Fig. 13 mounted in two card racks below the array. These circuits are of conventional design and are the same type of circuits used in other BIA $\bar{X}$  memory systems.

## 3. Power Supplies and Cooling Fans

Power supplies and blowers occupy the lower regions of the memory cabinet, and are of standard off-the-shelf variety. All power supplies have voltage regulation of 0.1% or better and are current limited to provide protection to the memory circuits.

### *Memory System Performance*

The 10Mc. NDRO memory has been extensively tested to determine its performance characteristics. Figures 15 and 16 show waveforms at various locations in the memory for NDRO operation. Figures 15A through F

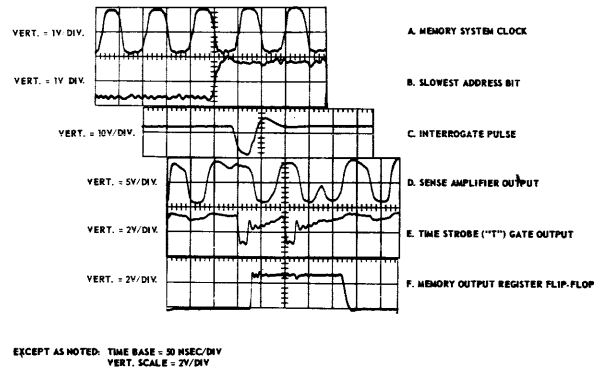


Figure 15. 10 Mc. NDRO BIA $\bar{X}$  Memory Read Cycle Timing Waveforms.

show the read operation from the beginning of decoding, through the decoder and interrogate driver selection, and through the sense amplifier and time strobing, and to the memory output register. Figure 16 shows detailed photographs of read circuit waveforms. In Figures 16C and D are shown the access time measurement from the 50% point of the system clock (negative going) to the response of the memory output flip flop. This access time represents the longest access time for any word or bit in the memory.

The memory also underwent considerable testing to determine its operating reliability under various conditions of patterns and cycle rates for both the NDRO and write/read modes. To facilitate the testing, a memory exerciser which was capable of generating an almost unlimited number of bit and word patterns and error checking each pattern in both NDRO and write/read, was employed. By

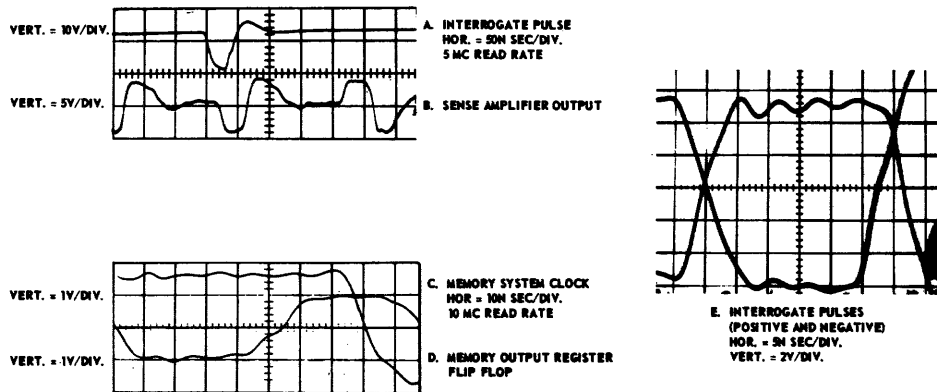


Figure 16. 10 Mc. NDRO BIA $\bar{X}$  Memory Read Circuit Waveforms.

utilizing this exerciser, errors from any origin caused the equipment to stop and indicate the word and bit location of the error. During the equipment checkout phase, tests representing voltage and write current variations, as well as worst-case patterns and cycle times, were run as a matter of course. To demonstrate that reliable system operation was being obtained, each pattern was run for a ten minute period of time, resulting in a total of  $3 \cdot 10^{11}$  bits having been error checked. In this time, each bit in the memory was error checked  $6 \cdot 10^6$  times, and because of the memory organization, had actually been interrogated  $24 \cdot 10^6$  times. As an acceptance test for the memory, fourteen patterns were each run for the required ten minute period, representing a total of approximately  $42 \cdot 10^{11}$  error checks of stored information. Each pattern was also run for the write-read-error check mode for a thirty minute period with a read-after-write error check at a write-read cycle time of 9 microseconds. The entire acceptance test procedure involved approximately 40 hours of error free system operating time.

#### FUTURE AREAS OF INVESTIGATION

Although work has been completed on the memory system described in this paper, many extensions of the techniques are possible. Below are described a few of the more promising approaches and areas in which further work is being done.

##### *Variations in Word Organization*

Although the memory described in this paper is organized as 1024 words of 48 bits per word, since the array is organized for reading as 256 words of 192 bits per word, many variations in effective memory organization are possible. For example, a read organization of 256 words of 192 bits per word could be readily achieved with minimum modifications. Similarly, word lengths between 48 and 192 bits can be achieved. In summary, many combinations of word lengths and bits per word can be realized for NDRO operation with the existing array design, as long as the total storage capacity is not exceeded, although with the present array, writing must still be performed on a 48 bit per word basis.

##### *Access Time Reduction*

In Figure 2 at the beginning of this paper, it was noted that the BIAX array contributed only about 10 nanoseconds to the 80 nanosecond typical access time. In view of this it appears quite feasible to reduce the access time substantially by appropriate circuit design effort, as the NDRO operation of the BIAX element is not a limiting factor.

##### *Faster Read Cycle Times*

In the same way that the access time can be reduced, it is quite possible to increase the reading rate to 20 Mc. or more while still using the same array and system organization principles.

##### *Increased Storage Capacity*

The present memory capacity of 1024 words in no way represents the practical limit for this fast NDRO technique. It seems quite likely that word capacities of two or four times the present memory could be achieved with perhaps a 120 nanosecond access time.

##### *Reduced System Volume*

No effort was made to minimize the physical size of the present memory, rather it was designed specifically for physical access to the array. By appropriately folding the array and repackaging the circuits, the physical size of a system should be consistent with other core memories of similar capacities.

##### *Airborne Applications*

The BIAX element and its low power non-destructive readout properties are particularly well suited for airborne applications. For this reason, BIAX elements for use at temperatures from  $-55^{\circ}\text{C}$  to  $+100^{\circ}\text{C}$  have been developed by Aeronutronic and are being employed in various systems. The techniques used in the 10 Mc. NDRO BIAX memory can be readily applied to this type of element to produce very fast NDRO operation over a wide range of temperature.

##### *MicroBIAX Applications*

The BIAX element used in the 10 Mc. NDRO memory employed elements developed before the start of the memory project. A major in-house program is now underway to develop a MicroBIAX element having outside dimensions of 30 x 30 x 50 mils. These elements offer

greatly improved characteristics, particularly for write cycles of 1–2  $\mu$  sec. In addition, faster NDRO operation, better performance over wide temperature ranges, simpler array wiring configurations, as well as the obvious size advantages are offered by the MicroBIAX element. The potential applications for this class of new elements is almost unlimited, and it is expected that MicroBIAX elements will be employed in most of the new memory systems which are developed in the future.

#### ACKNOWLEDGMENT

This work was sponsored in part by the Department of Defense. Many people in addition to the authors have contributed to the success of the program, but in particular the efforts of C. M. Sciandra in preparation of elements and C. L. Cantor and M. J. VanZanten in the design and testing of the system are greatly appreciated.

#### REFERENCES

1. J. A. RAJCHMAN: "Magnetic Memories; Capabilities and Limitations", *Computer Design*, September 1963.
2. C. L. WANLASS and S. D. WANLASS, "BIAX High Speed Computer Element", WES-CON, 1959.
3. DUDLEY A. BUCK and WERNER I. FRANK, "Nondestructive Sensing of Magnetic Cores", AIEE Technical Paper 53-409, October 1953.
4. ATHANASIOS PAPOULIS, "The Nondestructive Read-Out of Magnetic Cores", *Proceedings of the IRE*, Vol. 42, pp. 1283–1288, August 1954.
5. U. F. GIANOLA and D. B. JAMES, "Ferromagnetic Coupling between Crossed Coils", *Journal of Applied Physics*, Vol. 27, No. 5, pp. 608–609, June 1956.
6. JACOB MILLMAN and HERBERT TAUB, *Pulse and Digital Circuits* (Mc Graw-Hill Book Co., Inc., New York, 1956), Chap. 10, pp. 291–295.
7. MONTGOMERY PHISTER, JR., *Logical Design of Digital Computers*, (John Wiley & Sons, Inc., New York, 1958), Chap. 5, p. 126.

# ASSOCIATIVE MEMORY SYSTEM IMPLEMENTATION AND CHARACTERISTICS

*J. E. McAteer and J. A. Capobianco*  
*Hughes Aircraft Company, Ground Systems Group*  
*Fullerton, California*  
*and*  
*R. L. Koppel*  
*Autonetics Division of North American Aviation*  
*Anaheim, California*

## 1. INTRODUCTION

The implementation of a new system utilizing state-of-the-art technologies requires a careful engineering evaluation of all parameters affecting such a design. In particular, when new system concepts are needed and the available devices for mechanization have been designed for a different class of system, the problems become much more severe. Such is the case with Associative Memory (AM) systems where an entirely new system organizational concept places exacting requirements on the existing technology of information storage, as is evidenced by the many techniques which have been proposed for implementation.<sup>2-8</sup>

It has been determined through study and evaluation of storage media that the BIAX\* element,<sup>1</sup> a multiaperture ferrite core, possesses the most desirable characteristics for implementing an associative memory today. The utilization of the BIAX element in the mechanization of an AM is not limited to one configuration. The repertoire of possible methods consists of one-BIAX-per-bit and two-BIAX-per-bit schemes, and, within each of these areas, there exist different ways of utilization. The

choice of the mechanization methods is dependent on the application and would be a result of detailed system analysis and tradeoff studies. The first part of this paper details the mechanization techniques of an associative memory with the BIAX element. In particular, a new mode of use of the BIAX element is presented which enables extremely fast search times to be realized. The number of functions which an AM can perform are many and varied. These functions may be broadly classified in the following way:

1. Search Functions
2. Write Functions
3. Readout Functions

The functions which are provided in a given system are, as mentioned, dependent on the application. In addition, the methods of performing some of these functions, in particular the searching types, are dependent on the speed requirements. These in turn will, to some extent, determine the mechanization method chosen. The second part of this paper details the various functional characteristics an associative memory might have. A chart is presented which delineates the pertinent characteristics as a function of the mechanization technique.

\* Registered Trademark, Philco Corp.

The last part of the paper shows the results obtained from a demonstration model of an AM which utilizes the BIAx in the new mode of operation mentioned above.

## 2. BIAx IMPLEMENTATION OF ASSOCIATIVE MEMORIES

### 2.1.0 Normal BIAx Operation, Using Two-BIAx-Per-Bit

The BIAx is a rectangular block of ferrite having two orthogonal holes: the storage (information) hole and interrogate hole are as shown in Figure 1. Also shown in the figure are the read and write waveshapes and the readout signal produced by the BIAx element when operating in the normal mode. Note that the sense signal is bipolar and occurs during both the rise and fall time of the interrogate current and that the phase of the signal is information dependent. The sense signal is caused by a domain rotation phenomenon which results from the interaction of storage and interrogate hole flux, in the material between the holes, during interrogation. An AM implemented with the normal mode of the BIAx operation requires a serial-by-bit interrogation to prevent possible cancellation of pulses on the word-oriented sense lines.

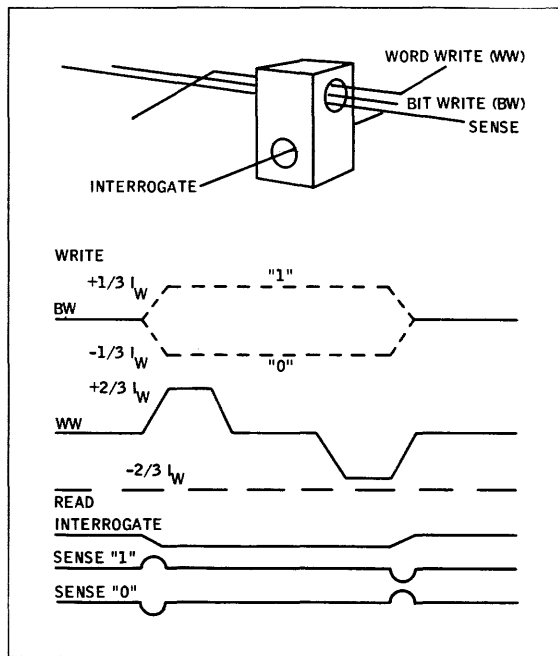


Figure 1. Conventional BIAx Operation.

Figure 2 depicts the technique used in the two-BIAx-per-bit method. The normal and complement of each word are stored. In order to decrease the required search time, the interrogate currents are staggered by an amount equal to or greater than the rise time of the current and left on until the last bit has been searched. This prevents the output signals produced by the trailing edge of the interrogate current from interfering with sense signals produced by subsequent interrogate pulses.

If the sense signal polarities are as shown in Figure 1, and if the normal bit is searched when looking for 0 at a bit position and the complement bit is searched when looking for a 1, then the input to the sense amplifier will consist of a series of negative pulses for a matching word. This is due to the fact that all elements interrogated would be in the 0 state. Should a mismatch occur at a bit position, a positive pulse will occur on the sense line. For example, in Figure 2, drivers C<sub>1</sub>, N<sub>2</sub>, N<sub>3</sub>, and C<sub>4</sub> would be turned on. In Word No. 1, several elements in the 1 state are interrogated resulting in a mismatch (positive) signal, while in Word No. 2 all elements interrogated contain 0 and only negative pulses occur on the sense line.

### 2.2.0 Operation of the BIAx in the Hughes Unipolar Mode

#### 2.2.1 Description of Operation

In the course of the mechanization studies, a technique for using the BIAx which greatly enhances the search speed has been invented. This new mode of operation results in a signal for a

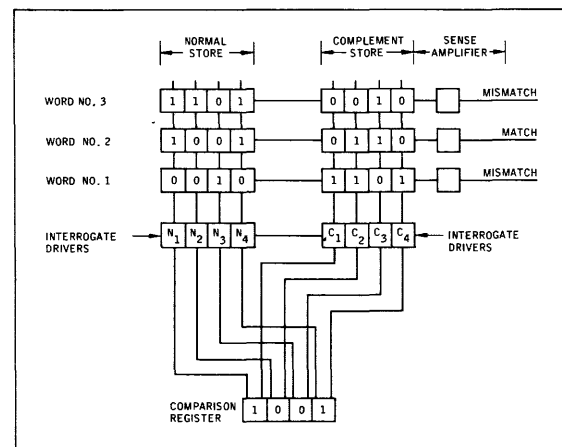


Figure 2. Typical Search Operation.

stored 1 and no signal for a stored 0, with a very high element signal-to-noise ratio. Thus, unipolar rather than the conventional bipolar operation is obtained. This technique allows parallel-by-bit interrogation. Thus, the search time is not directly proportional to the number of bits per word, as in the serial-by-bit approach, but is proportional to the number of bits per word divided by the number of elements interrogated simultaneously.

Using the same criteria for selecting the normal or complement driver as before, it can be seen that for the matching word, no signal will occur on the sense line since all 0's are being interrogated. Therefore, if the elements are interrogated simultaneously, only 0 noise buildup will be seen. For a word which mismatches (interrogation of an element in the 1 state), a large output signal will result.

The number of elements which may be interrogated simultaneously is a function of the signal-to-noise ratio of the elements. On a labora-

tory basis twenty-bit words have been interrogated with resultant word signal-to-noise ratios (twenty 0 signals versus one 1 signal) greater than 3:1 (see Section 7). In a practical system, the number interrogated simultaneously would be smaller due to environmental conditions and circuit tolerances. However, since the decrease in search time is directly related to the number of elements interrogated simultaneously, dramatic improvements result.

The method of obtaining this mode of operation is shown in Figure 3. The element is first written to the 1 state by a current pulse large enough to saturate the storage hole (Figure 3B). This pulse is then followed by a smaller pulse of the opposite polarity which is the Word Write 0 current. If a 0 is to be written then, in coincidence with the Word Write 0 current, a current pulse (Bit Write 0) is produced in the interrogate hole and the flux around the storage hole is reduced to a very small value. If a 1 is to be written, the Bit Write 0 pulse does not

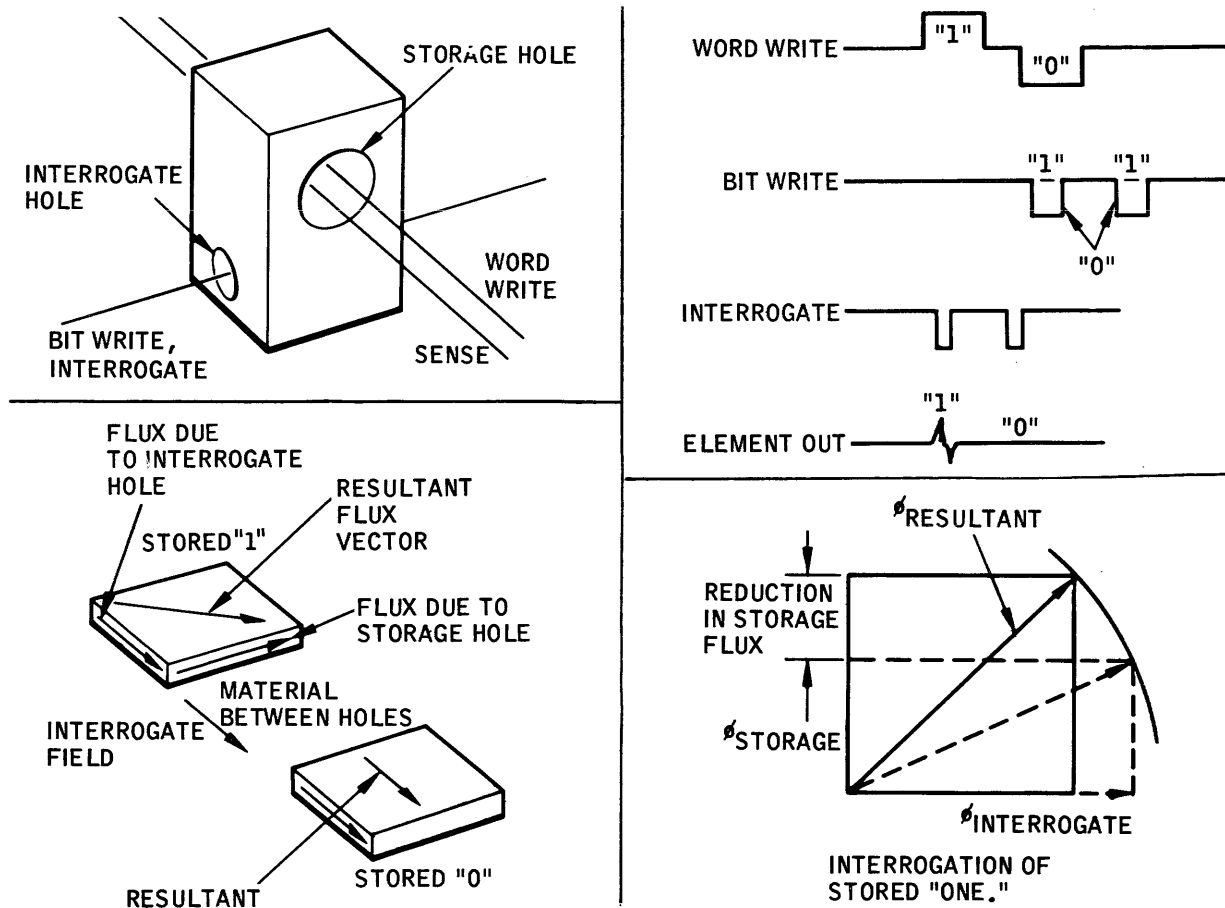


Figure 3. BIAx Element Operation in the Unipolar Mode.



occur and the storage hole flux remains in a saturated condition. Part C of Figure 3 illustrates the technique by showing what occurs in the common volume of material between the two holes.

This technique has one main disadvantage: Where writing into fields of words is desired, the disturb characteristics of the element in the 0 state result in a lowered signal-to-noise ratio. This is due to the fact that the flux around the storage hole of the unselected bits will "creep" to a higher value due to the word oriented disturb currents and thus produce a larger output for the 0 state than is desired. However, the method has many advantages. One which has been mentioned is that of decreasing the search time. This reduction in the basic search time can be traded for hardware cost by permitting time-sharing of sense amplifiers and thus reducing the number of circuits required. In addition, it can be seen from Figure 3A that all windings are orthogonal and thus the array noise problem is reduced and, since no woven windings are needed, the array fabrication is extremely simple.

### 2.2.2 Ternary-State Reading in the Hughes Unipolar Mode

A significant aspect of the unipolar mode of operation is that by reversing the sense winding between the normal and complement words and using serial-by-bit interrogation, a ternary output results. That is, if a bit matches the search, then no output results; if a bit mismatches, then the output can be positive or negative dependent upon whether the normal or complement bit was interrogated. In this manner it is possible to classify all words at one time as less than, greater than, or equal to the search word.

This can be explained by Figure 2. Assume that an element in the 1 state (signal output during interrogation) in the normal word produces a positive output due to the reversed sense winding. If the same criteria are used as before for selecting interrogate drivers, Word No. 2 will again produce no signal since it exactly matches the search word, and thus all elements interrogated are in the 0 state. However, Word No. 1 mismatches in the first bit position and, since the complement bit is in-

terrogated, will produce a negative output indicating that Word No. 1 is less than the search word. Word No. 3 agrees in the first bit position with the search word and thus will produce no output for that interrogation. However, in the second bit position a mismatch occurs and, since the normal bit is interrogated, a positive pulse occurs indicating that Word No. 3 is greater than the search word. Thus, (1) if a positive pulse appears on the sense line first, that word is greater than the input word; (2) if a negative pulse appears on the sense line first, that word is less than the word; and (3) no pulse on the sense line indicates an exact match.

The technique described is quite significant in that the search time now is independent of the search type for these three searches and, with a tristable sense circuit, all words are classified simultaneously. With the conventional mode of operation described previously, only two sense signals may be derived: positive and negative. Therefore, in order to accomplish limit type searches, a stepping algorithm (see the section on Associative Memory Search Functions) which alters the search word between steps must be used or logic in the sense amplifier is necessary to determine if the first mismatch occurs when a 1 or a 0 is searched for (if the first mismatch occurs when a 1 is being searched for the search word is obviously larger than the stored word and vice versa).

### 2.3.0 Operation Using One BIAx Per Bit

Mechanizing an AM with one BIAx per bit requires a serial-by-bit interrogation. However, the method of accomplishing this interrogation can take several forms.

One possible way to interrogate is to ripple through the bits serially and gate the sense signal logically at each amplifier against the information in the corresponding bit position of the input word. In this manner, mismatches can be detected and, if a tristable sense circuit is available, the limit searches (LESS THAN and GREATER THAN) can be directly implemented without use of an algorithm.

Another way to interrogate involves an interrogate-priming cycle and does not require logic in the sense amplifier. Referring to Figure 1, it can be seen that the sense signals for a 1 and

0 are out of phase for the same interrogation pulse. If the sense output were examined during the rise time of the interrogate when searching for a 1 and during the fall time when searching for a 0, it can be seen that, if a match occurs, the pulses on the sense line would all be positive. If a mismatch occurs (for example, examining the output of an element in the 1 state during the fall time of the interrogate pulse); then a negative sense signal occurs.

The above process can be implemented in a straightforward manner by merely turning on (priming) all interrogate drivers which are to search for 0's before rippling through the interrogate cycle and turning them off at the proper time during the interrogate cycle. Figure 4 depicts the procedure for accomplishing the interrogation.  $I_1$  and  $I_3$  are turned on during the priming period since they are to search for 0 as indicated by the contents of the Data Register. In Word No. 1, the corresponding bit positions contain a 0 and 1 respectively, and hence no output will appear on Sense Line 1 during the priming period because of cancellation. However, Word No. 2 contains 0 in both positions and therefore a double amplitude negative pulse will appear on Sense Line 2. During the interrogate period, a negative pulse appears on Sense Line 1 indicating that the word mismatches, while Sense Line 2 has all positive pulses which indicates a matching condition.

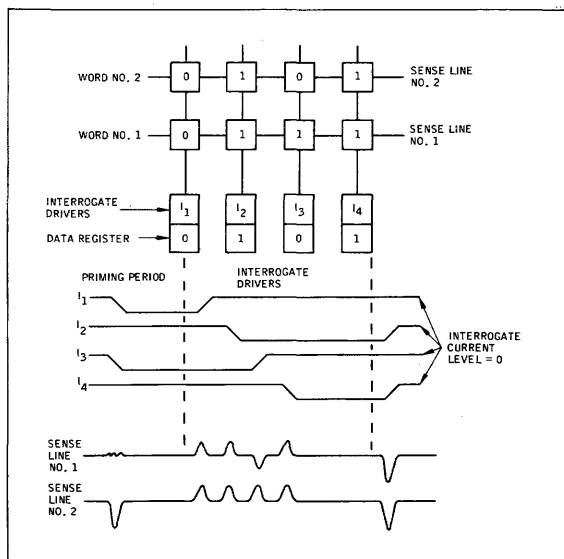


Figure 4. Interrogation Using One-BIAX-Per-Bit.

The method of implementation described here is quite straightforward and is such that conventional random access BIAx array windings with some modifications can be used. In addition, conventional memory circuitry, with the exception of the word-oriented sense circuits, can be used, and data readout is provided with relative ease. This technique would, however, be somewhat slower than the parallel-by-bit two-BIAX-per-bit scheme. Since the BIAx is operated in its normal mode, the disturb characteristics and writing mode are such that alteration of arbitrary fields within a word can be provided.

### 3. ASSOCIATIVE MEMORY SEARCH FUNCTIONS

#### 3.1.0 EXACT-MATCH Search

There are a variety of search types which can be implemented in an associative memory.<sup>9, 12</sup> These searches can be performed on an entire data word or on specified fields. The selection of fields is accomplished by having the ability to mask the data word. That is, any bit of the comparison word can be masked to a "don't care" state, and only those bits not masked will participate in the search. Thus, there is inherently a ternary search characteristic (1, 0, don't care) which may be taken advantage of in some cases to decrease the search time. A brief description of search types follows:

The most commonly used search operation is the EXACT-MATCH search. This search, as the name implies, would locate all words in memory which have a one-to-one correspondence with the bits of the search word. That is, any word in memory which mismatches the search word in one or more bit positions does not satisfy the search criterion. The search time is proportional to the number of bits in the word with the exception of the parallel-by-bit techniques.

#### 3.2.0 Limit-Type Searches

Under this category are included GREATER THAN, GREATER THAN OR EQUAL TO, LESS THAN, and LESS THAN OR EQUAL TO searches. The functions of these search types are fairly obvious. The time involved in performing these searches is dependent upon the method of mechanization. In the two-

BIAX-per-bit scheme, with the ternary output described previously, the search time is the same as an EXACT-MATCH search and no logic gating is required in the sense circuitry. In order for the other techniques to have a comparable search time, logical gating is necessary in the sense circuitry at each bit interrogate time.

Another technique for accomplishing the limit-type searches is to have an algorithm which alters the search word and then looks for exact matches at each step. In its simplest form this would consist of incrementing or decrementing a counter for each step and performing an EXACT MATCH search. However, by taking advantage of the ternary characteristics of the interrogation (1, 0, don't care), it is possible to reduce the number of steps required. With this method the maximum number of steps required is equal to the length of the field participating in the search and, on the average, will be one-half the length of the field. For example, if a 12-bit field were being used in a LESS THAN OR EQUAL search the maximum number of steps required to perform the search is 12 (contrasted to a maximum of 4096 in the simpler counter approach). Thus, if any word matches at one of the steps, it satisfies the search criterion. This method, while not requiring logical gating of the sense circuits, results in a significant increase in component count if the fields are not restricted. The process of finding all words in memory which lie between some specified bounds can be accomplished by the successive application of the GREATER THAN OR EQUAL TO and LESS THAN OR EQUAL TO searches to the same field with a change of the search word. A LESS THAN OR EQUAL TO search is performed on the upper bound search word which therefore eliminates all words greater than the upper bound. The GREATER THAN OR EQUAL TO search on the lower bound search word then leaves those words lying within the bounds indicating a matching condition. A somewhat more efficient algorithm can be implemented if the upper and lower bounds are available simultaneously.

### 3.3.0 Pattern Recognition

Another useful function which can be provided in an associative memory is a form of

pattern recognition. As an example, consider the case where it is desired to compare incoming patterns with stored patterns in the associative memory.

An incoming pattern is normalized, sampled, and quantized at set intervals. These quantized samples then become the keys with which the search is conducted. Since exact pattern matches are impractical, there are two words stored in memory (two-BIAX-per-bit mechanization is used) for a single pattern. The word outputs, which indicate match or mismatch, are OR'd together and "don't care" bits are written into the words in memory. That is, if a bit position is in the "don't care" state, no response will be obtained from the bit during interrogation for a 1 or a 0. This has the effect of performing a BETWEEN LIMITS search in memory and thus effectively establishes an envelope about the desired pattern. For example, if there are 32 quantization levels and one sample point has the value of 23, then the words stored in memory might be 101XX and 110XX (where X indicates a "don't care" bit) thus allowing a match indication for that point if the incoming waveform has a value between 20 and 27. Thus, the tolerance allowable is accounted for in the memory and is subject to control. This is illustrated in Figure 5.

### 3.4.0 Supplementary Search Operations

Ordered Retrieval—In some problems it is desired to retrieve information in an ordered manner. In a conventional system this can be

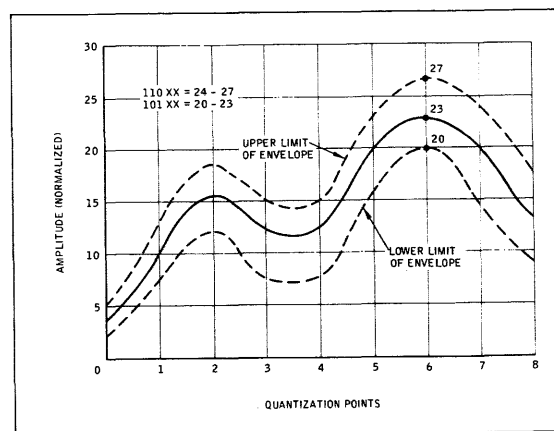


Figure 5. Tolerance Envelope Used in Pattern Recognition.

a very time-consuming process. Using the ternary characteristics of the associative memory, much more efficient ordering is possible.<sup>10,11,13</sup>

**Minimum and Maximum Searches**—In some applications it is desired to find the word in memory which has the minimum value, within the field, with respect to all other words in memory. The algorithm for accomplishing this is that which would be used for ordered retrieval. The algorithm would terminate when the first single response occurs.

The maximum search would use the same algorithm as for inverse ordered retrieval.

**Nearest Neighbor**—The capability of determining the nearest numerical neighbor above or below the value of the selected key can be implemented by the use of an ordered retrieval algorithm. This is accomplished by using either normal or inverse ordered retrieval starting from the initial value of the key. A more complex algorithm can be implemented to obtain the nearest neighbor on either side if it is desired.

**Composite Searches**—In some instances it is desirable to perform searches on different keys and to specify a logical relationship between the separate key searches. Accordingly, only those words which satisfy the logical relationship and key searches are retained. For example, if there are five keys, A, B, C, D, and E, it might be desired to perform an EXACT MATCH on key A, GREATER THAN OR EQUAL TO on key B, BETWEEN LIMITS on key C, and LESS THAN OR EQUAL TO on keys D and E. In addition, logical relationships such as ABCDE,  $ABC+DE$ , may be required. This type of search can be very useful in a variety of applications. The possibility of providing a match indication if a portion of the search keys match could also prove useful.

#### 4. AM WRITING FUNCTIONS

As with the search functions there are a number of different writing functions which may be provided with an AM. As might be expected some of these functions are identical with the normal writing modes encountered in conventional memories. However, there are also those modes which are peculiar to the AM organization and which add to the power of the system

and extend its range of usefulness. The writing functions which are provided in a system would be strongly application dependent.

**Sequential Load**—In some applications, blocks of data may be transferred to the associative memory for use in subsequent searches. Sequential load starting from the first word location is then useful as it allows the words to be loaded very rapidly by minimizing the controls necessary while retaining a spatial relationship with respect to the source store. In a partial load, word locations not written into can be prohibited from responding to subsequent searches.

**Random Load**—Another loading feature which is often useful is the random load. This is the same as for a conventional random access memory and requires that the physical location (address) to be written into be specified.

**Load First Empty Location**—An associative memory can be implemented to keep track of its own empty locations, such that when a word is to be entered, it is automatically written into the first empty location. In a memory where the retrieval time is location dependent, this is very effective since all data is held at the "front" of the memory, thus minimizing access time. This data-packing feature can be very useful.

**Write "Don't Care" Bit**—Masking within the data word in the associative memory itself can be accomplished by writing a bit to the "don't care" state. With this technique bounds can be stored in the memory as described previously. This is one of the more interesting features which should find great utility. The two-BIAX-per-bit schemes are, at present, the only techniques which can be used to accomplish this.

**Field Alteration**—The ability to alter a single bit or field of all words or selected words as a consequence of the result of a search is another writing characteristic which might be provided. This feature is particularly useful when using the memory as an aid to parallel computation. The element must be operated in the conventional mode to implement this feature. This could also be termed "writing through a mask."

**Memory Partitioning**—It is possible to partition an associative memory so that there effec-

tively exists "micro-associative memories" within the main associative memory. This feature is useful when several types of data are stored and the access time needs to be kept to a minimum. Thus, only that portion of the memory containing the data to be interrogated is accessed and the non-pertinent data for that search is bypassed. This of course assumes that the memory contains multiple planes and the normal search process consists of a sequential search of the planes (all words in a plane are of course searched in parallel).

## 5. AM READOUT FUNCTIONS

As with the other characteristics of an associative memory, there are a number of different types of readout possible. The type of readout necessary is, of course, dependent upon the application.

**Address Readout**—In an application where the key or keys are well defined, the use of an associative memory with a conventional random access memory may be advantageous. In this mode, a block of keys is transferred from the conventional memory to the associative memory in a specified sequence so that the physical location of the keys in the associative memory is spatially linked with data stored in the conventional memory. Upon searching the associative memory, the output indicates the addresses of the words in the random access memory which satisfy the applied search criteria. This mode may be particularly useful where the ratio of the search key to the remaining data word is small, since at present the associative memory is expensive relative to conventional random access memory.

**Data Readout**—The ability to read out the contents of an associative memory is another feature which is useful. The flexibility that this allows in a system can be significant, since any portion of the data word may be searched on, and the data word itself, or perhaps the portion of the data word not searched, can be read out.

**Multiple Match Resolution**—In any search the possibility of more than one word matching the applied criteria has to be contended with. The ability to retrieve all matching words is, in most cases, a necessity. This is usually accomplished

by retrieving them sequentially through a commutating network. An efficient design of the commutating network is necessary since it can be an important factor in the retrieval time.

**Yes-No**—In some applications a decision is made regarding the next course of action after interrogation of the memory, based only on information as to whether or not the search has been matched. The Yes-No operation is a relatively easy feature to provide.

**Count Matches**—When the memory is searched there is a possibility, dependent on the application, that a significant portion of the memory could respond to the search. In such cases, an indication of the number of matches which exists may be wanted before output occurs. If the dump is excessive, then the search may be refined to reduce the number of response.

## 6. CHARACTERISTICS OF AM SYSTEMS

The foregoing has been a brief description of some of the more salient features of AM mechanization techniques and functions. Table I lists the techniques mentioned and shows the relative performance of the mechanization schemes. It can readily be seen from the discussion above and Table I that an absolute comparison of techniques is not practical. For an absolute comparison, detailed knowledge of the system application would be required, so that the various factors and tradeoffs could be intelligently evaluated.

None of the schemes shown in Table I requires logical gating of the sense circuits for performing limit-type searches. Thus, for Schemes 11, 3, and 4, a stepping algorithm (similar to that in Ref. 11) is used for these search types and therefore the limit search time is a function of the length of the field used in the search. However, by providing logic in the sense circuit, the limit search time for these three schemes becomes proportional to  $M$ , the number of bits per word. The merits of providing this mode would be ascertained from the total system analysis.

In the equations for the relative limit search time, the first term represents the time required for storage of intermediate results (considering one unit of time as the time between

Table I. Associative Memory System Characteristics.

<u>Scheme</u>	<u>Mechanization</u>	<u>Relative Exact Match Search Time</u>	<u>Relative Limit Search Time</u>	<u>Data Writing</u>	<u>Data Reading (Additional Array Requirements)</u>	<u>Restriction On Fields For Limit Search*</u>	<u>Write: "Don't Care" Bit Available</u>
1	Two-BIAX-per-bit Signal-no signal Binary sense output	$\frac{M}{k}$	$\frac{3F}{2} + \frac{F^2}{2k}$ (ave) $3F + \frac{F^2}{k}$ (max)	Whole word only.	Additional array windings required.	Fixed fields only.	Yes
2	Two-BIAX-per-bit Signal-no signal Ternary sense output	M	M	Whole word only.	Additional array windings required.	No restrictions (any combination of bits selected by mask permissible)	Yes
3	Two-BIAX-per-bit Signal-signal Binary sense output	M	$\frac{3F}{2} + \frac{F^2}{2}$ (ave) $3F + F^2$ (max)	Unrestricted as to location and number of bits	No new windings required.	Fixed fields only	Yes
4	One-BIAX-per-bit Signal-signal Binary sense output	M	$3F + \frac{F^2}{2}$ (ave) $6F + F^2$ (max)	Unrestricted as to location and number of bits	No new windings required	Fixed fields only	No

LEGEND: M = Number of bits per word  
k = Number of bits per word interrogated simultaneously  
F = Field length used in limit search

\*See text.

successive interrogations). Thus, since the average number of steps (field interrogations) in the incrementing algorithm is one-half the field length, the number of storage cycles required is  $F/2$  and, in the type of system being considered, the storage cycle is about three times the "ripple" time, hence the term  $3F/2$  in Scheme 1. The second term represents the total "ripple" interrogate time. Since again  $F/2$  steps are required on the average, and there are  $F/k$  ripple times, the total is  $F^2/2k$ . Of course if  $k = 1$

(serial-by-bit interrogation) there results the equations shown in Scheme 3. In Scheme 4 the first term is increased due to the priming cycle and the need for sense amplifier recovery due to the priming cycle.

The table attempts to compare systems of approximately equal logical complexity, hence the restriction on the fields in the limit searches. Obviously, it is logically possible to have completely variable fields for the limit searches in

all Schemes at the expense of additional components (which can be quite significant in number).

## 7. THE ASSOCIATIVE MEMORY MODEL

Of the techniques for mechanization described earlier, the only one which departs significantly from the conventional use of the BIAX is that which produces signal-no signal operation. For this reason it was decided to verify the approach experimentally by the construction of a model which utilized this new mode of operation.

The block diagram of the model is shown in Figure 6. The array portion of the system consists of 16 words of 10 bits each, and one word of 20 bits for purposes of signal-to-noise experiments. Since both the normal and complement

of information are stored, there are therefore 340 bits in the array (the complement of the 20-bit word is not needed for the experiments for which this word is intended).

The block diagram of Figure 6 is not complete in every detail but shows the more pertinent features. The Data and Mask Registers consist of a bank of 10 manual switches each with the provision for patching the address counter into the Data Register to permit dynamic search and write operations. The model is also capable of performing "write" and "read" in a single step process by means of push button control. The search timing can be controlled to allow serial-by-bit operation or parallel-by-bit with  $k$  from two to ten.

Figure 7 shows three photographs of the demonstration model. The top figure is inter-

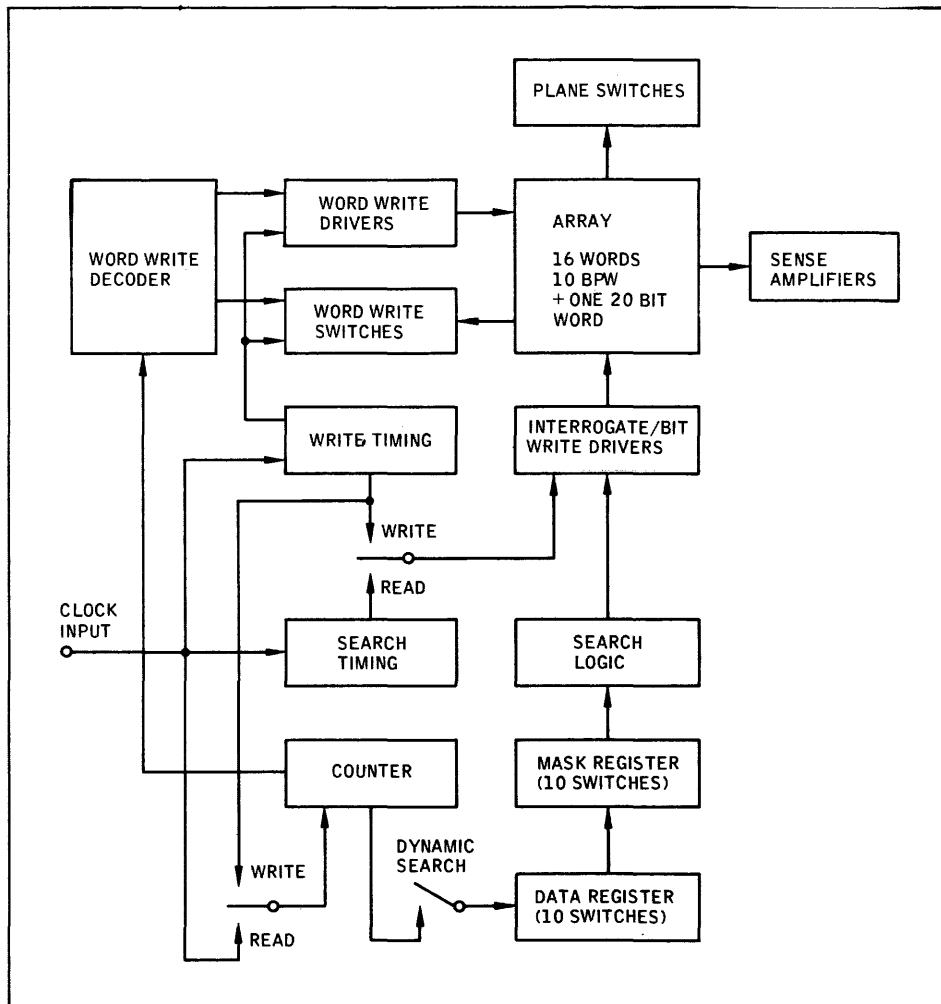


Figure 6. Simplified Block Diagram of Model of Associative Memory.

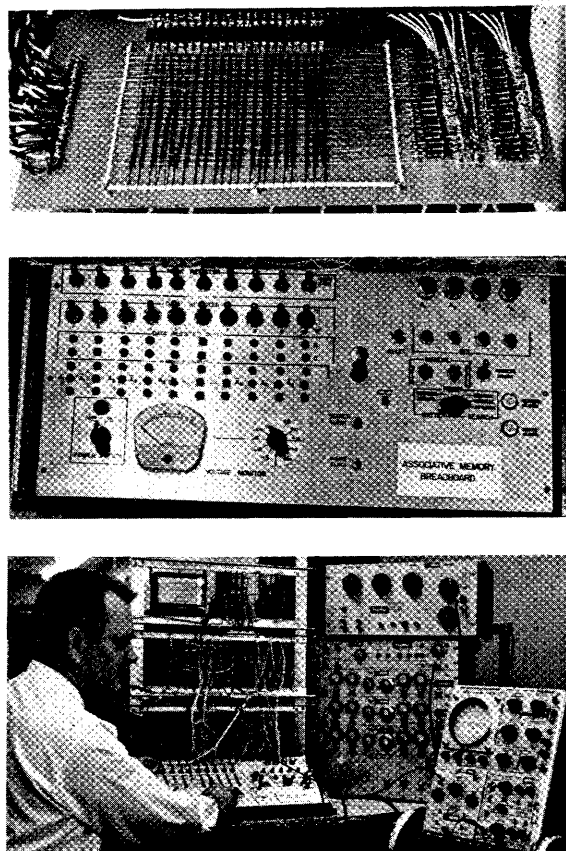


Figure 7. Photographs of Associative Memory Model.

esting in that it shows that no woven windings are necessary in the array. This suggests an array structure with all elements in contact which provides highly compact and noise-free systems. The model proved that the technique is valid and could be applied to larger systems.

Figure 8 shows waveform photographs obtained from the 20-bit evaluation word. The write program is shown in part (a); part (b) shows the interrogate current waveform and the disturbed 1 output of a single element. Part (c) shows the 0 output of a single element, which together with part (b) indicates  $S/N \geq 40$ . Part (d) shows the result of interrogating a string of 20 elements, 19 of which have stored 0's while the 20th has stored in it a 1. Part (e) shows the result of interrogating the same string of 20 elements while all are in the 0 state. Comparison of parts (d) and (e) clearly indicates a sense winding output signal-to-noise ratio of better than 3:1, which permits relatively straightforward amplitude discrimination.

The model has been operated at search clock rates of 2Mc (limited by external equipment) with simultaneous interrogation of all bits of the word ( $k = 10$ ).

## 8. CONCLUSIONS

This paper has presented several techniques for the utilization of the BIAx in an associative memory system. The techniques presented have, in some cases, significantly different operating parameters. In addition, the influence of the various techniques on the search speeds has been pointed out. From this discussion it can be seen that the number of trade-off areas which exist, and the resulting influence on system complexity and performance, make it necessary to have an intimate knowledge of ultimate system utilization in order to effect a proper associative memory design.

## ACKNOWLEDGMENTS

The work presented in this paper is the result of the contributions of many people. The authors would particularly like to acknowledge the contributions of L. H. Adamson and D. A. Savitt.

## REFERENCES

1. WANLASS, C. L., and S. D. WANLASS, "BIAx High Speed Magnetic Computer Element," WESCON Convention Record, Part 4, pp. 40-54, San Francisco, California, August 18-21, 1959.
2. KISEDA, J. R., H. E. PETERSEN, W. C. SEELBACH, and M. TEIG, "A Magnetic Associative Memory," IBM Journal of Research and Development, Vol. 5, pp. 106-121, April 1961.
3. BROWN, J. R., Jr., "A Semi-Permanent Magnetic Associative Memory and Code Converter," Special Technical Conference on Nonlinear Magnetics, Los Angeles, California, November 1961.
4. LEE, E. S., "Solid State Associative Cells," Proceedings of the Pacific Computer Conference, California Institute of Technology, March 15-16, 1963.
5. SLADE, A. E., and C. R. SMALLMAN, "Thin Film Cryotron Catalogue Memory," Sym-



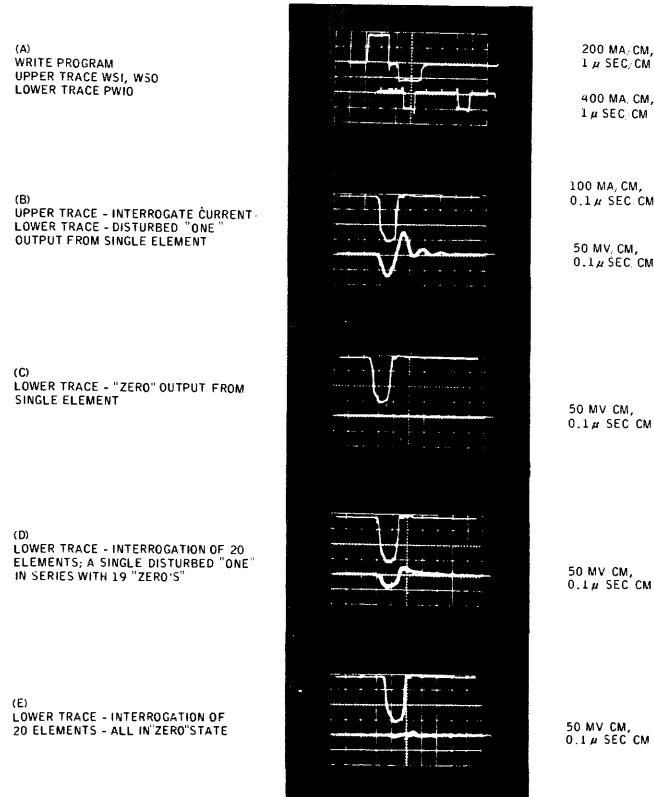


Figure 8. Waveforms Showing Unipolar Operation in the Associative Memory Model.

- posium on Superconductive Techniques for Computing Systems, Washington, D. C., May 1960.
6. NEWHOUSE, V. L., and R. E. FRUIN, "A Cryogenic Data Addressed Memory," Proceedings of the Spring Joint Computer Conference, May 1-3, 1962.
  7. DAVIES, P. M., "A Superconductive Associative Memory," Proceedings of the Spring Joint Computer Conference, May 1-3, 1962.
  8. ROWLAND, C., and W. BERGE, "A 300 Nanosecond Search Memory," Proceedings of the Fall Joint Computer Conference, November 1963.
  9. ESTRIN, G., and R. FULLER, "Algorithms for Content Addressable Memories," Proceedings of the Pacific Computer Conference, November 1963.
  10. SEEGER, R. R., "Associative Self Sorting Memory," Proceedings of the Eastern Joint Computer Conference, pp. 179-188, December 13-15, 1960.
  11. SEEGER, R. R., and A. B. LINDQUIST, "Associative Memory with Ordered Retrieval," IBM Journal of Research and Development, Vol. 6, pp. 126-136, January 1962.
  12. FALKOFF, A. D., "Algorithms for Parallel-Search Memories," Journal of the ACM, Vol. 9, pp. 488-511, October 1962.
  13. LEWIN, M. H., "Retrieval of Ordered Lists from a Content-Addressed Memory," RCA Review, Vol. XXIII, No. 2, pp. 215-229.

# A 16k-WORD, 2-Mc, MAGNETIC THIN-FILM MEMORY

*Eric E. Bittmann  
Burroughs Corporation  
Defense and Space Group  
Great Valley Laboratory  
Paoli, Pennsylvania*

## INTRODUCTION

Small magnetic thin-film temporary-data memories<sup>1,2</sup> have been in use in operational computers since mid-1962, when the prototype Burroughs D825 Modular Data-Processing System<sup>3,4</sup> was installed at the U. S. Naval Research Laboratory. To the present, some 43 additional D825 systems have been placed in use or ordered. The experience gained in the successful operation of these small thin-film stores has encouraged the more ambitious construction of a large, random-access memory for a modular processing system.

Control of the memory module is effected by descriptor words containing 52 bits. The descriptors originate at either a computer or I/O control module. A memory module can receive four descriptors during one request.

Each memory module can perform a number of logic manipulations independently of other modules. A memory module can: execute the conventional read or write instructions on a single word, or on two, three, or four consecutive words simultaneously; read  $n$  words, where  $n$  is a quantity contained in the descriptor; perform a block transfer operation from one area in memory to another, or to another memory module; or perform a search for a requested word or a requested digit, either in itself or in any other memory module, matching against a word or digit supplied.

"Party lines" interconnect the memories with either computer or I/O. Each party line is

assigned a number. If two or more requests appear simultaneously on different party lines, the signal on the lowest-numbered line receives priority. A separate party line interconnects all memory modules, allowing communication from memory to memory.

The memory module is physically divided into two cabinets, each storing 8,192 words of 52 bits each, for a total capacity of 16,384 words. The 52-bit word contains 48 data bits, three control bits, and one parity bit. The control bits act as tags which tell the program whether or not the instruction has been executed.

The read/write cycle of each memory is 0.5  $\mu$ sec, and the access time is 0.3  $\mu$ sec. During the remaining 0.2  $\mu$ sec, the word is rewritten or replaced at the selected address.

The two cabinets of a module can be tested independently of each other. Several test features are built into each cabinet. A test word can be written into all addresses, or into alternate addresses, or into a selected address. A continuous stop-on-error mode compares every readout with the test word. Operation halts on an error, and the faulty word and its address are displayed on the control panel. Single-cycle and single-pulse operation are also possible.

## MEMORY MODULE ORGANIZATION

Figure 1 is a block diagram of one memory module; the interwiring in the memory stack is shown in Fig. 2. To keep the total sense

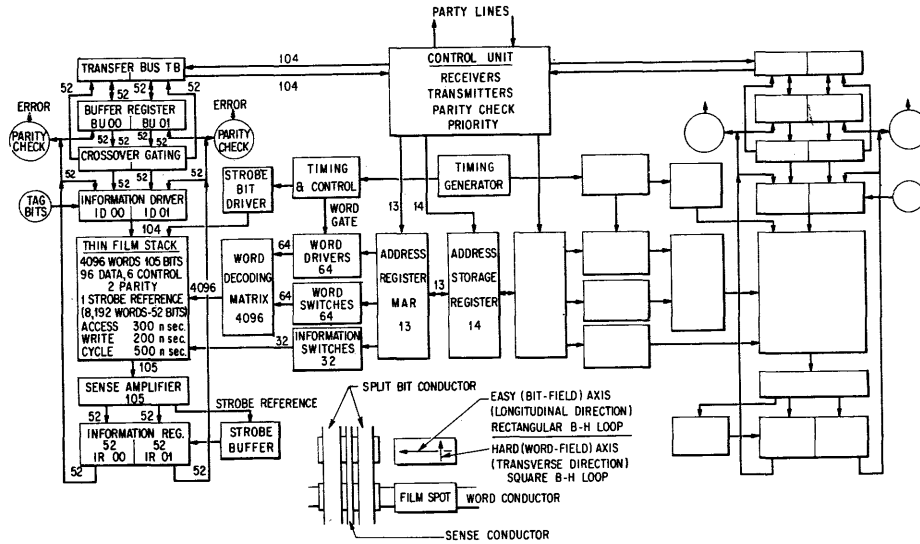


Figure 1. Memory Module, Block Diagram.

delay and sense signal attenuation reasonably low, we organized the stack into a configuration of 4096 words, of 104 bits each, rather than 8192 words, 52 bits each. This kept the total sense delay below 100 nsec for the worst-case address location.

Film elements are deposited 768 per substrate, in a  $32 \times 24$  array. Five substrates in a row provide storage for 32 words, 120 bits each. A single five-substrate film word, therefore, can easily store two 52-bit computer language words. Four such rows (or 128 words on 20 substrates) comprise a plane. A plane with certain associated circuit cards, connectors, and structures is assembled as an integral plug-in unit called a *frame*; 32 frames comprise a 4096-word stack.

A pair of computer words requires 105 bits, including two parity, six control, and one reference bit. The unused bits, or spares, are distributed through the stack for possible replacement use. A row of spare bits can be easily wired into position to replace another row, if necessary. This is normally performed during testing of memory planes, prior to module assembly.

A descriptor word arriving at the control unit receivers initiates a memory cycle. The address data is transferred to the address registers, and a memory cycle is initiated.

The address (6 bits) is decoded at the input of every word driver and at the input of every word switch. Selection of a film word occurs in a diode-transformer matrix. The matrix contains 4096 transformers and the selection diodes. The memory is addressed by the word-organized (linear-select) scheme; each film word line is driven from a single transformer. The current from a selected word driver flows through the matrix to the selected word switch. The transformers have linear (not square-loop) characteristics, and the selected film word line receives a word current pulse. This current interrogates all the film bits, inducing signals into the sense amplifiers.

Planar films remagnetize under the influence of two orthogonally opposed fields. (See inset in Fig. 1.) A word field applied parallel to the film's hard direction rotates the magnetization vectors from their rest position (easy axis) into the hard direction. (Vectors of a bit storing a ONE and a bit storing a ZERO rotate from opposite directions, each passing essentially through  $90^\circ$ , to an almost common hard-direction alignment.) This rotational switching induces a readout signal into the associated sense line. A second field, the bipolar bit or information field, applied parallel to the easy direction (by a bit conductor lying in the hard direction), while the film is still magnetized in the hard direction, determines the future

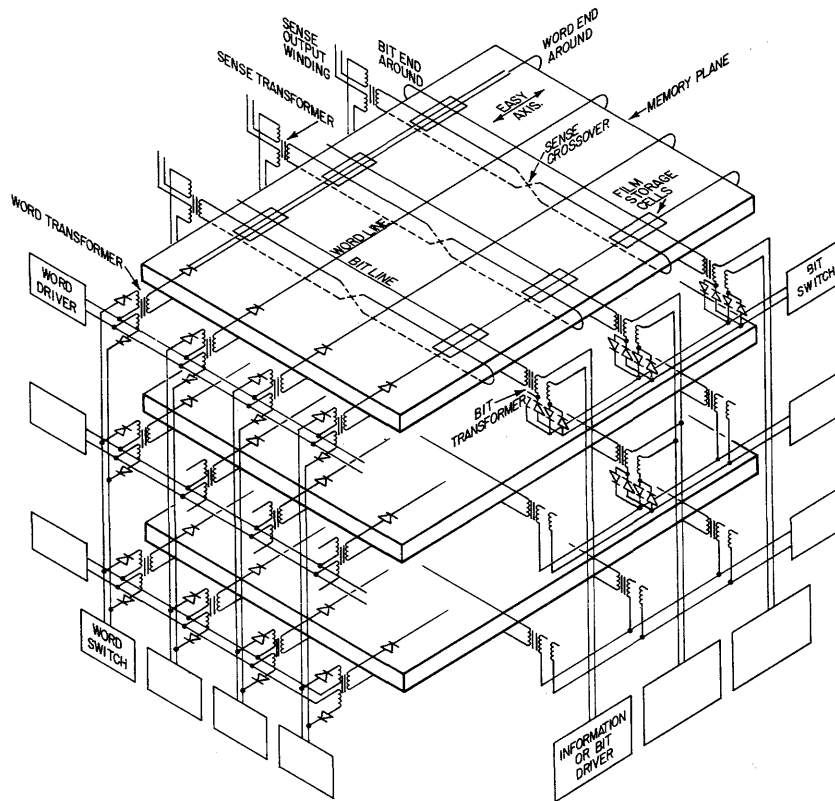


Figure 2. Memory Stack Interwiring.

state of the cell after the word field has been removed. (Vectors fall back through  $90^\circ$  toward either the ONE or ZERO orientation, along the easy axis.)

Interrogation of a word occurs during the leading edge of the word current, and data is written into the films during the trailing edge of this current. Bit currents, present in all lines during word-current turn-off, ensure correct storage of data to be written. The polarity of each bit current determines the storage of a ONE or a ZERO.

A reference bit in each film word (104 bits) was included for the following reason. The sense readout signal has a width of only 50 to 60 nsec, and the delay in the stack can vary as much as 70 nsec for different address locations. To generate a variable time strobe pulse, a strobe reference bit, storing always a ONE, is included in the stack as the 105th bit. The strobe reference bit sense amplifier drives a clock buffer amplifier (strobe buffer) which

supplies a 25-nsec-wide strobe pulse to the information register flip-flops. The strobe pulse sets each bit in the flip-flops to the data state represented by the sense signal passing at that moment through the corresponding sense amplifier.

The bit current flows parallel to the sense conductor, and induces large inductive noises into the sense signal. Transposition of each sense line with the corresponding bit line by a crossover connection in the middle of the memory plane reduces this noise. This connection in every sense line is made after the glass has been sandwiched between the printed-circuit boards. Due to mechanical imbalance between each sense-line/bit-line pair, some noise (as much as 5 mV) remains. Further reduction of this noise is possible by manually adjusting the small sense end-around loop on the plane. Bit-noise cancellation prevents sense amplifier overloading, and ensures reliable operation at high speed.

As an additional means of keeping noise in the sense lines at a minimum, we included another feature in the design; during a write cycle, the flow of bit current is restricted to a single memory plane, rather than being permitted to flow through the entire stack. Each information switch circuit is associated with one plane (32 per stack). One of the switches is enabled by the decoding of five address bits. The information (bit) drivers connect to the appropriate bit lines on each frame through a diode-transformer assembly (Fig. 2). Employing four (rather than two) diodes per transformer has the advantage that the bit switch circuit can be designed for single-polarity current pulses even though bipolar bit currents flow in the plane. Also, the same amount of current flows through the switch, regardless of the information being written into the films, and only two conductors per bit are needed to interconnect the corresponding bit lines between frames. Sneak ground currents are also eliminated with a four-diode scheme. The information drivers see high impedances in every plane but the selected one. This arrangement eliminates the time delay in the bit current, because the bit lines are effectively connected in parallel.

Words are stored 128 to a memory plane, on 32 planes, rather than in the more conventional fashion of a plane storing one bit position for all words. Because of this geometry, and the restriction of bit currents to a single memory

plane, each plane is effectively a 128-word memory stack in itself, functionally isolated from other planes, during write. The sense lines, on the other hand, are series-connected through all bits in the stack, one line per bit position.

### MEMORY TIMING

The memory timing waveforms are shown in Fig. 3; waveforms of actual word current, bit current, sense readout, and strobe pulse are shown in Fig. 4.

Memory read operation begins with an initiate pulse received from the memory control unit. Storing the address information in the address register requires 20 nsec. Address decoding occurs in a single gate level, and requires an additional 20 nsec. The decoding enables the selected word switch circuit, and also one of the 32 information switches. A word gate pulse turns on the chosen word driver at 100 nsec. With a circuit delay in the driver of 50 nsec, current flows in the selected word line at 150 nsec. The sense signals are induced on the sense lines during the word current rise, but, depending upon the location of the word in the stack, may be retarded at the sense amplifier input by as much as 70 nsec. The earliest time at which a signal can appear at this input is at 160 nsec, the latest at 320 nsec. An amplifier delay of 40 nsec allows signals to arrive at the information register at between 200 and 270 nsec. The strobe pulse clocks the informa-

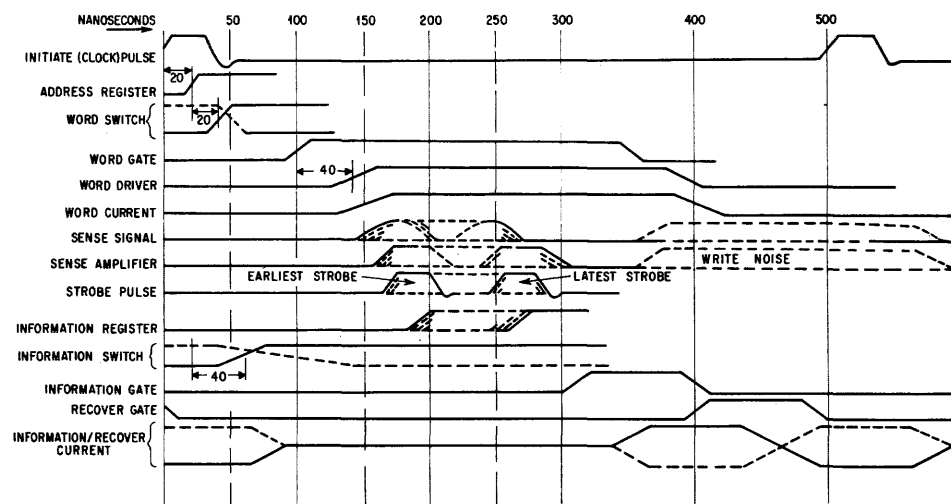


Figure 3. Memory Module Timing Diagram.

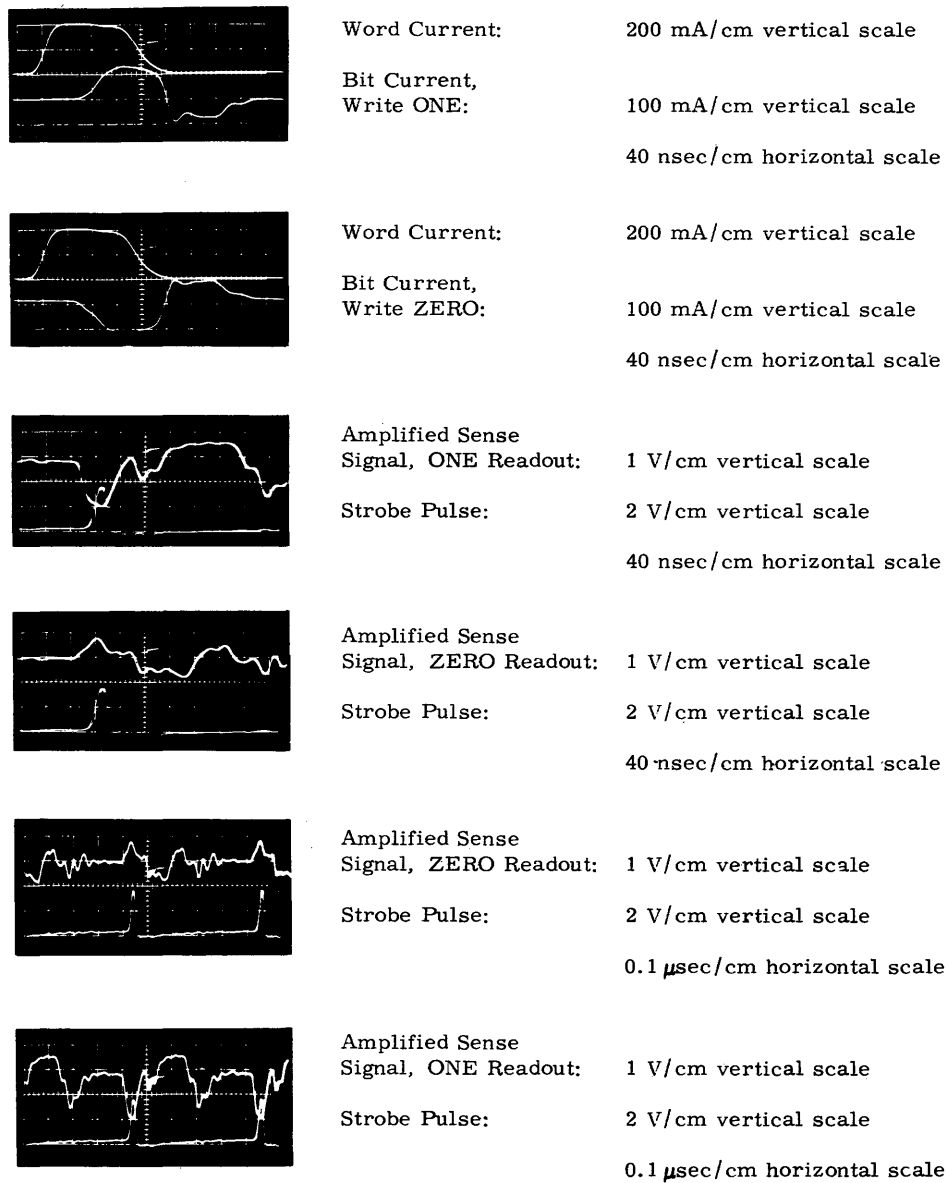


Figure 4. Waveforms: Word and Bit Currents, Sense Readout, and Strobe Pulse.

tion register, which has a delay of 20 nsec. At the latest possible time of 290 nsec, the information register contains the read data.

The write operation begins at 300 nsec. The write cycle either replaces the data read out during the previous read (and contained in the information register), or enters new data into the selected word location, via information drivers. The new data is taken from the buffer register, and is substituted for the signals from the information register. With a circuit delay of 40 to 50 nsec in the information driver, bit

current flows at 350 nsec for a duration of 100 nsec. While the bit current is at its crest, the word current (which has continued to flow since initiation of read) is terminated. Termination of the word current allows the magnetization vectors of the films to rotate in the directions established by the bit currents, and the word is written. To eliminate magnetizing energy which would otherwise remain stored in the pulse transformers employed in the bit circuits, a recover pulse is selectively applied to bit lines. The recover pulse, opposite in polarity to the bit current, and of about the same

duration, terminates at 550 nsec, to complete the write portion of the memory cycle.

### MECHANICAL CONSTRUCTION

A single cabinet (Fig. 5) houses one stack and all associated circuitry; two such cabinets, one containing certain common circuitry (that shown in the middle of Fig. 1) make up a memory module. The front-opening door on each cabinet carries the control panel for that stack and permits full access to the interior. The interior of the cabinet (Figs. 6 and 7) contains two circuit-card racks which may be locked together, and can be swung either separately, or in unison, around a vertical hinge. The stack, mounted in the lower portion

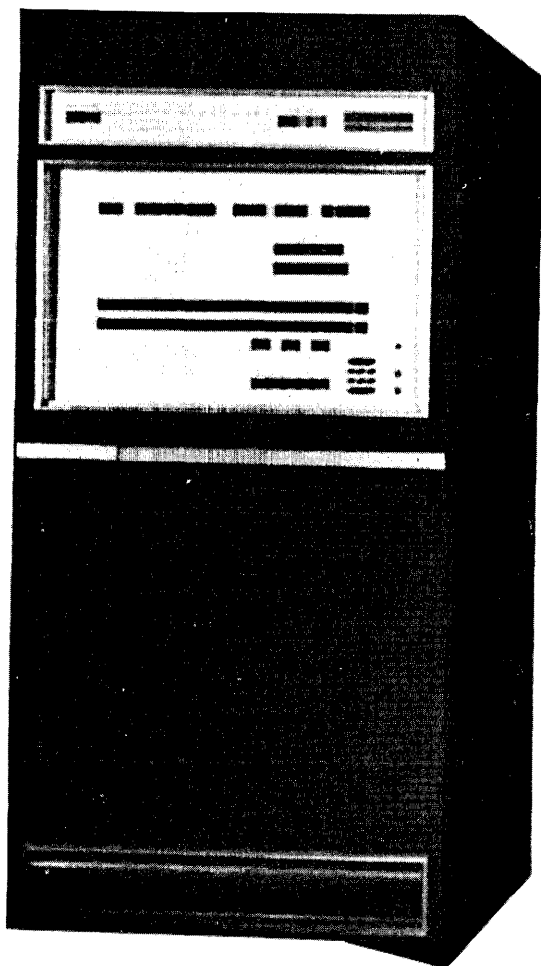


Figure 5. Memory Cabinet, Front Door and Control Panel.

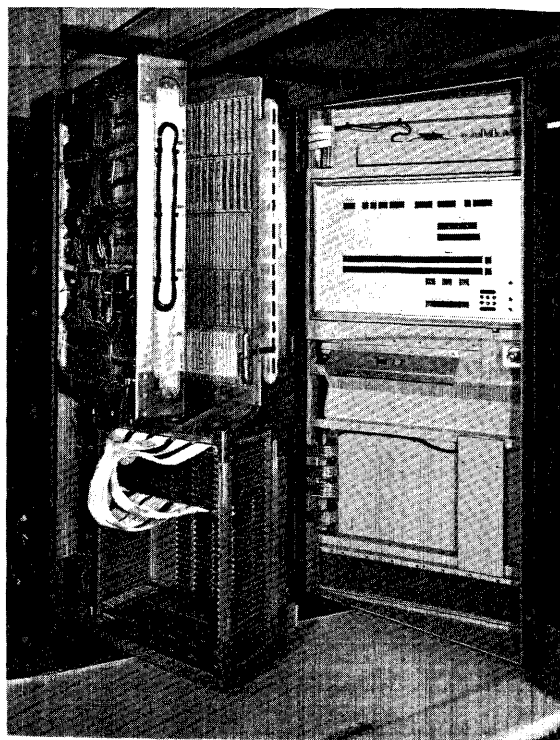


Figure 6. Memory Cabinet, Door Open and Racks Extended.

of one of these racks, as shown in Fig. 6, has the following dimensions: height 30 in., width 26 in., and depth 12 in. The stack housing is an integral part of the rear card rack, which can be swung completely out of the cabinet. The memory frames slide into the stack enclosure from the front, and engage with bit connectors located in the rear panel. The 32 frames lie in the stack horizontally, with a frame-to-frame spacing of 0.7 in. The word driver and switch lines engage the frames through "side-entry" connectors located at the left side of the stack. Placing the word-driver cards and the word-switch cards to the left of the stack keeps the interconnecting wires quite short. (The address decoding matrix is contained on the 32 memory frames.)

The five rows of cards above the stack (Fig. 7) contain, in bottom-to-top order, sense amplifiers, information register, bit drivers, address register, and timing and control circuits.

The separately hinged front rack (Fig. 6) includes space for five rows of logic cards for the party-line transmitters and receivers, input

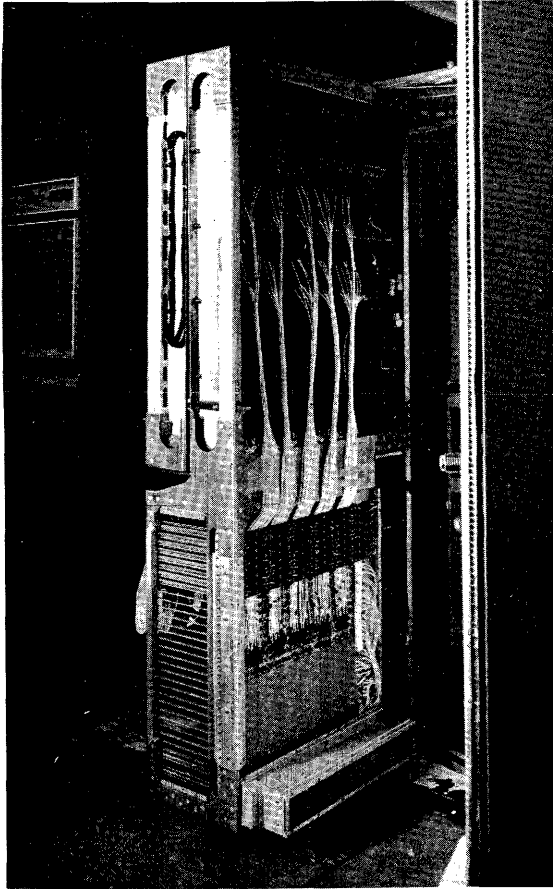


Figure 7. Memory Cabinet, Showing Rear Rack and Bit Switches.

and output decoding, receiving and transmitting registers, and parity-generating and check circuits.

A magnetic shield surrounding the memory stack reduces the disturbing influence of the earth's magnetic field.

A separate power supply is located in the rear of each cabinet, behind the card racks.

The unit operates in a temperature range from 0° to 50° C. Fans, located in the top and bottom of the cabinet, provide air to cool the equipment.

#### THE MEMORY PLANE

The magnetic thin films employed in this system are produced by vacuum deposition of nickel-iron alloy onto glass substrates, while

under the influence of a magnetic field. The films are 1000Å thick; the glass measures 70 by 43 mm, and is 0.2 mm (8 mils) thick. An etching process, applied after deposition, removes the unwanted material from the glass. The 768 rectangular cells contained on one substrate measure 30 by 80 mils each, spaced on 50-mil and 100-mil centers, respectively. The easy direction of film magnetization is along the length of the cell, hard direction along the width, to accommodate shape anisotropy—the demagnetizing effect of the air return path is less significant in this orientation. Two small registration holes, drilled into the glass prior to deposition, help in the alignment of the glass with the conductors during test and assembly.

Each substrate stores 32 words of 24 bits each. Twenty such glass substrates are assembled into one memory plane, as shown in Fig. 8. Arrangement of the substrates into five rows of four each provides storage for 128 words of 120 bits each. (Each word includes 15 spare bits, which are distributed evenly, for possible later replacement of weak or faulty bits.)

The glass substrates of each memory plane are sandwiched between two printed-circuit-board assemblies which measure 20 in. in length and 9 in. in width. Three conductors address every memory cell: a word conductor, a sense conductor, and a bit conductor. The word conductor, 20 mils wide, is parallel to the film easy direction, and lies orthogonally to the sense and bit conductors. (The fields associated with the conductors are, of course, orthogonal to the conductors.) A split bit conductor, each half 20 mils wide, and separated from the other by 50 mils, embraces the 10-mil-wide sense conductor.

Five printed-circuit boards, each with 24 bit and 24 sense conductors, bond to a single flat backing board 0.1 in. thick (Fig. 8). The 128 word lines, printed onto 1-mil-thick Mylar, bond to the rigid sense-bit assembly. All conductors terminate into tab connections on 50-mil centers, located at the edges of the printed-circuit boards.

A 9-mil-thick glass epoxy spacer separates the two printed-circuit assemblies, and prevents excessive forces from pressing onto the glass substrates. A small amount of epoxy glue holds each substrate in its proper location.



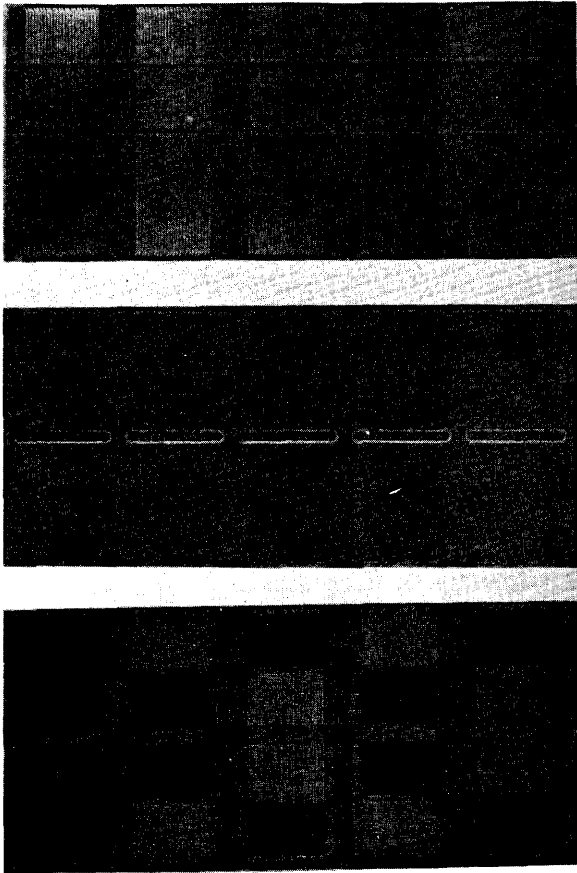


Figure 8. Elements of Memory Plane.

### MEMORY FRAME CIRCUIT BOARDS

A frame (Figs. 9 and 10) surrounds each completed plane. Three types of circuit boards mounted on the edges of the frame—the word selection matrix, five sense boards, and five bit boards—connect to the plane. (There are five rows of substrates in the plane.) The attachment of connectors to the frames helps greatly during debugging and testing, and during replacement of faulty semiconductor components on the plane. (The circuits employed on these boards—for word selection, sensing, and bit selection—are described in greater detail in the next section.)

#### Word Selection Matrix

The word selection matrix, which is part of the frame, contains 128 selection elements. Each element consists of a pulse transformer and three diodes. The transformer is wound with three windings—two primary windings

and one secondary winding—with a turns ratio of 1:1:1. To maintain balanced drive conditions between the word drivers and the word switches, we included two selection diodes, one in each primary winding of the transformer. The third diode in the secondary circuit speeds transformer recovery (Fig. 11). The transformer reduces the capacitive noise induced into the sense signal from the word current, as well as the noise generated during transition of the address selection. Each word line is electrically isolated from all other word lines. Reverse biasing of all diodes in a selection matrix prevents undesired sneak currents. During a

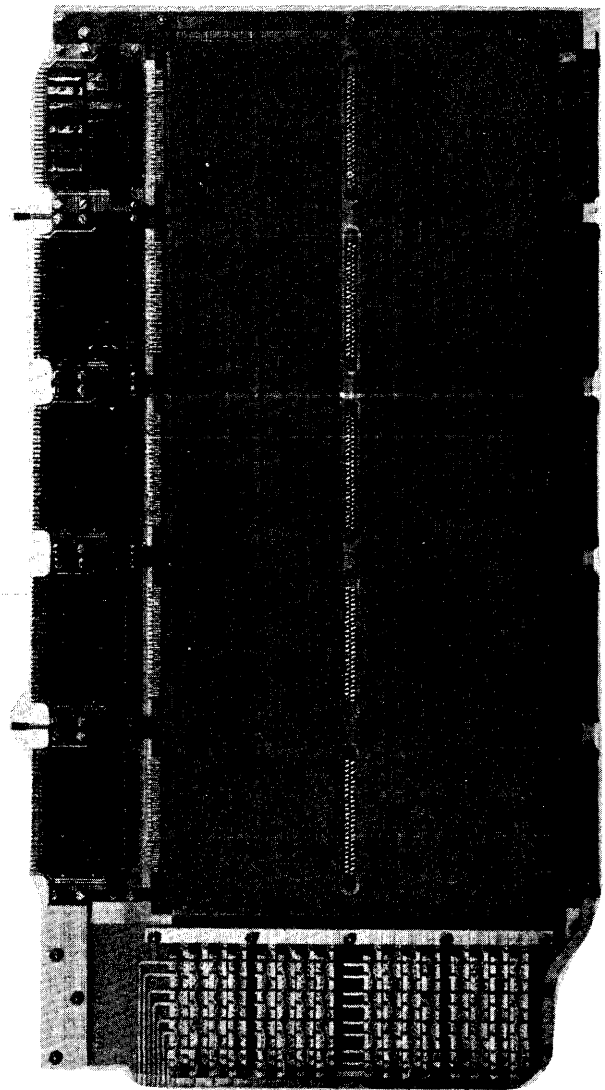


Figure 9. Complete Memory Frame, Front.

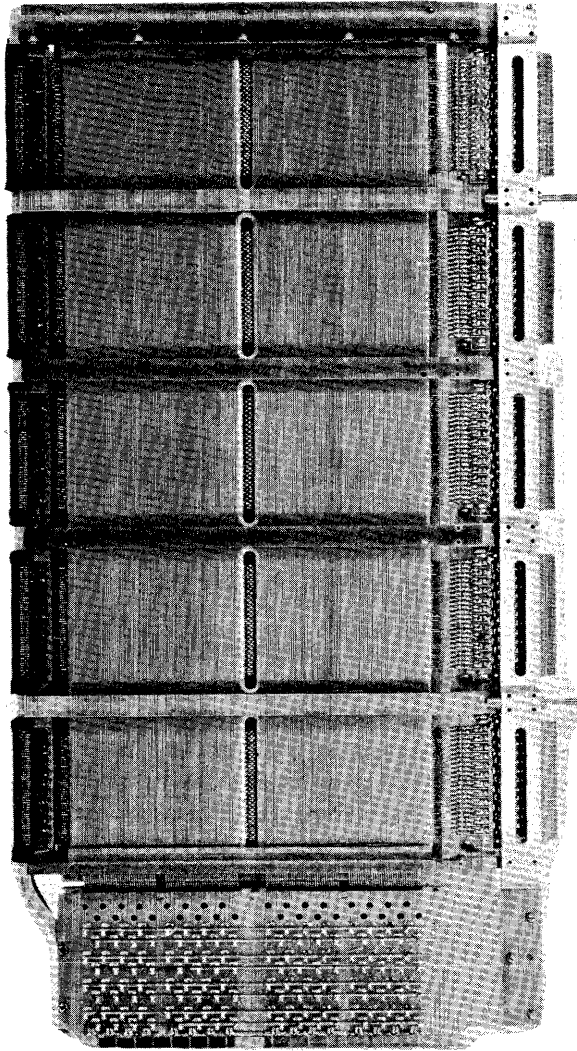


Figure 10. Complete Memory Frame, Back.

memory cycle, this bias is removed from the row of diodes connecting to the enabled switch. In a matrix without transformers, a large voltage swing would be coupled into the sense line, because of the capacity which exists between word line and sense line. The capacitive currents would induce a normal-mode signal which cannot be removed in a differential input circuit.

The memory operates at 2 Mc; this selection scheme, however, operates<sup>6</sup> at speeds up to 6 Mc.

The word-drive and word-switch connections are made through "side-entry" edge-board con-

nectors. The 128 paired output terminals of the matrix, spaced on 50-mil centers, align with the film word lines, and connect to the word lines through welded-on jumper wires. Welded "end-around" connections jumper the far ends of the word lines on the plane, to complete the return path. The nominal word-current amplitude is 400 mA, with a tolerance of  $\pm 10$  percent.

#### Sense Boards

The five sense boards are located at one edge of the plane, and the five bit boards at the opposite edge. Therefore, all sense connections are made from the same edge. A small wire loop shorts the far ends of each sense line. The near end connects to the secondary winding of a transformer, as shown in Fig. 12. Each board contains 24 transformers. Each sense transformer contains three windings; one connects to the film sense line, and the other two connect to an edge-board connector. Four output connections per sense line are required. The connector terminals are spaced on 50-mil centers.

#### Bit Boards

The bit-line selection scheme employed in this memory utilizes a transformer in every line (Fig. 13). The secondary winding connects to the corresponding bit line. A bit current of 100 mA is required to write a single bit. The printed-circuit end tabs on the bit boards mate with the edge-board connectors located in the backplane of the stack (Fig. 7).

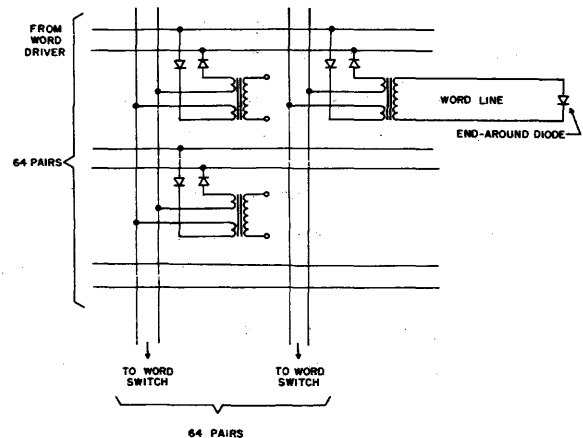


Figure 11. Word Selection Matrix.

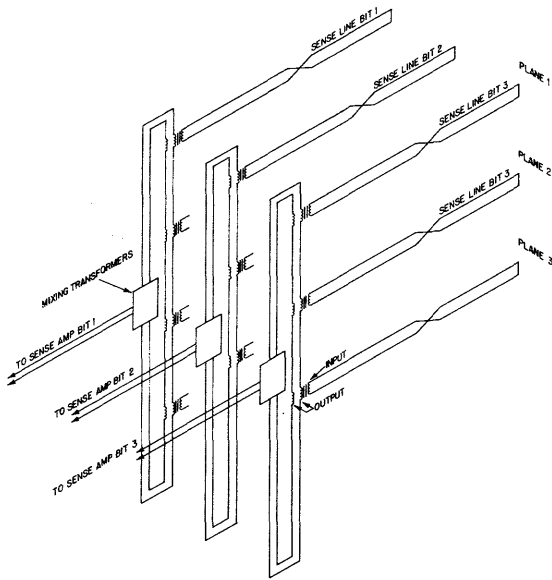


Figure 12. Sense Line Interwiring.

## CIRCUIT DESCRIPTION

### Word-Current Drivers and Switches

The word driver and word switch circuits, which resemble those described by Bates and D'Ambra,<sup>6</sup> can generate currents with 20-nsec rise and fall times when loaded by a small (128-word) memory, such as that employed in the computer module of the D825. The loading of the 4096-word selection matrix (Fig. 11) increases the current rise time to 35 nsec, the fall time to 50 nsec. The driver supplies both a positive pulse and a negative pulse to the interconnecting twisted pair of conductors. The balanced drive arrangement eliminates ground currents and radiating fields which can greatly add to the noise problem. The closeness of the driver and switches to the stack keeps the interconnecting wires short. The total delay (about 10 nsec) from a driver to the word-line end-around short is less than the current rise time, and does not deteriorate the current shape. The drivers and switches supply a current of 200 mA to the selection matrix. The word transformers in the matrix have a 1:1:1 turns ratio, and the output winding receives a current of 400 mA. The drivers and switches connect to the stack through twisted-pair conductors, and both circuits have output impedances of 100 to

150 ohms. Drivers contain a 7-input AND gate at the input, and the switches contain a 6-input AND gate. Delay in the circuits is 35 nsec.

### Sense Amplifiers

The sense lines are effectively series-connected through all of the bits in the stack. The sense transformer output windings of corresponding bits connect together from plane to plane in series fashion. Every sense line contains an end-around loop which shorts the far end of the line. This short reflects to the beginning of the line connecting to the transformer. The sense signal which travels through a transformer on an unaddressed plane receives only a small attenuation after the short is reflected to the output winding. This reflection appears at the input after two line delays, and accounts for the signal delay in the stack. The pickoff is taken from the middle of each line, to halve the delay, as shown in Fig. 12. Signal attenuation for worst-case locations is 6 dB. Nominal sense output at the plane is 1.5 mV. The amplifier has a gain of close to 3000, and a delay of 40 nsec. The amplifier is transformer-coupled into a differential input stage, which is succeeded by two amplifying stages. The amplifier digitalizes the signal, and sends both true signal and complement signal to the associated information register flip-flop.

### Information or Bit Drivers

The bit drivers supply bipolar current pulses, 100 nsec wide, to the stack. The bit-driver input circuit contains the decision elements which

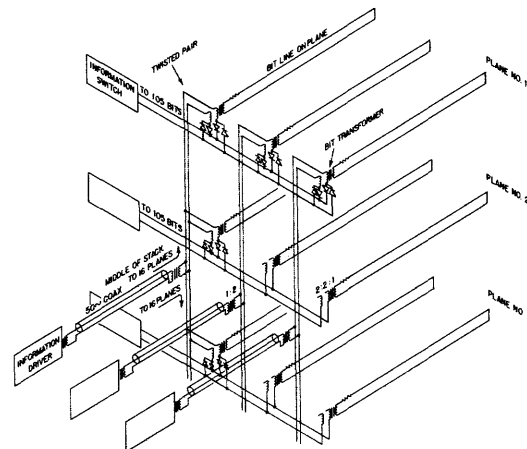


Figure 13. Bit Selection Scheme.

either copy the word stored in the information register or allow new data, obtained from the buffer register, to be written into the stack. In addition, the logic function which determines whether a ONE should be stored as a ONE or a ZERO is included. This is necessary because of the sense-line transposition in the center of every plane. The sense amplifier amplifies only single-polarity signals, and all ONEs stored in the stack must appear as negative signals at the input to the sense amplifier. Therefore, all ONEs are stored as ONEs in half of the stack, and as ZEROS in the other half. The reverse is true for the storage of ZEROS.

The driver input contains two OR gates driven from four two-input AND gates. The bit driver delay is 35 nsec, and the output stage is transformer-coupled and has an output impedance of 50 ohms.

All lines connecting the stack to the back-plane are impedance-matched. The bit drivers, located in the second and third rows, connect to the middle of the stack through five groups of coaxial lines, as shown in Fig. 7. On the frame bit connectors (hidden by wiring and circuits), corresponding bits interconnect through twisted pairs, with an impedance of 150 ohms. A matching transformer connects the coaxial line to the twisted pair.

The bit switches for the eight frames shown installed in Fig. 7 cover the coaxial bit lines. Each bit switch circuit is contained in a strip which aligns with the associated memory frame. One switch circuit handles the currents from one row of substrates. Five such circuits on a strip are driven from a common drive circuit (visible at the far right in the photograph).

The bit line impedance on the frame is 10 ohms. The nominal bit current is 100 mA. The bit transformer has a turns ratio of 2:2:1, which requires 25 mA of current in the primary. The matching transformer which connects the interconnecting twisted pair to the coaxial drive line has a 1:2 turns ratio. This transforms the impedance from 160 ohms to 40 ohms, which is close enough to match the 50-ohm coaxial cables. The current necessary from a bit driver, to produce 100 mA of bit

current in a plane, is 50 mA.

The total of 105 times 25 mA of current is received by the selected information switch. The switch is divided into five individual circuits, operating in parallel, each handling a total of 500 mA; each circuit handles the bit currents in one substrate row.

#### SUBSTRATE TESTER

A substrate tester (Fig. 14) submits every bit on a substrate to a pulse test which subjects the bit to disturbing fields resembling worst-case examples of those encountered during actual operation.

The films exhibit pronounced magnetic anisotropy; the B-H hysteresis loop along the film easy axis is rectangular, while that along the hard axis is linear. The films also exhibit various disturbing thresholds for fields applied in different directions. Because of the film's

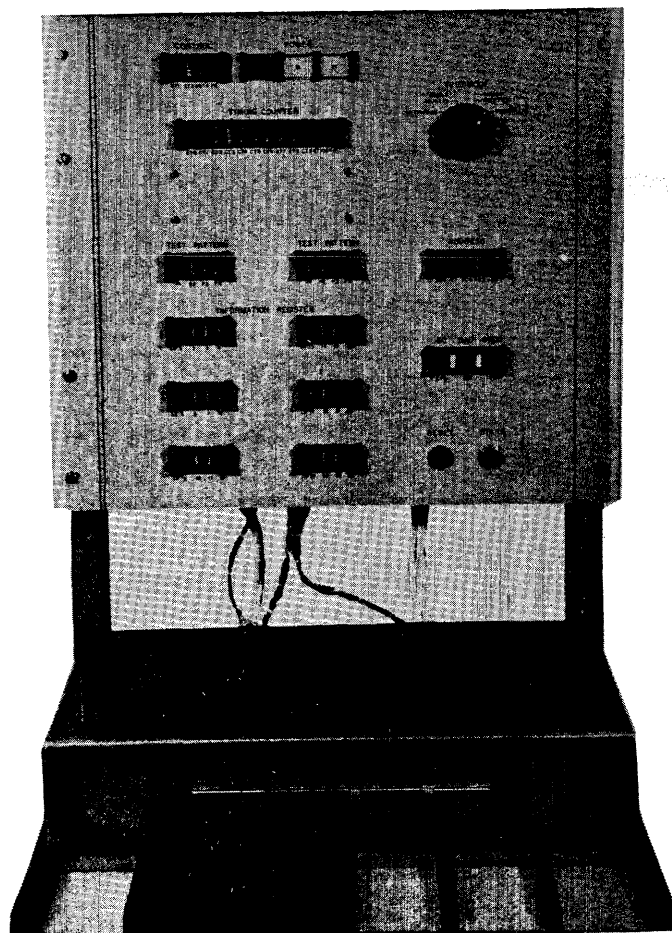


Figure 14. Substrate Tester.

linear loop in the hard direction, a low disturb threshold exists for fields parallel to the transverse (hard) direction. (See Fig. 2.)

The test fixture (Fig. 15) is assembled from circuit boards similar to those surrounding the memory plane in the actual memory stack. Substrates are inserted and removed through a narrow slit located at the word end-around connections. Two pins through holes in the glass, used to register substrates and circuit boards in the actual memory-plane sandwiches, furnish similar registration in the substrate tester.

A relay rack contains the circuitry necessary to test the substrates. Indicators for all flip-flop circuits, located on a control panel, allow observation of the test, and help during operational maintenance.

The worst disturb condition exists when a stored ONE bit is surrounded by all ZEROs, or when a ZERO is surrounded by all ONES. The test word 10001000100. . ., disturbed by all ZEROs in adjacent locations, is tested for ONES; the test word 01110111011. . ., disturbed by all ONES in adjacent locations, is tested for ZEROs.

At the beginning of an operation, the test word (ONES) is written into all 24 bits of the first address. Next, all ZEROs are written into the adjacent word location; the latter is repeated as many as 32,000 times. The rewrite

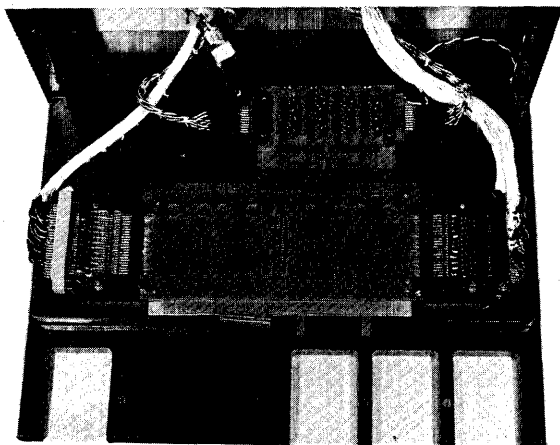


Figure 15. Substrate Tester Test Fixture.

process subjects the test word to transverse and longitudinal disturbing fields applied simultaneously. The test word is read after the completed disturb cycle, and its content compared with a program register. A match continues the test, by shifting the test word to the next bit (0100010. . .), and the disturbing continues. After three shifts, every bit in the first address has been tested for ONES. The test word for ZEROs follows. This test continues until all bits are checked for ONES and ZEROs. The disturb word can be written to the right or to the left of the test word, alternatively to the right and then to the left, or parallel to the right and left. The parallel writing of two words which embrace the test word constitutes a more than worst-case condition—a condition that never occurs during memory operation—but allows grading of the substrates.

Substrates which pass the 32k disturb test are assembled into memory frames. These substrates tolerate about 4k to 8k disturb pulses, when tested in the more-than-worst-case parallel mode. The test is fully automatic, and the output signal of the sense lines is not monitored. The tester operates at a frequency of 1 Mc; one disturb test requires 8 seconds if no error occurs. Evaluation of a good substrate requires about 30 to 60 seconds, because the substrate is also retested in the parallel mode. Operation stops on a bad bit, and panel indicator lights display the location of the bad bit, and whether the failure represents a bad ONE or a bad ZERO.

Film disturbance shows dependence upon current rise times. Slower current pulses tend to disturb less. Current rise and fall times of 20 nsec are available in the tester, as compared with 35 nsec in the stack. The size and the larger number of selection elements reduce the current rise and fall times in the stack.

Substrates which pass the 32k disturb-pulse test also pass consistently a test of 4 million word and 250 million bit disturb pulses in the frame tester.

#### FRAME TESTER

Evaluation of fully assembled memory planes (frames) takes place in a frame tester (Fig. 16). A relay rack houses the circuitry and

power supplies. A specially designed fixture allows the frame to slide into the word connector in an upright position. This provides the connection necessary to address all 128 word lines. A movable rack, containing sense amplifiers and bit drivers for the examination of 24 bits, can slide vertically to the desired group of lines. Printed-circuit edge-board connectors mate with the appropriate conductors. With this mechanical arrangement, an equal conductor length is consistently maintained during the examination of the five groups of substrates, to correspond to the actual stack construction.

The evaluation consists of two phases: first, the bit-write noise is reduced by manual adjustment of the "end-around loops." Secondly, a disturb test, similar to that performed on the substrates, is run. The frame tester operates as a memory exerciser, with the capability of inserting worst-case patterns into the plane. The automatic program rewrites the disturb word up to 32,000 times; manual operation allows any desired number of disturb operations.

The tester operates at one of three different frequencies: 2 Mc, 4 Mc, or 250 kc. Single-pulse operation is also available. Although substrates cannot easily be removed from an assembled plane, up to three bad bits can be tolerated in each of the five sense-bit groups. The spare lines can replace lines containing faulty or marginal bits, but a small wiring change is necessary.

## CONCLUSIONS

The operation of this half-microsecond-cycle memory module represents a significant achievement in a program of magnetic thin-film development for computer storage which was begun at these laboratories in 1955. Large numbers of substrates were processed and tested, and memory plane assembly and test are now routine operations.

Memory frames which contain 20 substrates (15,360 bits) can be assembled without great difficulty. The limitations were imposed by the printed-circuit boards, and were due to dimensional tolerances.

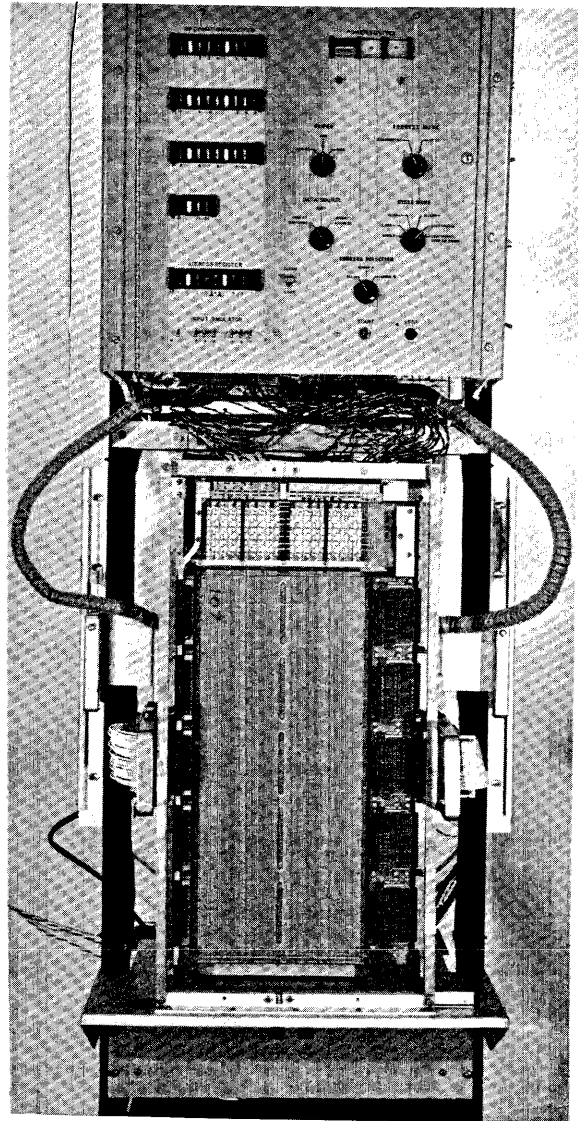


Figure 16. Frame Tester.

Cost-per-bit reduction can be achieved by increasing the number of bits contained in a single pluggable unit, because the interconnections in the stack contribute significantly to the total memory cost.

A shorter memory cycle can be made possible by reducing the total sense delay, and by the elimination of the bit recover pulse. The pulse transformers will be replaced by active solid-state devices. A reduction of 150 nsec—50 nsec from a shorter sense delay and 100 nsec from elimination of the bit recover pulse—make a cycle time of 350 nsec, or 3-Mc operation, possible.

The capacity and speed attained with this memory are clear indication that magnetic thin films have become the optimum storage elements for reliable, nonvolatile, fast-access memory.

#### ACKNOWLEDGMENT

The author wishes to express thanks to the many people at these laboratories who were responsible for the successful completion of this memory, especially to F. C. Doughty, A. M. Bates, P. A. Hoffman, J. W. Hart, J. S. Jamison, and L. N. Fiore, for their technical assistance; J. H. Engelman and R. P. Himmel for film fabrication; G. Sabatino, K. McCardell, and S. V. Terrell for film and memory testing; B. C. Thompson and F. Rehhausser for logic design; and R. E. Braun, G. J. Sprenkle, and G. J. Kappe for mechanical design.

#### REFERENCES

1. BITTMANN, E. E., "Thin-film memories: some problems, limitations, and profits," an invited paper presented at the International Nonlinear Magnetics Conference (INTERMAG), April 1963, and published in the *Proceedings*.
2. RAFFEL, J. I., *et al.*, "The FX-1 magnetic film memory," Report: MIT Lincoln Laboratories TR278, November 1962.
3. ANDERSON, J. P., *et al.*, "The D825: a multi-computer system for command and control," *AFIPS Proceedings, 1962 Fall Joint Computer Conference*, December 1962.
4. ANDERSON, J. P., "The Burroughs D825," *Datamation*, April 1964.
5. THOMPSON, R. N., and WILKINSON, J. T., "The D825 automatic operating and scheduling program," *AFIPS Proceedings, 1963 Spring Joint Computer Conference*, May 1963.
6. BATES, A. M., and D'AMBRA, F. P., "Thin-film drive and sense techniques for realizing a 167-nsec read/write cycle," *Digest of Technical Papers, 1964 International Solid-State Circuits Conference*, February 1964.

# A SEMIPERMANENT MEMORY UTILIZING CORRELATION ADDRESSING

*George G. Pick*

*Applied Research Laboratory, Sylvania Electronic Systems  
A Division of Sylvania Electric Products, Inc.  
Waltham, Massachusetts*

*Summary:* A mechanically changeable, semi-permanent, random access memory with a 16,384 twenty bit word capacity is described. This solenoid array memory is useful for stored programs and tables in computers, character generation and as a combined input and storage device for special purpose computers. It utilizes an associative technique to allow addressing of any of its 1024 sixteen word data-containing sheets, which completely avoids any need for electrical connections to the data-containing sheets or for any ordering of the sheets within the memory. Each sheet is a very thin printed circuit onto which data is entered by etching or punch-card controlled cutting. The data is inductively interrogated by means of solenoids which pass loosely through the sheets. The sheets are contained in loose-leaf notebook-like magazines which fit into a file drawer.

The present memory's access time is 0.7 microsecond and its cycle is below two microseconds.

## INTRODUCTION

In recent years there has been an increasing interest in read-only random access memories. This class of memory has developed along two paths, those electrically alterable and those mechanically alterable. This device falls into the latter class. The solenoid array memory described here is a development which followed the solenoid array correlator and memories de-

scribed in an earlier paper.<sup>1</sup> In previously reported work<sup>2, 3, 4</sup> inductive coupling was also used, but electrical connection had to be made to the stored data, or else the capacity was very limited. Some work<sup>5, 6</sup> allowed use of connectionless data containing media, but in each case, precise data alignment and interleaving structures were needed. The described memory uses thin copper clad "Mylar" printed circuit sheets which are placed adjacent to each other with no interleaving of any sort. All coupling, into and out of the data planes, is by inductive coupling from and to two respective solenoid arrays which pass loosely through holes in the data planes.

The addressing solenoid array is driven by an input address which has been transformed into an error-correction type code. The address is simultaneously correlated or matched to the stored addresses on each data plane with the result that the autocorrelation on one plane is a voltage positive enough to exceed its diode conduction voltage, and on all unselected planes the cross-correlations result in voltages which are well below, or negative, to that voltage. In consequence, a current is allowed to flow in only the selected plane's data path, allowing only that plane's data to be sensed by the pick-up solenoids.

This association between a coded address and its plane's data allows the mechanical flexibility mentioned earlier. The data plane may



be positioned anywhere along five inches of the solenoids' lengths, and as long as there is only one data plane in the stack with that address, it is uniquely accessible.

Each sheet contains sixteen words or 320 bits. To avoid the need for 320 amplifiers, the four lower order address bits are used to select the appropriate group of solenoids and connect them to the sense amplifiers. This technique allows the packing of many words on a single data plane, thereby efficiently multiplying the capacity of the memory and radically increasing its effective bit packing density on normal length words. Single data sheets of practical size can contain upwards of a thousand bits.

The ease of data change and the low cost of the storing medium allow this memory to be considered for tasks where it acts not only as a memory, but as an input device as well. Magazines, containing tens or even hundreds of thousands of bits, can be stored on a shelf and inserted into the memory when required with an ease comparable to changing a modern magnetic tape cartridge. Thus for some computing systems, mechanical input reading devices may be replaced by a rugged, mechanically static, semi-permanent memory of this type.

#### Overall Description

The solenoid array device described utilizes long, thin solenoids to provide a simple, non-critical magnetic coupling between the data containing planes and the array memory structure. The memory is organized so that the data-containing planes need have no connections other than magnetic, and this realization required two basic functions—unique data plane drive and appropriate sensing of the driven plane for the selected stored data. These two functions are almost independent and are realized by a driver solenoid array and a sense solenoid array. The driver solenoid array is in essence a substitute for a 1024 line linear addressing matrix and the resultant connection pair that would be needed to each of the 1024 data planes.

The sense or pick-up array detects if the various bit positions on each driven plane contain a one or zero. The sense solenoid outputs are connected so that appropriate gating can con-

nect the output of only one group of solenoids, a word group, to the sense amplifiers. The arrays are shown in Figure 1.

The principle of operation of the solenoid array is based on the transformer. On any transformer, if a wire passes around its flux path, there is coupling, and if it bypasses its flux path, there is only stray or minimal coupling. With solenoids, the same rules apply with little modification.

In the memory, the drive array and the sense array are separate components which are only connected through the stored data planes. In series with this connection on each plane is a diode which acts as a switch that allows one and only one plane to be connected at one time during the interrogation. See Figure 2.

The address is stored on each plane on that portion which slips over the addressing array. This matrix of mutual inductances which couple a digital input address word simultaneously to all the data planes perform a correlation or dot product operation. The operation thus performed is given by

$$T_j = \sum_{k=1}^{15} W_{jk} U_k$$

$j = 1, 2, \dots, 1024$ ; where  $U_k$  and  $W_{jk}$  are the  $k^{\text{th}}$  components or cells of the input address word  $U$  and the stored address word  $W_j$ , respectively, and  $T_j$  are the simultaneous individual output voltages generated on each plane.

The correlation is formed by simultaneously energizing 15 solenoid pairs, in either the "zero"

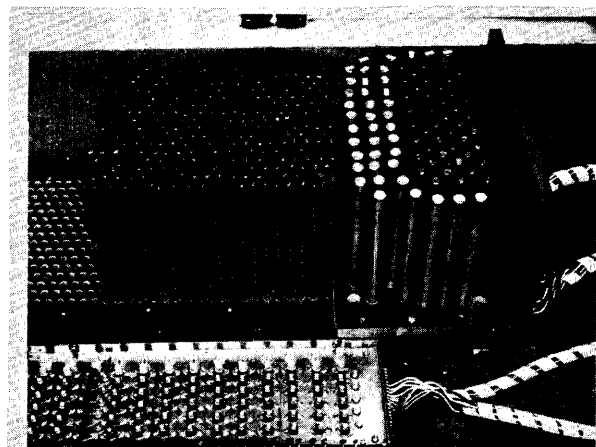


Figure 1. Solenoid Array Without Planes.

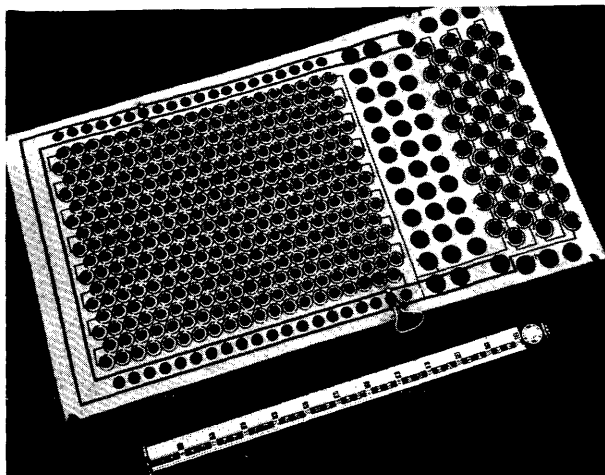


Figure 2. Data Plane.

or "one" positions—depending on the input address. The individual multiplications that result are shown in Figure 2A and these positive and negative voltages on each plane are summed together because all of the paths are in a series circuit.

The right portion of the photograph in Figure 2 shows the addressing paths, plus bias positions to be described later. It may be noted that the loops on vertically adjacent aperture pairs always encircle one and bypass the other.

At the bottom of the photograph is a small component, a diode, which "detects," and allows current to flow only in that plane where the addressing correlation resulted in a positive voltage sum with respect to the diode conduction polarity.

The described addressing operation is, as was explained earlier, a substitute for a connection pair to the left portion of the data plane, which stores the actual data.

The data portion of the plane on the left side of Figure 2 is organized into 16 rows, each representing a 20 bit word. Two rows are paired into one major loop and there are eight loops on the plane, one above the other. The sense solenoids are not paired like the drive solenoids, hence the data for each bit is stored by cutting the path so that each solenoid, or bit, is either inside or outside the enclosed area or loop. The distance between the two sides of each major loop is of little consequence, hence

two rows or words can be placed on one loop. (The coupling loops are the paths starting on the right of the data portion, going to the left, up a short distance and returning to the right. All these loops are in series with all the others, the diode and the addressing array loops on the right.)

Figure 2B shows the operation of the sense solenoids and the manner in which they pick up the stored data. In a later section the organization for solenoid selection switching will be described more fully, however, it should be clear that the interrogation of a data plane results in parallel output of all data bits on the plane. The selection switching circuit is used only to reduce the number of sense amplifiers and subsequent gating circuits. If each plane contained only one word, selection would be eliminated.

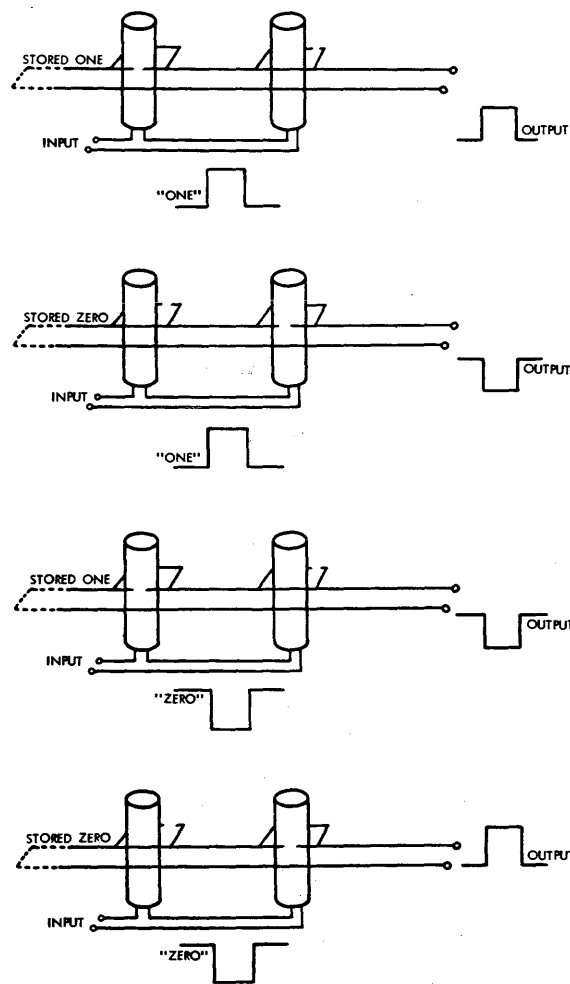
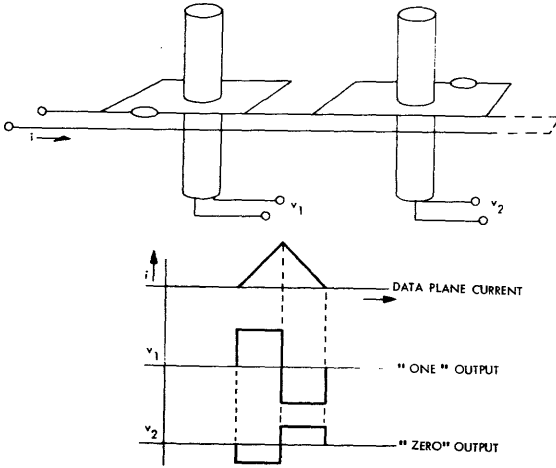


Figure 2A. Data Plane Driving Solenoid.



2B. Addressing Solenoids Driving Stored Address.

The error-correcting code's first function is to allow unique selection of one plane, thereby addressing many planes with a modest number of drivers and solenoids. The long "distance"\* of the code brings with it the advantage of redundancy which results in high reliability and driver load sharing. (In practice, it has been found that degraded or missing drive pulses on a few drivers have little effect on memory operation.)

The operation of the memory can be described by Figure 3. The address is stored in the address register. The addresses' lower order bits are decoded into sixteen sub-addresses for selecting the appropriate sensing solenoid group. The higher order address bits are operated on by the coder to form a Hamming code which operates the addressing solenoids. When the addressing array is driven, current rises on the selected plane and the pick-up or sense solenoid array emits the selected word to a bank of sense amplifiers.

#### Addressing by Correlation

A solenoid array memory has previously been built in which the data planes were conductively connected and were addressed by a relatively straightforward coincident voltage technique in rectangular matrix with a diode connected in series with each plane's path at each crosspoint. Another memory was built in which a single solenoid was used to drive each single

\* In a coding sense.

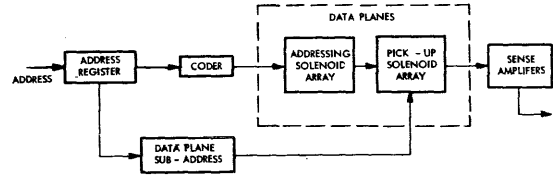


Figure 3. Block Diagram.

data plane with a unique and orthogonal address. The first memory required connections to the data planes, which was acceptable only if data was to be changed infrequently, and the latter memory was limited to a modest number of planes equal to a practical number of drive solenoids, namely, about fifty planes.

Addressing by correlation provided an exit from these limitations. Well developed correlation techniques were available<sup>1</sup> from which a correlator could be designed that would accurately correlate a binary address and thereby achieve a form of connectionless associative addressing. However, the original solenoid array correlator used air cored solenoids whose output voltages were too low to generate enough voltage to drive a data plane. Ferrite cored solenoids were designed which improved the coupling, allowing much higher drive voltages to be delivered to the data planes. However, in spite of compensation, the cored solenoid's coupling to the data planes was much less uniform than that of the air cored units (e.g., 15 per cent versus 1 per cent), and, even with the high outputs available, single output voltages were too low for reliable operation. Hence, the need for load sharing and the requirement for less critical drive voltage amplitudes combined to recommend an error-correction type code.

Use of a non-orthogonal code causes the requirement for a nonlinear component which would detect the positive selection voltage and allow the drive current to flow in the plane—a diode. Since the diode could be a permanently prefabricated part of each plane, and a mechanical arrangement was found that did not increase the total thickness of a stack of planes, the diode was not considered objectionable.

#### Applicable Error-Correcting Codes

Mathematics recognizes many types of codes that could be applied to the present device.

The memory field has seen the use of the usual binary codes and classes of orthogonal load sharing codes<sup>7</sup> for addressing or driving core memory selector matrices. In the case of the described memory, binary codes would be unsatisfactory because the difference between a matched or selected correlation and the closest unselected one is too small, namely, one bit. The aforementioned orthogonal codes excel in that the selected address correlation would receive the sum of all or most of the driving bits, and all the unselected addresses would have no drive at all, by mathematical definition of orthogonality, but unfortunately, orthogonal codes always require at least as many bits as there are addresses. They would fulfill the described memory's load sharing requirement, but would sharply limit the possible number of data planes.

The error-correcting alphabets were designed to encode relatively long blocks of bits, hence the number of possible addresses is relatively large. The use of a diode detector removes the need for code orthogonality, and error-correcting codes are inherently efficient in their use of redundant bits to achieve long coding distances or weights.

Two codes were found that were easily applied to the addressing problem. The first is the well known Hamming code, in particular a code with 10 information bits, 5 redundant bits and a "distance" of 4 bits. A second code, even more attractive, is the Golay code, which for this application would represent 10 information bits, with 12 redundant bits and a distance of 8 bits. Both codes can be conveniently generated by either shift register encoders or parallel modulo-2-sum networks. Many variants of these codes are available for both larger or smaller addressing capacity requirements.

For magnetic reasons to be discussed subsequently, correlating by means of single solenoids, where zero or one is represented by absence or presence of an input drive, is undesirable. A more practical arrangement is to drive pairs of solenoids in parallel, and to drive them with one polarity for a "zero," and the other for a "one."

The effect of this arrangement is that the range of correlation outputs is extended from

0 to  $+N$  ( $N$  = number of bits) to a range of  $-N$  to  $+N$ , thereby doubling the distance or weight of the code. As a result, using the Hamming code, an autocorrelation is  $+15$  units of voltage, and the nearest cross-correlation is  $+7$ . In the Golay code, the respective figures are  $+22$  and  $+6$ . This distribution of Hamming code outputs, with no bias, is shown in Figure 4.

The correlation outputs, as they stand, possess the necessary "distance" properties, but their absolute levels are not optimum for practical operation. The distribution of the unselected outputs must be shifted so that a diode (or diode-Zener diode) detector on the plane can efficiently prevent current flow. (The reasons for choosing either type of detector are discussed subsequently.) The output distribution shift is readily achieved by adding fixed bias in the form of additional solenoid drivers which always operate in the same polarity.

#### Application of the Data Plane Addressing Techniques

This section describes the technique of generating the codes, the circuitry of the solenoid drivers, the structure and design criteria of the drive solenoid themselves and the data plane detector considerations.

#### Input Register and Coder

The binary address of the desired data plane is entered into a buffer register. This address contains the data plane address along with the additional address bits for the subselection of data within the plane. The data plane address bits, in ordinary binary code, are themselves

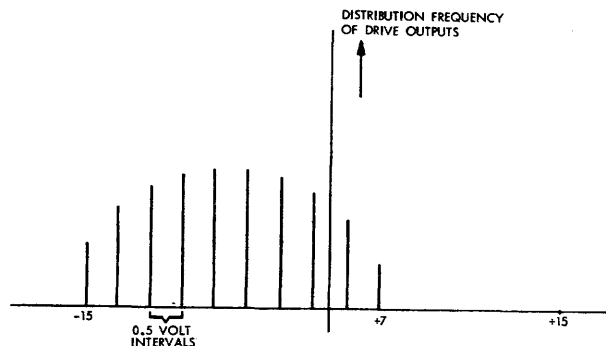


Figure 4. Output Distribution on "Paired" Hamming Code.

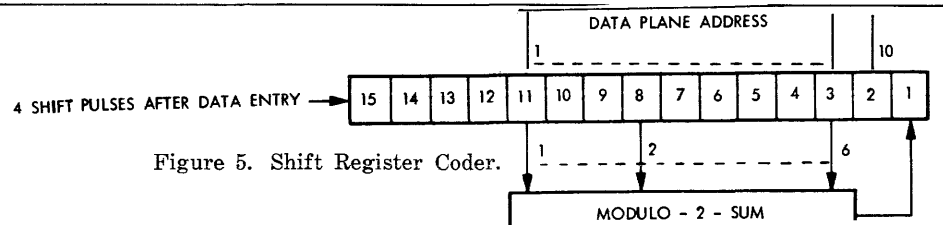


Figure 5. Shift Register Coder.

used to each control a solenoid drive polarity; and redundant bits must be generated from the original bits by one of several common techniques.

The best technique for generating the codes is the shift register encoder as shown in Figure 5. In this device, the original address is entered as shown in shift register positions 2–11 and the modulo-2 adder operates to set position 1 accordingly. The register is shifted, a new bit generated and the operation is repeated until the data has been shifted to the left-most position. For a Hamming code with 5 redundant bits, 4 shifts are necessary. This technique is simple to instrument, uses a small number of circuits and is most attractive with the following exception. The coding must be done before interrogation, hence the speed of this operation directly affects the access time. Thus for a given logic speed, the minimum access time is clearly limited by the time consumed by the required shifting operations.

The alternative is to generate all the redundant bits in parallel. It can be shown<sup>8</sup> that all redundant bits can be determined as modulo-2 sums of the original information or input bits. Hence by instrumenting parallel modulo-2 adder logic (exclusive-or), all redundant bits were generated at once. This is shown in Figure 6. Unfortunately, efficient codes have little logical overlap between their redundant bits, hence the amount of circuitry is not inconsiderable. (Typically, a Hamming coder requires about 50 NAND circuits and a Golay coder about 110.) Alternative simplified circuits and magnetic configurations are available, but some degree of complexity remains.

For the Hamming codes used in the described memory, the code generation was parallel. A second unit now under construction, uses a Golay code and its redundant bits are generated serially, with higher speed logic to partially compensate for the multiple logic cycles. In that unit, with modest access time, it was found more economical to supply a sepa-

rate, faster clock and control counter, all instrumented in higher speed logic, than to generate the Golay code in a parallel.

#### Drive Solenoids

The drive solenoids are operated in pairs, with their respective windings connected in parallel so that for one given drive polarity the solenoid flux polarities are opposite as shown in Figure 7. This balanced configuration achieves an approximation of a closed magnetic circuit without the need for an actual closure. Although, because of the air gaps, the mutual inductance between the two solenoids of a pair is not large, the superposition of the individual solenoid fields radically reduces the stray flux. Further, although the correlation used for addressing is very non-critical, the drive pattern sensitivity of individually driven solenoids would be unacceptable. Thus, to minimize the need for magnetic shielding between the drive and pick-up arrays, to minimize drive pattern sensitivity and to somewhat improve mutual coupling to the data planes, paired solenoids can be fully justified.

As a bonus, as was mentioned earlier, the availability of two bit positions on the plane allows the storage of positive and negative cor-

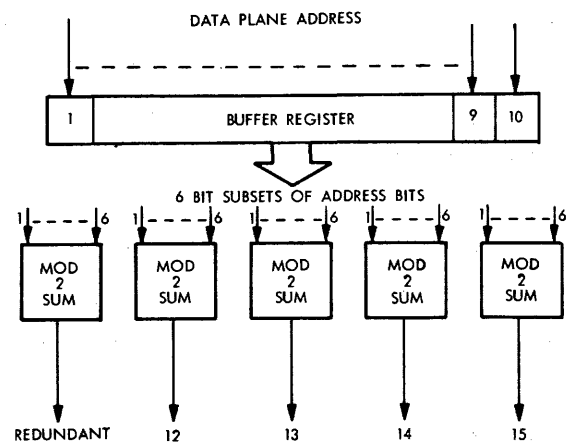


Figure 6. Parallel Coder.

relation weights, thereby automatically doubling the correlation distance, when the solenoids are reversibly driven for one and zero inputs.

### Magnetic Structure

A solenoid using no ferromagnetic core has a very uniform coupling to a surrounding loop over almost its whole length as is shown in Figure 8. However, for a given number of turns, an air core solenoid has a relatively low self-inductance, therefore presenting a low impedance to its driver. Hence many turns must be used for practical air core solenoids with the consequence that the transformer step-down turns ratio is large. To allow a smaller number of turns, larger diameter air cores or ferrite cores must be utilized to maintain a practical level of self-inductance. The resulting coupling to a loop unfortunately becomes very

nonuniform as shown in Figure 9 by the dashed line, an undesirable situation since the drive voltages induced in the planes would vary depending on the position of the plane along the solenoid.

To compensate for the nonuniform coupling, two techniques were evolved. The simplest was to vary the turns density along the winding so that regions near the end were more densely wound than near the middle. For good results, this technique will require careful control of winding density, which will be easy to achieve on production machinery but is difficult to do in the laboratory. The alternative technique was to use a linear winding and to vary the ferrite permeability by using short ferrite rods butted against each other. Since the reluctance of the solenoid return path is relatively large, small air gaps between the rods were found

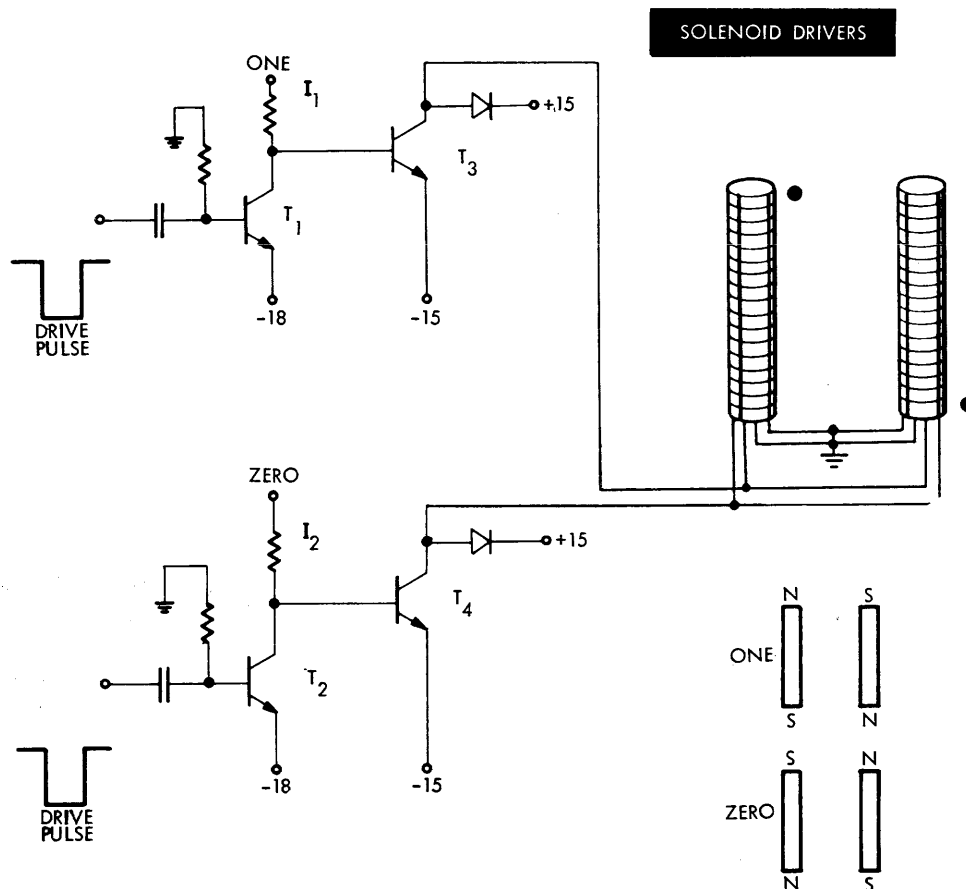


Figure 7. Solenoid Driver Circuit.

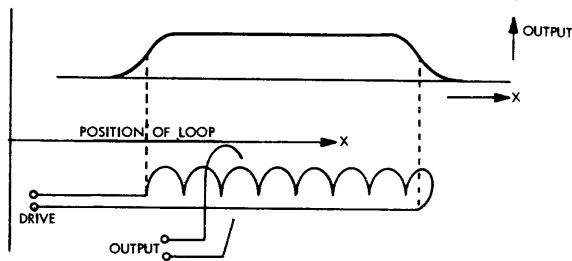


Figure 8. Air Cored Solenoid and its Relative Coupling to a Loop.

quite unobjectionable. Figure 9 shows a solenoid constructed in this way with a middle rod of low permeability and the two end rods with higher permeability. It may be interesting to note that this technique bears a similarity to a triple-tuned bandpass filter with a broadly tuned middle section and more sharply tuned end sections. This technique was applied in the memory described.

The question may reasonably be asked why it is necessary to use all these techniques and to pay the price of higher drive currents instead of using a closed toroidal-like structure. The reasons are as follows:

1. A very elongated thin-legged closed structure is likely to have high leakage flux between its long members when the leakage reluctance becomes comparable to the path reluctance.
2. The need for an easy data exchange would require split cores whose alignment would need to be carefully maintained. Since this memory is intended to applications where data is frequently changed, precisely mated surfaces would require critical protection and very precise alignment mechanisms.

Further, since the operating pulse widths are short because of access and cycle time requirements, the maximum drive currents remain at acceptable levels.

To give the structure mechanical strength, the ferrites are inserted into a phenolic-paper tube and appropriately glued in, and the windings are laid into a shallow threaded groove cut on the outside of the tube. The entire assembly is then appropriately varnished or epoxy coated.

## Solenoid Drivers

The solenoid drivers are designed to supply a 16 volt, half microsecond long pulse to an inductive load of about twenty microhenries, with ample margin. They are also designed to withstand an inductive overshoot equal to the drive pulse, or about 32 volts peak.

As was mentioned earlier, each solenoid pair is connected in parallel, as shown in Figure 7, so that the solenoids in each pair always have opposite polarities. A second consideration is that a "one" input should drive the pair one way, and a "zero" the other. In earlier designs, a transformer with two input drive windings of opposite winding polarity, one from each of two separate drivers, coupled to an output winding that was connected to the solenoid load. When "one" was asserted, one switch closed and drove the transformer, and when a zero was asserted, the other switch and winding drove the transformer, thereby generating opposite drives on the output winding for the two states. Unfortunately, the transformers were relatively bulky and somewhat inefficient.

Instead, each driver solenoid was cut with 2 grooves instead of one to allow a bifilar winding, and one winding on each solenoid was driven for a "one," and the other for a "zero." Both of the respective pairs of windings on the solenoids were connected in parallel, in order to maintain opposite magnetic polarity on the solenoids for either drive.

The drivers themselves are arranged to be controlled by the input address code. The state of the address turns on either of two currents  $I_1$  or  $I_0$  in the driver, which flow as soon as the input address code bits are set up. The currents are shunted to  $-18$  volts by transistors

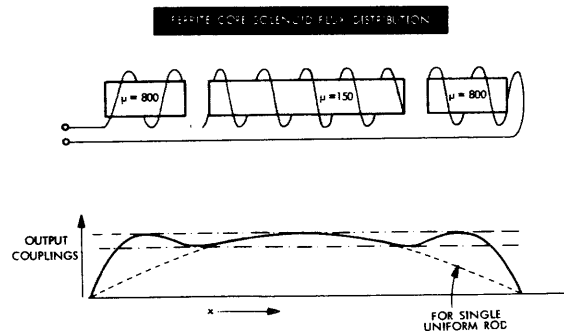


Figure 9. Ferrite Cored Solenoid and its Relative Coupling to a Loop.

$T_1$  or  $T_2$  which are saturated continually except during the drive pulse.  $T_1$  and  $T_2$  turn on as soon as power is applied to form a fail safe timing circuit for the solenoid drivers.  $T_1$  and  $T_2$  can only be shut off by a negative drive pulse to their bases, and the resistance-capacitance time constant in their base input circuit is made long enough to allow them to open only slightly over the maximum desired drive pulse width. When either  $I_1$  or  $I_2$  flows, the opening of  $T_1$  and  $T_2$  shunts the current into  $T_3$  or  $T_4$ . This turns on one drive transistor, causing the required drive voltage pulse. At the end of the drive pulse,  $T_1$  and  $T_2$  again saturate, rapidly shutting off the conducting transistor with a low impedance drive.

The fail-safe circuit of  $T_1$  and  $T_2$  is needed since the D. C. resistance of the solenoid circuit is very low, and the drive transistor would soon be destroyed if allowed to stay on.

It should be noted, that when  $T_3$  is switched on, the transformer coupling between the two solenoid windings causes the collector of  $T_4$  to go to double the supply voltage. Similarly, when  $T_4$  is switched on,  $T_3$ 's collector rises. For this reason alone, the inductive overshoot clamp diodes on  $T_3$  and  $T_4$  must be tied to almost twice the supply voltage. Thus, the inductive transient causes an overshoot approximately equal to the drive pulse both in amplitude and pulse width. (Volt-time areas are equal.)

The large overshoot transient is desirable because it shortens the transient duration, but tends to produce other unwanted transients. These transient currents in the data planes occur after the data has been strobed, hence they do not affect data read-out, but they do require a few microseconds to settle, thereby lengthening the cycle time. These transients, and means to suppress them, are discussed later.

#### Correlation Selection Techniques

The addressing solenoid drive causes a parallel correlation operation on all the stored addresses on each respective plane. Figure 10 shows the outputs of one selected plane, and three typical outputs of unselected planes, the former being the positive drive pulse. A means must be provided to uniquely separate the selected plane by allowing a current to flow in

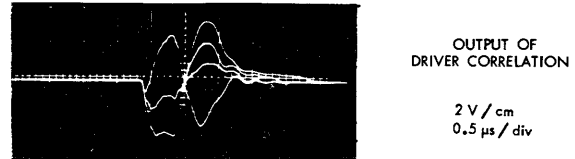


Figure 10. Correlation-Coder Outputs.

its path which is much larger than the linear sum of ALL the unselected data plane currents.† An ordinary silicon epitaxial diode has a forward-to-reverse resistance ratio of over a million to one. Its capacitance of a few picofarads in series with the data plane impedance of perhaps ten microhenries and three ohms allows only an extremely short transient current in the unselected planes, and the sum of all these currents is still far exceeded by the select current over the drive pulse interval.

The outputs of the stored address code correlations may be represented by a distribution as shown in Figure 4 if no bias is applied. If a four or five microsecond memory is desired, a simple diode may be placed in series with the path on each data plane and additional solenoid drive bias of minus nine units may be added to shift the distribution to that shown in Figure 11A. The diode characteristics in Figure 11B will allow current to flow in that addressed plane whose output is to the right of the origin.

Unfortunately, after the data has been strobed out, and after the drive pulse is terminated, the overshoot reverses the distribution so that the unselected planes then go into conduction as shown in Figure 11C. As a result, current flows in many planes for a few microseconds with a period determined by plane inductance, resistance and diode conduction voltage drop. Due to less than perfect coupling from solenoids to planes, the sum of the currents is far less than would be expected in a good transformer, hence the solenoid flux collapses rapidly.

The technique used to prevent the flow of current during the overshoot period is as fol-

† This statement is actually a simplification intended to clarify. Actually, the differential of the desired current flow over the interrogation period must greatly exceed the sum of all the unselected differentiated currents. Since the small, unselected current transients are very short, their positive and negative differentials essentially cancel out during the first fraction of the driver pulse period.



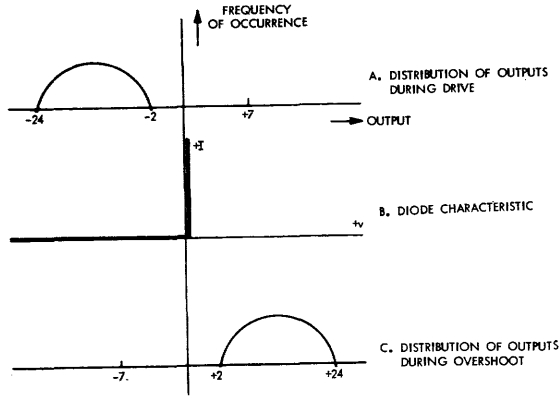


Figure 11. Distribution of Outputs for Given Drive and Diode Characteristic.

lows. First, the drive bias is changed so that the distribution of outputs is that shown in Figure 12A. Second, a zener diode with a breakdown voltage  $V_1$  is placed in series with the diode. Its voltage is chosen such that the sum of the diode forward conduction voltage drop and the zener breakdown voltage drop equals a voltage which is two units higher than the right limit of the distribution at +11 as is shown in Figure 12B. Now, during drive, only the selected current flows just as before. However, during the overshoot period, the distribution is such that no current flows at all, as is shown in Figure 12C. Hence, a new memory cycle can begin in less than a microsecond. The oscillograph in Figure 13 shows the voltage sensed by a solenoid in an array loaded with fifty planes using simple diodes, and the oscillograph in Figure 14 shows a similar output due to fifty planes utilizing the diode combination.

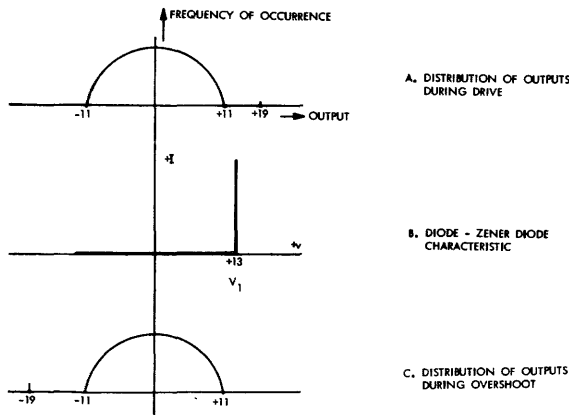


Figure 12. Distribution of Outputs for Given Drive and Diode—Zener Diode Characteristic.

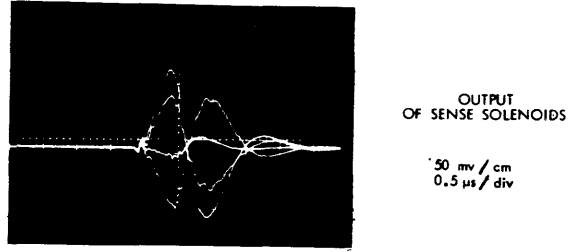


Figure 13. Worst Case "Ones" and "Zeros"—Diode Detector.

The diodes are ordinary planar epitaxial types, but "gold bonded" germanium point contact types work almost as well. The zener diodes most commonly available have relatively high capacitance since they have large junctions designed for high dissipation. The emitter-base junctions of a small silicon transistor such as a 2N706 have very sharp and uniform zener-breakdown voltages, and low junction capacitances on the order of a few picofarads. These transistors, used as zener diodes, may be seen in the photograph of the memory in Figure 15 hanging from the sides of the data planes. However, the diodes are mounted integrally on the data plane in small holes, staggered around the planes' peripheries as shown in Figures 2 and 16. In later units, where the zener diodes are needed, the diode-zener diode would be in one component mounted as the diodes are now mounted as shown in Figure 16. The dual component is a commonly built one, and is in essence a transistor with no base lead connection, in which the usual collector junction is

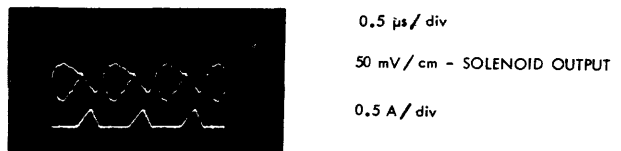
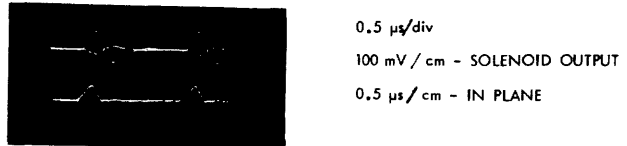


Figure 14. "One" and "Zero" Outputs for  $3\mu s$  and  $1.5\mu s$  Cycle Time with Zener Diode—Diode Detector.

the diode and the emitter junction is the zener diode. Dozens of silicon epitaxial type transistors were found to have the desired characteristics, shown in Figure 12B, hence no problems are anticipated in obtaining these "integrated" circuits.

#### Data Readout

The device described up to this point of the paper constitutes a substitute for a pair of connections to each plane, with appropriate gating and drive circuitry. Little has been mentioned as to how the current in the plane is detected and used. The organization for data readout is the subject of the following paragraphs.

The current ramp in the data plane constitutes a primary drive to a multitude of long sense solenoids or transformers. Each loop or enclosure of a solenoid on a driven plane is a primary coupled to the solenoid secondary, or a "one" and each bypassed solenoid constitutes a "zero," when that plane is interrogated. A "one" causes a voltage output of one polarity, and a zero causes a smaller voltage of opposite polarity.

Figure 17 is an oscillograph that shows the drive current in the data plane, and Figures 13

and 14 the outputs of a number of "ones" and "zeros" from the respectively driven solenoids.

Obviously, all bits on a plane are emitted in parallel, hence some gating is usually desirable to connect only the desired subset, or word, to the output amplifiers. The method of achieving this is quite simple. One terminal of each solenoid in a word group is tied together with the similar terminals of the solenoids in that group. This common terminal becomes the word-select control terminal. All the other word-groups of solenoids are similarly tied together. This is shown in Figure 18.

In each word group, the other terminal of each solenoid representing each bit of the word is tied to all the other respective solenoids in the other words by means of diodes. All the common "word line" terminals are biased so that their respective diode switches are back-biased except for those of the addressed word. The diodes on the addressed word's solenoids are forward-biased before the data plane is pulsed, hence effectively connecting the addressed solenoids to the preamplifiers before the interrogating pulse. Since a fraction of a microsecond is needed for currents to change and for diodes to switch, this word preselection procedure is timed ahead of the main pulse.

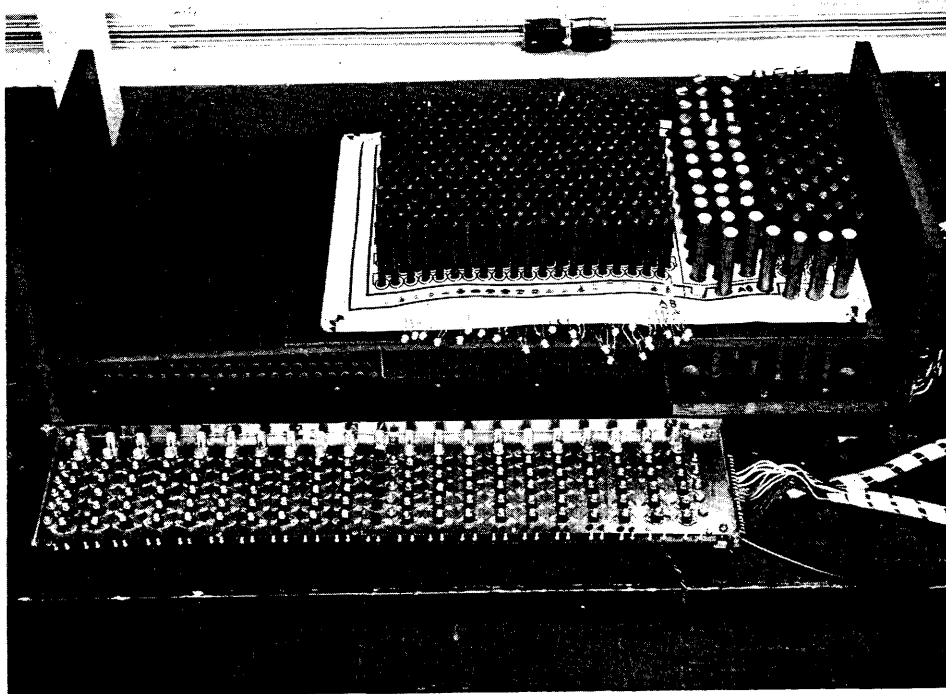


Figure 15. Photograph of System.

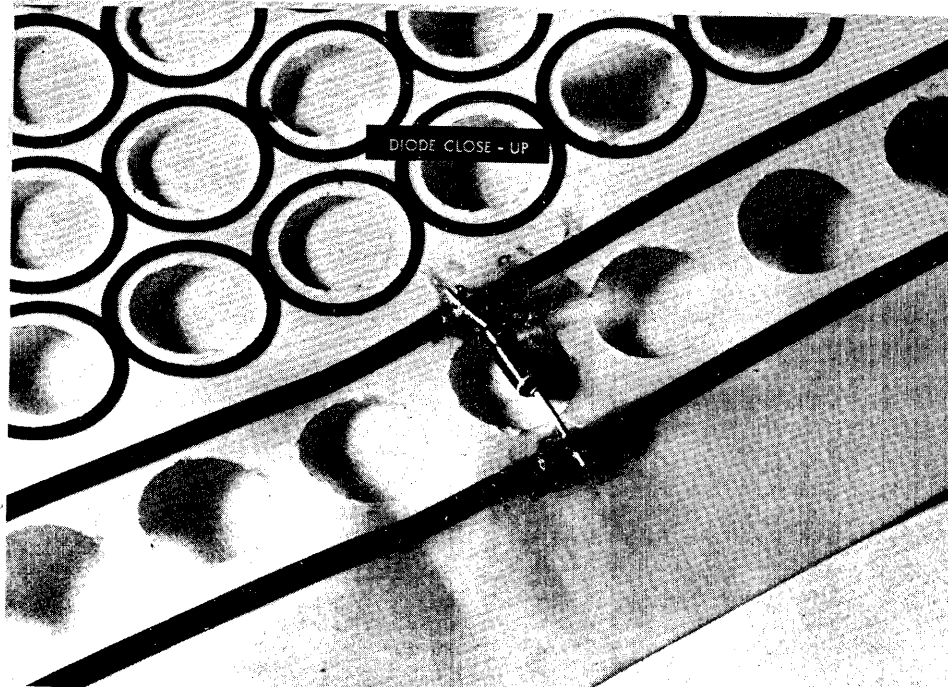


Figure 16. Diode Mounting Close-Up.

The solenoids themselves for most applications are air-cored. These are simply phenolic paper tubing which is wound with a helix of thin copper wire, typically, number 30 wire gauge. After winding, it is suitably coated to protect it with either varnish or epoxy coating.

For applications where higher outputs are desired, ferrite cored solenoids, smaller in diameter than those in the drive array (quarter inch diameter instead of three eighths) may be used. To minimize pattern sensitivity, they should either be paired and connected in series or spaced far apart.

#### Output Circuitry

The diodes used at the output of the solenoids could be matched to those on each bit line to allow the use of a direct coupled, single ended

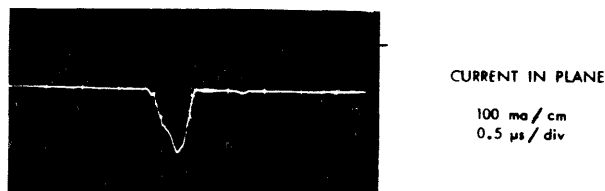


Figure 17. Drive Current in Plane Due to Hamming Correlation.

system. However, for signals below 100 millivolts, typical of the simpler air-cored pick-up solenoids, capacitor coupling is indicated. Hence the gated signal is amplified in a linear amplifier, along with any pedestal shifts due to word-line switching, and voltage restoration is applied. If the access time is to be short compared to the pulse width, keying or gating of the restoring voltage is required. If the access time is long compared to the pulse, an ordinary resistance-capacitance high-pass filter is adequate.

The waveforms of the connectionless memory shown in Figures 13 and 14 require strobing for reliable operation, and it is timed to occur just before the end of the drive pulse.

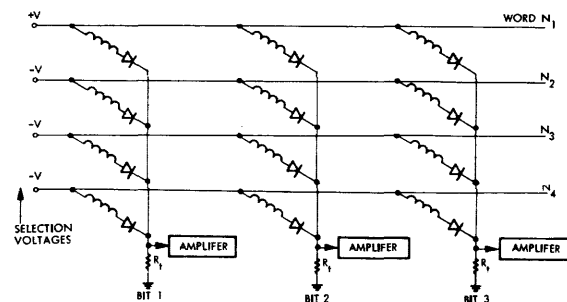


Figure 18. Word Pre-Selection Matrix.

In the limit, with a pick-up array of paired ferrite-cored solenoids, which may deliver outputs of as much as a volt, output flip-flops could be strobed directly from the solenoids. In a more typical case, one or two transistors can be used for amplification, and another for slicing and strobing. For increased speed, a two-stage amplifier followed by the voltage restoring switch, followed by a slicer and output stage is desirable, and the memory described here used this system and is shown in Figure 19. It should be emphasized, that in all cases, the amplifiers were single-ended and differential amplifiers were not required.

#### Data Planes

The data planes in the earliest work were simply thin wire, wound on plastic sheets with small bobbins attached. However, this technique did not allow for quick and easy exchange of individual circuits paths or planes. Copper-clad Mylar was soon found to be applicable to the requirement.

The tooling technique involves accurately drawing the printed circuit layout, drilling a master template which fairly accurately matches the layout, and having a stainless steel mesh "silk screen" fabricated from the printed circuit layout.

To avoid critical alignment and fabrication problems, copper path widths are made about 0.040" wide, distances between closest conductors are also 0.040" and the closest a copper path passes to a hole is nominally 0.060". The solenoid array base plate and the data planes themselves are drilled through the same template, hence with only modest care, alignment is no problem. The holes in the planes are about 0.1" larger than the solenoids; hence they fit quite loosely.

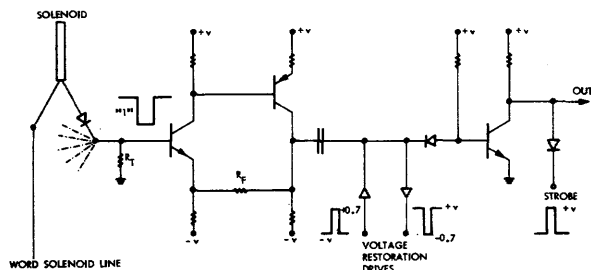


Figure 19. Pre-Amplifier.

All data planes contain both alternate paths around the solenoids. In the laboratory, the data is entered by simply scraping off the etch resist before the planes are etched. The result of this operation can be seen in Figure 2. A machine has been constructed that mills away the copper path, under control of an ordinary punch card, at the rate of six bits per second, and this is shown in Figure 20, and can be seen operating on a large 1428 bit plane.

The fabrication of the planes themselves is straightforward. Silk-screening, mass drilling under the template, data insertion and etching comprise the laboratory process. Screening, drilling and etching are the large scale process with subsequent data insertion by punching, scraping or milling on the aforementioned machine. In the field, the copper paths can be severed with a knife.

The material used mostly so far has been one ounce copper (0.00135") on 2 mil Mylar (0.002") which has a total thickness of about 0.004", allowing well over two hundred planes per lineal inch along the solenoid.

#### Mechanical Considerations

In a memory in which changes are not frequent, it is simple enough to slide the planes on or off individually. For greater convenience, many planes can be prealigned to thin base

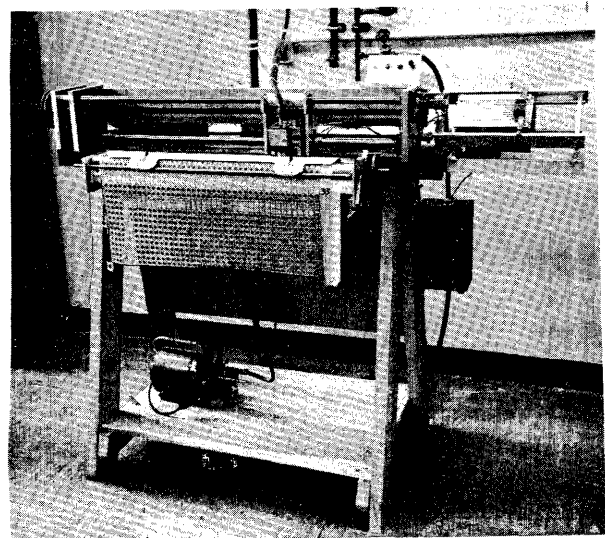


Figure 20. Punch-Card Controlled Cutter for 1428 Bit Planes.

plates, covered with another thin sheet, and handled as magazines. In either case, the only disadvantage is that in removing data behind or below other data, which is inaccessible until the covering data is removed.

A "file cabinet" like mechanism has been designed to avoid this problem. In this technique, shown in Figure 21, the solenoid array is fastened to the stationary back panel behind a drawer section, with the solenoids extending through the drawer when the file is closed. The solenoids are withdrawn from the drawer when the file is opened. Thus when the file is open, magazines can be removed or replaced individually, and closing the file mates the data containing drawer and the solenoid array. A unit such as this is now under construction.

The magazines themselves contain one to two hundred planes each, which are aligned to the magazine by several pins.

Changing one data plane requires opening of the drawer, opening of the magazine much as a loose-leaf notebook, finding the "page," and exchanging it.

In some operations, entire programs or tables would be stored or shipped in magazines, and when the data was to be used, that magazine simply dropped into the drawer, and the drawer closed.

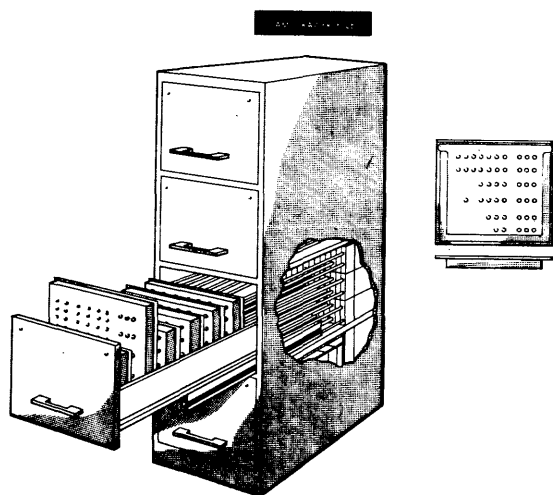


Figure 21. "File Cabinet" Mechanism.

## Conclusions

At present, a 360,000 bit memory has been built and about a hundred planes placed in it. The signal degradation in increasing the number of planes from a few to a hundred was very minor, hence extrapolation to full capacity appears justified. The unit built used a Hamming code and has operated well. However, for the small increase in complexity, the Golay code more than doubles the selected current, hence a unit now under construction will use that code.

A practical limit to this technique is about 4,000 data planes since the practical but powerful Golay code is extendable to a 23-12 code. Physical dimensions also suggest that a stack of 4,000 planes, or about twenty inches, is a reasonable limit. Lengthening the solenoid causes linearly increased driver voltage requirements, and they show practical limits which are equivalent to between 2,000-4,000 planes, with presently available transistors.

The limits on the number of bits per plane is also between 2,000 and 4,000, imposed by the limits of the correlation voltage output drives versus the data planes' path resistance and inductance as well as the propagation time in the data planes' paths.

In summary, the size limit per module is about  $10^7$  bits, and  $2 \times 10^6$  bits appears easy to reach.

As to access and cycle times, the limits vary with module capacity, and for a 1 megabit memory 0.5 microsecond access and 1 microsecond cycle probably are close to the limit, and twice this is relatively straightforward. Decreases in memory capacity, particularly data plane bit capacity, should be followed linearly by access and cycle times down to a limit of about 0.25 and 0.5 microseconds respectively. Below this, directly connected data plane memories should be considered.

The cost of these memories is low but highly variable since the associated electronics, input-output buffers, coder, sense amplifier and word line selectors set up an "overhead" that is almost invariant over a range from under a hundred to a thousand planes, and goes up very slowly beyond that. The cost of the array, even

in hand-made versions, amounts to only a small fraction of a cent per bit capacity. The data plane cost is between a quarter and one cent per bit, including data entry and all fabrication; and a full electronics complement can add anywhere from a quarter to two cents a bit depending on memory size and speed. In summary, this memory has a bit cost ranging from half a cent per bit for large memories to a few cents per bit for relatively small ones, with some downward revision when produced in quantity.

It is believed that this type of memory will find application in digital computers where large, infrequently changed blocks of data are used, and other applications where the memory's rapid data change capabilities allow it to be used as an input device as well.

#### ACKNOWLEDGMENTS

The author would like to acknowledge the valuable suggestions and discussions with many of the members of Sylvania's Applied Research Laboratory and particularly those of Messrs. Stephen Gray, Benjamin Eisenstadt, Allan Snyder, Gerald Ratcliffe; Doctors Donald Brick, Richard Turyn and Paul Johannessen and our Director, Dr. James Storer.

#### REFERENCES

1. PICK, G. G., GRAY, S. B., and BRICK, D. B., "The Solenoid Array—A New Computer Element," *IEEE Transactions on Electronic Computers*, Vol. EC-13, Number 1, February 1964.
2. YOUNKER, E. L., et al., "Design of an Experimental Multiple Instantaneous Response File," *AFIPS Conference Proceedings*, Vol. 25, Washington, D. C., pp. 515-527, April 1964.
3. KUTTNER, P., "The Rope Memory, a Semi-Permanent Storage Device," *AFIPS Conference Proceedings*, Vol. 24, Las Vegas, Nevada, pp. 45-58, October 1963.
4. BUTCHER, I. R., "A Prewired Storage Unit," *IEEE Transactions on Electronic Computers*, Vol. EC-13, No. 2, April 1964.
5. ISHIDATE, T., YOSHIZAWA, S., and NAGAMORI, K., "Eddycard Memory—A Semi-Permanent Storage," *Proc. of the Eastern Joint Computer Conference*, Washington, D. C., December 1961, pp. 194-208.
6. FOGLIA, M. R., MCDERMID, W. L., and PETERSON, M. E., "Card Capacitor—A Semi-Permanent Read-Only Memory," *IBM J. Res. and Dev.*, Vol. 68, p. 67, January 1961.
7. MINNICK, R. C., and HAYNES, J. L., "Magnetic Core Access Switches," *IEEE Transactions on Electronic Computers*, Vol. EC-11, No. 3, June 1962, pp. 352-368.
8. TURYN, R., "Some Group Codes," Internal Applied Research Laboratory Note Number 404.
9. DONNELLY, J. M., Card Changable Memories, *Computer Design*, Vol. 3, No. 6, June 1964.



# A $10^5$ -BIT HIGH-SPEED FERRITE MEMORY SYSTEM – DESIGN AND OPERATION

*H. Amemiya, T. R. Mayhew, and R. L. Pryor  
Radio Corporation of America  
Camden, New Jersey*

## INTRODUCTION

With the advancement of computer technology in recent years, the demand for a very-high-speed memory has greatly increased. Scratch-pad memories of smaller than 100 words with cycle times faster than 500 nanoseconds are commonly found in computers on the market. However, larger memories of the same speed range are not yet commercially available, due to the fact that the problems in building a large memory are much more complicated than those in building a small memory. These problems center around the transients generated in the digit sense system.

In order to understand these problems, a 1024-word 100-bit memory was built. The storage cells consist of ferrite cores (30 mils O.D., 10 mils I.D., 10 mils thick) used in a two-core-per-bit arrangement in a linear organized array. In order to simplify the core-threading work, only two conductors per core are used; one conductor is plated, leaving only one wire to be threaded.

As a new approach, digit lines are treated as a set of mutually coupled parallel transmission lines and are terminated accordingly. Recognition that different modes of wave propagation exist on digit lines was probably the most important step in obtaining the high-speed operation of the present memory.

The word drive system uses a square selection matrix with transformer coupling to in-

dividual word lines. This arrangement reduces the noise voltages that are coupled into the memory stack from the word drive system.

The sense amplifier is a differential amplifier in which a delay line is used to minimize dc imbalances and level shift. A tunnel-diode strobe circuit is used to provide low-level thresholding and high-speed operation.

Some portions of the electronics of the memory system are located very close to the memory stack. Interconnections are made either by cables or by microstrips. The use of these techniques has resulted in a memory cycle time of 450 nanoseconds for the memory system.

## MEMORY CELL OPERATION

Linear selection (word-organized memory) and partial switching<sup>1, 2, 3, 4, 5, 6, 7</sup> are the two techniques commonly employed to achieve a cycle time of one microsecond for a high-speed ferrite memory. Linear selection offers the advantage that read currents of large amplitude (limited only by drivers) can be used to increase speed. This method contrasts with coincident current selection, where read currents are dictated by the threshold characteristics of the ferrite cores used.

As the memory speed is increased by narrowing the width of the write and the digit pulses and subsequently the width of the read pulse, a point is reached where two-core-per-bit operation becomes necessary. There are two reasons



for this: the sense signal generated on reading a ZERO becomes large as the rise time of the read pulse is decreased; and the sense signal difference between reading a ONE and reading a ZERO becomes small because the digit pulse in the presence of the write pulse switches only a small fraction of the core irreversibly. Figure 1 illustrates these reasons qualitatively. The ZERO signal is due to reversible flux change, and the ONE signal is due to both irreversible and reversible flux changes, with the former contributing to the net signal difference between a ONE and a ZERO.

Two-core-per-bit operation provides a means of cancelling the reversible flux contribution to the total sense signal. There are many schemes employing two cores per bit,<sup>8,9,10</sup> but the one used in this memory is shown in Figure 2. Here, each core is threaded by two conductors, one in the word direction and the other in the digit direction. When writing, both core A and core B of the same bit pair receive a write pulse. In addition, either core A or core B receives a digit pulse depending on the information being written in. When reading, a read pulse is applied to both core A and core B in the direction opposite to that of the write pulse. Digit pulses are always applied through the cores in the direction which is the same as that of the write pulse, because the digit disturb threshold of a core becomes much lower if opposite-polarity digit pulses are used.<sup>7,9,10</sup>

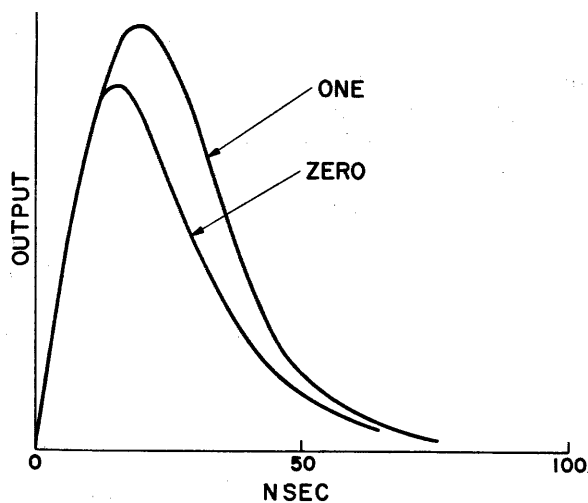


Figure 1. Sense signals of one-core-per-bit memory at increased speed.

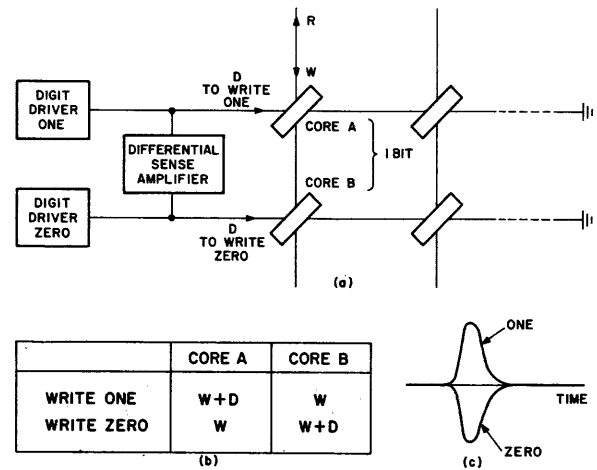


Figure 2. Two-core-per-bit scheme: (a) basic read-write scheme, (b) magnetization applied to cores when reading, (c) net sense signals.

The sense signals generated at core A and core B are added differentially in a differential sense amplifier, where the signal due to the reversible flux change is cancelled. Therefore, only the net signals as shown in Figure 2(c) reach the threshold circuit of the sense amplifier.

This two-core-per-bit scheme has the following features:

1. Bipolar sense signals provide more reliable sensing compared to a unipolar sense signal.
2. Word line impedance is constant regardless of the information pattern because each bit (a pair of cores) presents a constant impedance to word pulses even if a ONE or a ZERO is stored.
3. Read and write pulses may have loose tolerances.
4. Balanced digit lines that are paired for one bit location offer a possibility of controlling wave propagation inside a memory stack. This point will be described in more detail later.

The ferrite cores used in this memory have an outer diameter of 30 mils, an inner diameter of 10 mils, and a thickness of 10 mils. The operating conditions are shown in Table I. A test has shown that the worst-case disturb pattern changes the sense signal by less than 10 per cent.

Table I  
OPERATING CONDITIONS OF THE FERRITE CORES

CORE DIMENSIONS	30 mils O.D., 10 mils I.D., 10 mils thick			
DRIVE PULSES	AMPLITUDE	RISE TIME	FALL TIME	WIDTH (50% point)
READ	630 ma $\pm 5\%$	80 nsec	30 nsec	80 nsec
WRITE	220 ma $\pm 5\%$	40 nsec	40 nsec	80 nsec
DIGIT	70 ma $\pm 3\%$	30 nsec	30 nsec	75 nsec
SENSE SIGNAL FROM CORES			$\pm 50$ mv	
KICKBACK VOLTAGE WHEN READING			0.25 v/bit	

WAVE PROPAGATION IN THE MEMORY STACK AND TERMINATIONS

It is a very basic requirement that a memory system must be able to store any information pattern desired at any word location. Since some words are located close to the digit drivers and sense amplifiers whereas others are located far away from them, it is required that digit lines must be able to carry digit pulses and sense signals without distortion. These requirements make it essential that the wave propagation inside a memory stack be well understood.<sup>11,12</sup> The problem is complicated because many digit lines are parallel for a considerable distance and because many word lines cross the digit lines at right angles, with ferrite cores at the intersections. A relatively simple mathematical analysis of this structure can be made if one assumes that the delay on the word lines is zero. Then, the presence of word lines may be considered as contributing only to the coupling between digit lines. With this assumption, the problem of two-dimensional wave propagation changes into that of one-dimensional wave propagation on multiple parallel transmission lines with mutual coupling. The mutual coupling now consists of two parts, namely, the

inherent coupling due to digit lines running in parallel and the coupling due to word lines.

To fulfill the requirement that digit lines carry digit pulses and sense signals without distortion, it is necessary that digit lines be lossless and that there be no interference among waves propagating on separate digit lines. The first condition is met approximately by a memory stack. The second condition is normally not satisfied because of mutual coupling. However, there is a wave mode that propagates on a pair of lines without interference, provided that a certain manipulation of coupling is made.

In Figure 3, line n and line n' belong to pair n, and line k is a line outside pair n. Assume that the coupling between line n and line k is made equal to that between line n' and line k. Then, the differential-mode propagation on pair n (i.e., simultaneous propagations of same amplitude but of opposite polarities on lines n and n') does not induce propagation on line k, because of cancellation effect. In other words, if digit lines are paired, each pair can have independent differential-mode propagation without interference, provided that equalization of coupling is made.\* The transposition method used in the stack to obtain equalization of coupling will be explained later.

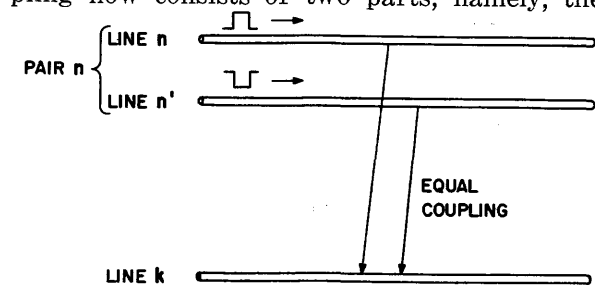


Figure 3. Equalization of coupling and differential mode.

Therefore, it is desirable to have all the propagations in differential mode. However, this is not the case with the memory being discussed here. In Figure 2(a) it is shown that digit lines are paired, a result of the consideration given above. But the digit pulses are not applied differentially because negative digit

\* Proof is given in the appendix.

pulses are not permitted. Since either of the two lines of a pair is always driven by a digit pulse, whether a ONE or a ZERO is being written in, digit pulse propagation can be regarded as a superposition of the differential-mode component and the common-mode component as shown in Figure 4, with current amplitude one-half that of the digit pulse on each line.

The differential mode component consists of a number of differential mode propagations, one for every digit line pair, that are made independent of each other by equalization of coupling. The common mode component obviously has no interference problem, since all the digit lines carry the same current pulses simultaneously. The information is carried by the differential mode component and not by the common mode component, as the latter merely serves as a fixed bias, independent of the information being written in.

Digit lines are terminated to eliminate reflections, since undesired reflections reduce system reliability and prolong cycle time. For instance, a proper termination is the only means to minimize the waiting time between writing and reading, as the digit pulses must be completely dissipated before sense signals can be detected.

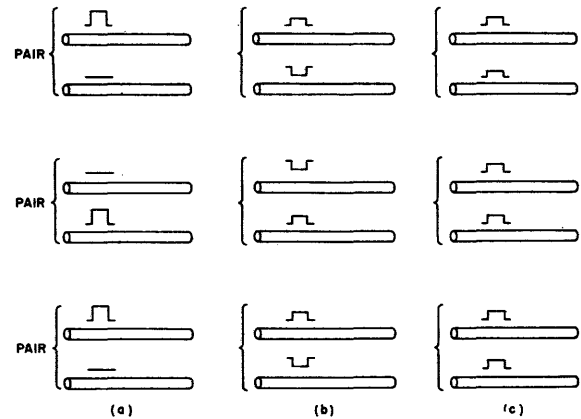


Figure 4. Propagation of digit pulses: (a) digit pulses on digit lines, (b) differential mode component, (c) common mode component.

The differential mode component and the common mode component require different impedance for termination. As shown in Figure 5, let  $Z_d$  and  $Z_c$  be the proper termination for the differential mode and the common mode, respectively.  $Z_d$  is smaller than  $Z_c$ , and the difference between the two is rather appreciable due to the effect of word lines. To terminate both modes, either a T network or a  $\pi$  network may be used, as shown in Figure 5 (c) and (d).

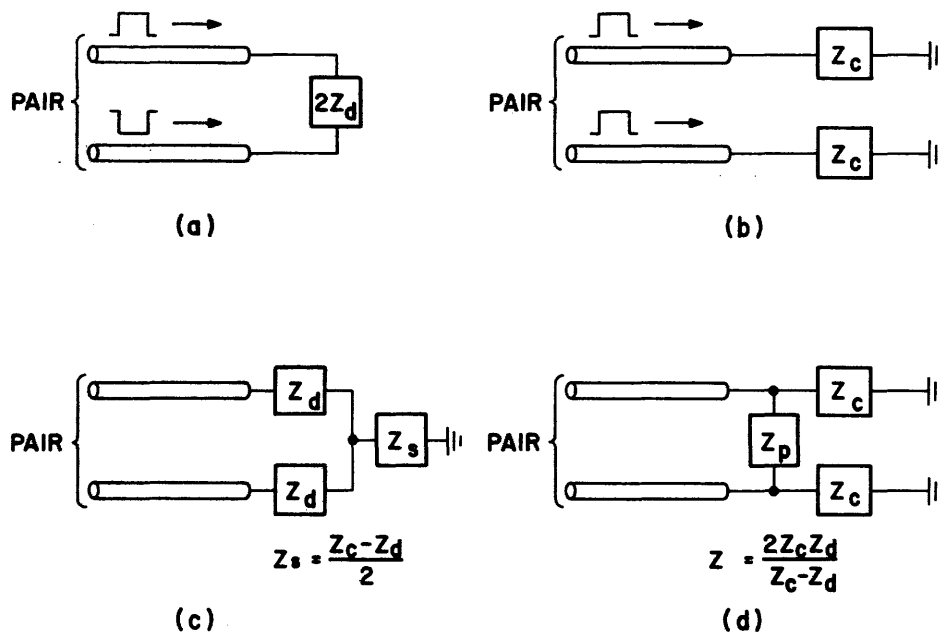


Figure 5. Digit line terminations: (a) differential mode termination, (b) common mode termination, (c) T termination (both modes), (d)  $\pi$  termination (both modes).

When reading, signals are sensed by differential sense amplifiers. It is noted that the net signal is propagated in the differential mode, and there is no interference problem. Since common-mode voltage is not sensed by the amplifiers, the common-mode termination is less critical than the differential mode termination. The fact that the digit lines are terminated means that only one-half of the raw sense signal reaches the sense amplifier. This seeming disadvantage is far outweighed by the advantage of being able to control the wave propagation generated in the memory stack.

Since the actual memory stack consists of eight memory planes, digit lines are folded. Figure 6 shows digit lines unfolded in order to show the transposition details. This transposition method equalizes coupling between any two adjacent line pairs if transpositions are done at short intervals. Figure 6 also shows that the digit lines are terminated on both ends by T terminations and that digit drivers and sense amplifiers are connected to the mid-points of digit lines. This connection minimizes the digit

line delay measured from the driving and sensing point. Yet, the digit line delay across 1024 words of 40 nanoseconds requires two different timings for the read and write pulses. Digit pulses used have negative polarity and are applied through diodes. These diodes disconnect digit driver cables and digit drivers from digit lines to avoid loading the sense signals. The emitter followers are the first stage of a sense amplifier and work as impedance transformers. These diodes and emitter followers are mounted on the stack assembly.

The effect of the new termination method on the digit pulse waveform and on the digit transient will be shown later.

### DESIGN OF MEMORY STACK

In the memory, one of the two conductors that go through cores is a conventional wire and the other is a plated conductor. Figure 7 shows the plated conductor as well as how memory cores are assembled into a strip. Individual cores are first metallized by vacuum deposition and then inserted into a groove cut in the mid-

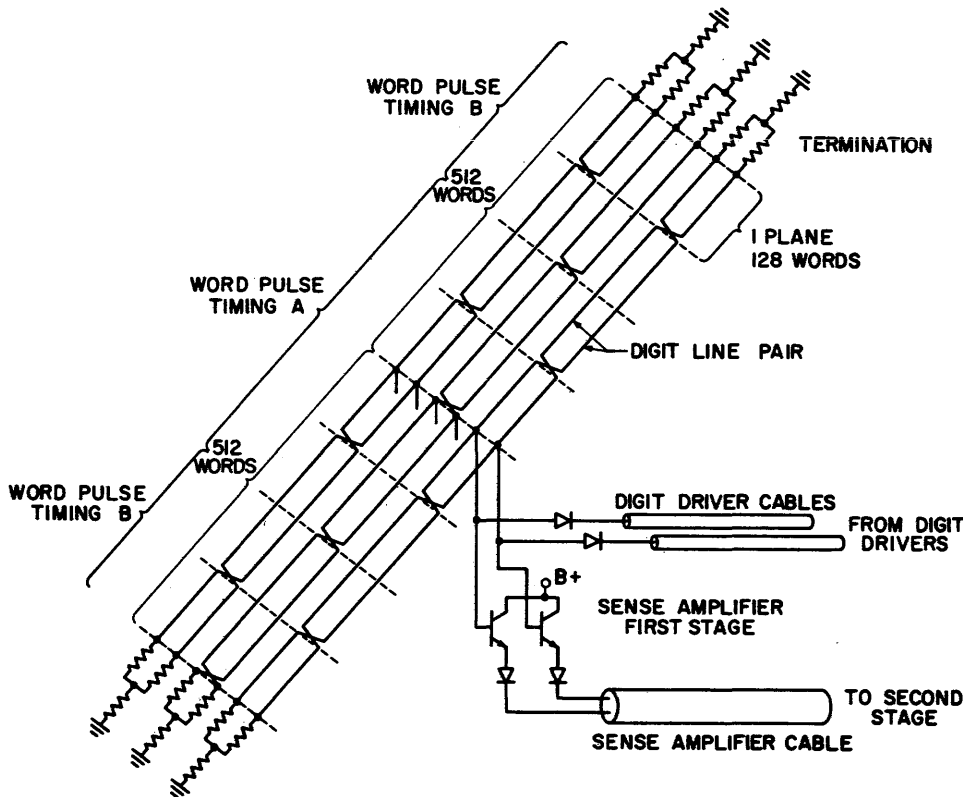


Figure 6. View of unfolded digit lines showing transpositions to obtain equalization of coupling and connections to digit drivers and sense amplifiers.

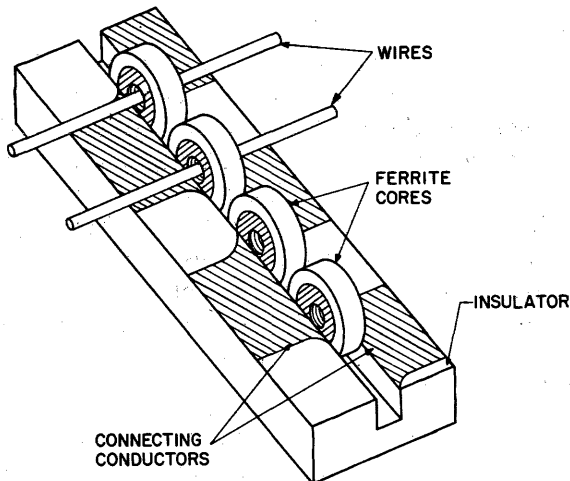


Figure 7. Ferrite core strip.

dle of an insulator strip, with connecting conductors already etched. Then the strip is electroplated to improve contact and also to lower the over-all resistance of the conductive path.<sup>9</sup> Since the connecting conductors on an insulator strip connect two neighboring cores on the same side, the resulting conductive path has a zig-zag pattern.

Each ferrite core strip contains 128 cores, with each memory plane holding 200 strips. Since plated conductors are used as digit lines, each memory plane contains 128 words of 100 bits each with 8 planes comprising a full memory stack. Plated conductors were used as digit lines because they permit pairing two neighboring conductors to form a bit pair. Such pairing is helpful in maintaining a good balance between the two lines of a pair and also to simplify transposition. If, on the other hand, the plated conductors are used as word conductors, it will be necessary to pair two nonadjacent digit lines because of the zig-zag pattern of the plated conductors.

As shown in Figure 8, a memory plane consists of a substrate and 200 ferrite core strips, of which 100 are mounted on the top surface and the remaining 100 on the bottom surface. This packaging technique causes the word lines to be folded into hair-pin shape to facilitate connection to the word drive system. Two opposing sides are used for word line connections; i.e., on each memory plane 64 word lines have their ends brought out to one side and the other 64 word lines to the other side. Ground planes are provided on the top and the bottom surfaces

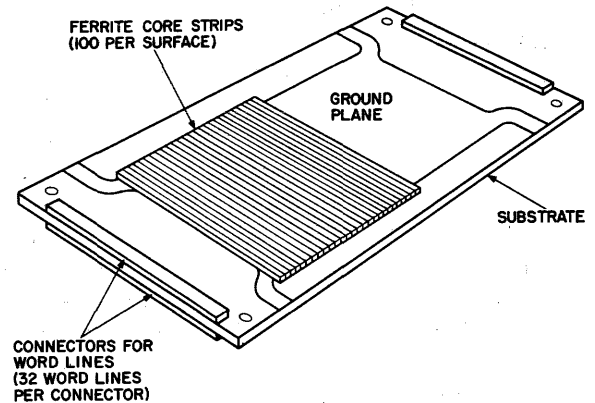


Figure 8. Memory plane.

of a substrate, over which ferrite core strips are placed. The ground planes are connected to the supporting structure on the four corners of the memory stack assembly.

When eight memory planes have been assembled, ferrite core strips are connected to form digit lines. Since each strip contains 128 cores, eight strips connected in series make up a full digit line. One group of 50 bit pairs (100 digit lines) is made up of core strips mounted on the top surfaces of the eight memory planes; another group of 50 bit pairs consists of core strips on the bottom surfaces of the memory planes. This packaging technique is shown in Figure 9. It is noted that these two groups have symmetry; i.e., (b) is obtained by rotating (a) 180 degrees. The digit system is divided into two groups to make the best use of the stack surface areas for external connections, which include 200 transistors, 400 diodes, 600 termination resistors and 300 cable connectors for the digit system. (See Figure 6.) Figure 10 shows the utilization of the memory stack surfaces for the digit and the word connections; all usable surfaces are being used. The top and bottom surfaces are actually the top surface of the top memory plane and the bottom surface of the bottom memory plane, and are not usable for external connections.

In the construction of the present memory, bit sense signal testing was done after each memory plane had been completed with core strips. Bad cores were then replaced. The resistance of the digit lines (plated conductors) across 1024 words was found to fall between 1.6 and 2.0 ohms.

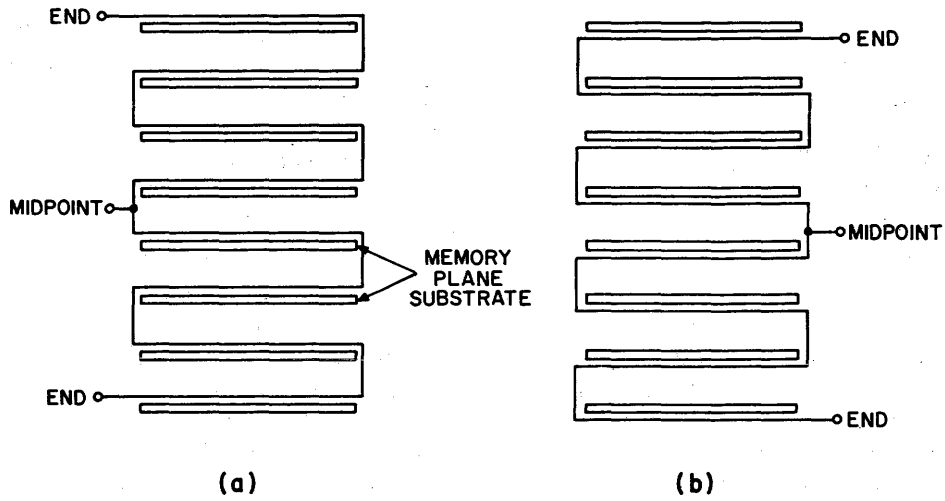


Figure 9. Methods of connecting ferrite core strips: (a) one group of 50 bit-pairs is obtained by connecting ferrite core strips on the top surfaces of all eight planes, and (b) the other group of 50 bit-pairs is obtained by connecting ferrite core strips on the bottom surfaces.

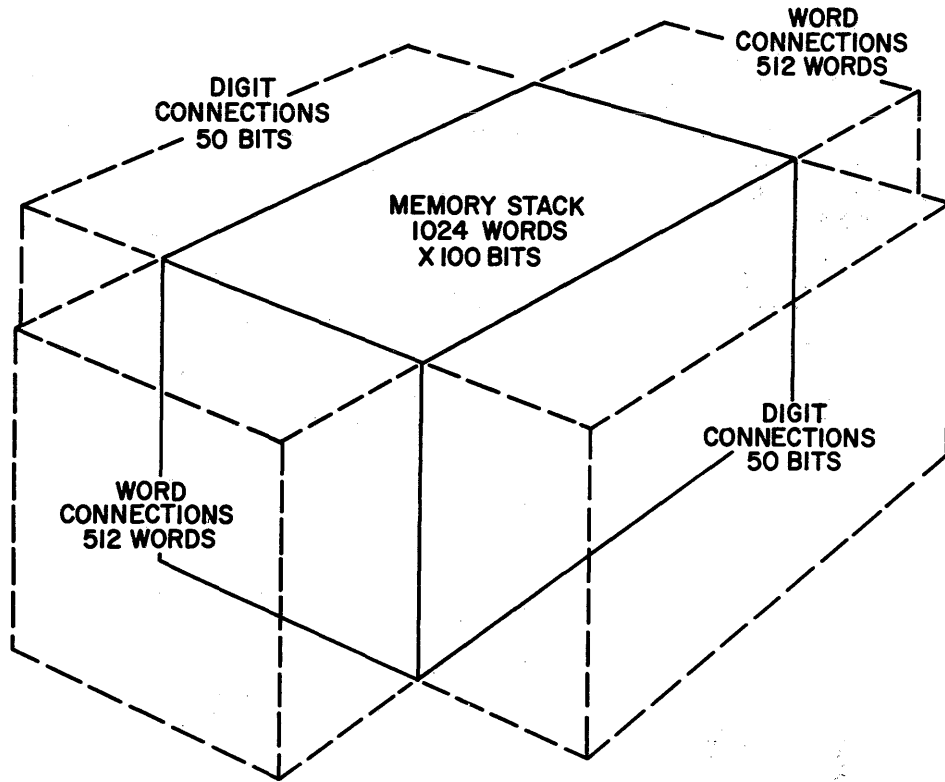


Figure 10. Utilization of the memory stack surfaces for external connections.

## ELECTRONICS FOR THE 1024-WORD MEMORY

Figure 11 is a block diagram of the memory system, which has four major portions:

1. Memory stack assembly,
2. Control system,
3. Word system,
4. Digit system.

The memory stack has been described. The control system generates and supplies all timing pulses for the drive system and for transferring data. The word system at the command of the control system supplies the proper read and write current pulses to a selected word for reading and writing information out of or into the memory. The digit system is used in a dual fashion: to provide sensing of the information stored; and to write back into the memory, simultaneously with the write pulse, formerly stored or new information. Parts of both the word system and the digit system are packaged on the memory stack.

## CONTROL SYSTEM

The control system is built of a logic building block, which has a typical two-level AND-OR logic delay of seven nanoseconds with fan-out of six. The system controls all the necessary timing pulses for each of three cycle types: 1) read cycle, 2) write cycle, 3) split cycle. The first two are standard for destructive random access memories, the first being the standard read-out operation which must be followed by regeneration while the information is still in the memory register. The second is the standard means of getting new information into the memory, the read half of the cycle being used only to clear the memory while the strobe pulse is inhibited. The memory register is loaded with the new information which is then written into the memory. The only feature which is unusual is the split cycle. The first command for this cycle generates only a read operation accompanied by a strobe of the sense amplifier. The retrieved information is available for processing but is not regenerated since the entire memory cycle has been temporarily suspended. When the con-

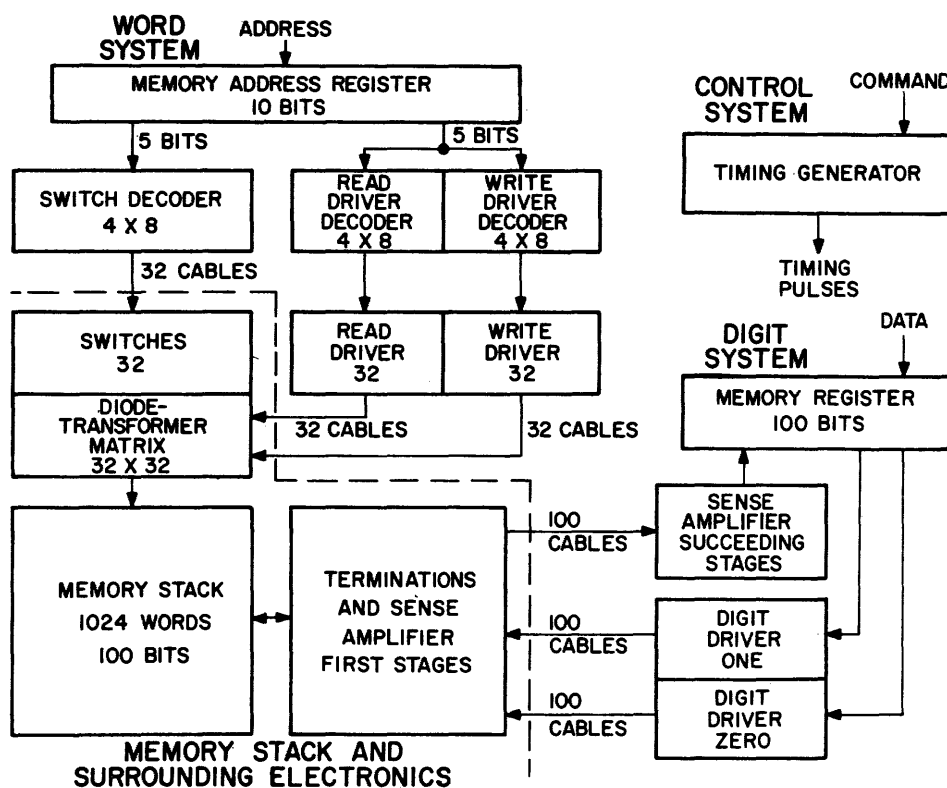


Figure 11. Block diagram of the memory system.

tinue command is given, the memory register is reset a second time to receive the newly processed information which is then stored in memory. Thus, the first half starts a conventional "read" cycle which stops itself in the middle, upon later command to continue as a "write" cycle after clearing the memory register. The time-saving features of this type of cycle are compatible with many of the common computer operations. The timing pulses generated by the control system for a read cycle are shown in their time relationships in Figure 12.

**WORD SYSTEM**

The word system is required, with the generation of minimum noise, to distribute a large read pulse, followed by a smaller write pulse of opposite polarity, to any of the 1024 words which happens to be addressed by the 10-bit address register. The bulk of this decoding is done in a bipolar diode matrix driven by 32 pairs of read and write drivers along one side

and 32 switches along the other side. The 1024 intersections of this main matrix are transformer-coupled to the 1024 word lines of the memory stack. The dc level of the word line is restored by a diode-resistor network in the secondary of the transformer. Without this network a dc level shift will appear, as the read current pulse is greater in amplitude and duration than the write current pulse. The circuitry of the main matrix is shown in Figure 13. Each of the drivers and each of the switches has its own preamplifier channel complete with an AND gate having one negative and one positive input. The complete read and write driver channels are shown in Figure 14, and the switch channel in Figure 15. These driver and switch channels are arranged in three 4 × 8 matrices. These matrices permit the selection of one of 32 switch channels and one each of 32 read drivers and write drivers to select any word and drive it.

The main problem encountered in designing a word drive system for a high-speed, high-bit-

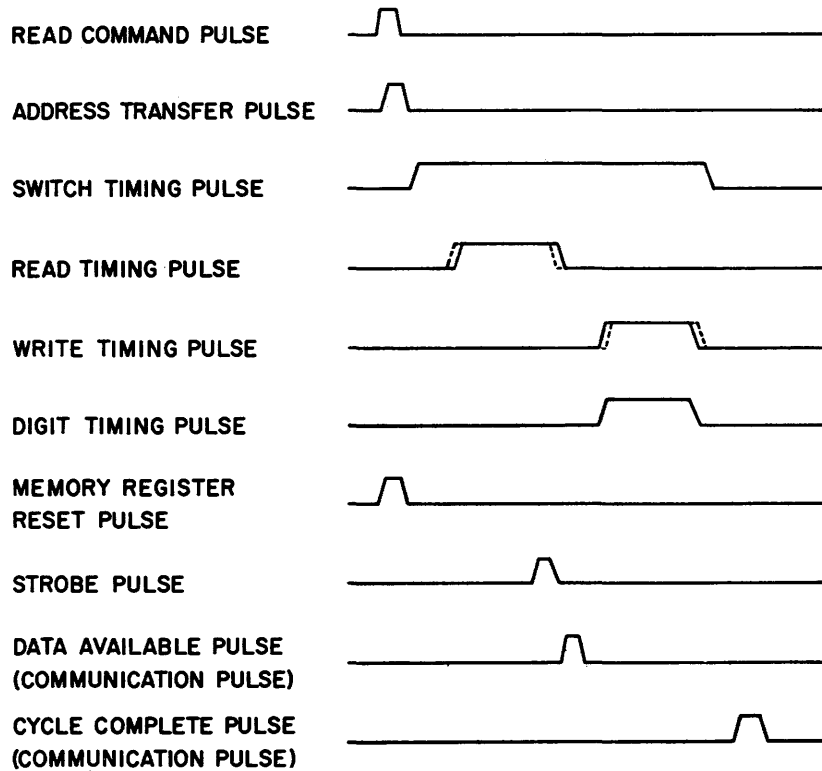


Figure 12. Timing diagram for a read cycle. Read and write timing pulses shift in time depending on the word address. (Solid lines show "Timing A" and broken lines show "Timing B").



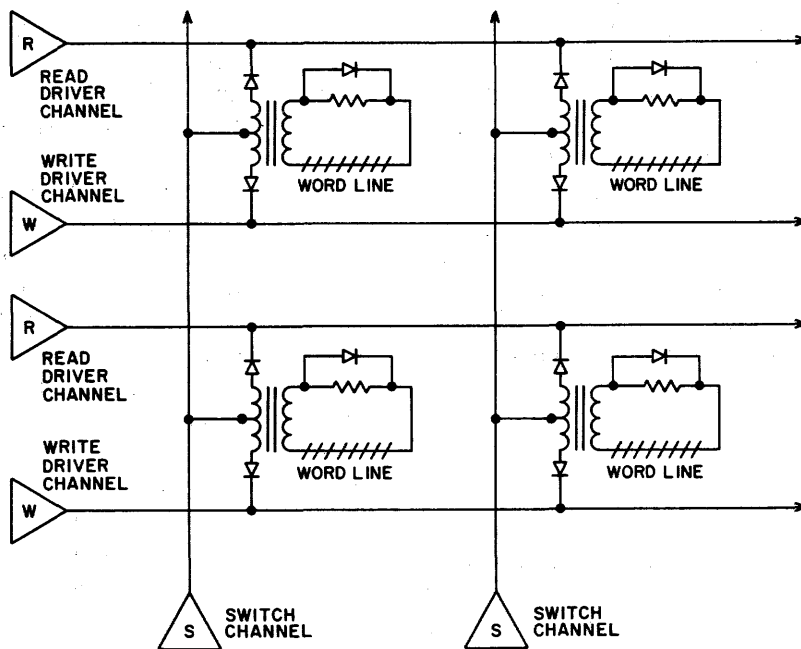


Figure 13. Portion of the 32 x 32 bipolar matrix.

capacity memory is that of minimizing the noise introduced into the stack. As the electronics are a significant cost factor, it is desirable to use a bipolar diode matrix performing selection with drivers and switches; such a matrix reduces the cost of the word-system elec-

tronics for a 1024-word memory about ten to one. It has been our experience, however, that with all the types of bipolar matrices that we can devise, severe switching transients are introduced on  $n$  lines of an  $n^2$  matrix when the switch selection is made. Moreover, it is found

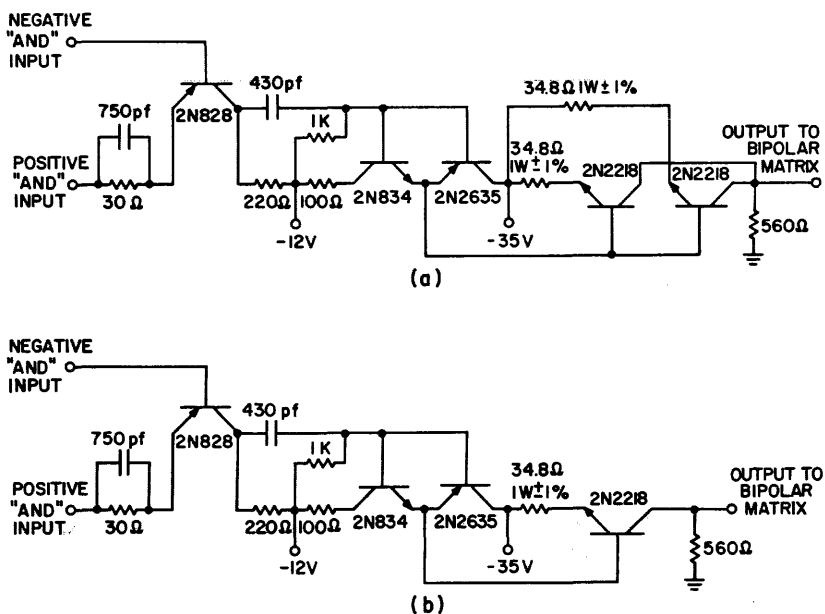


Figure 14. Word drivers: (a) read driver channel, (b) write driver channel.

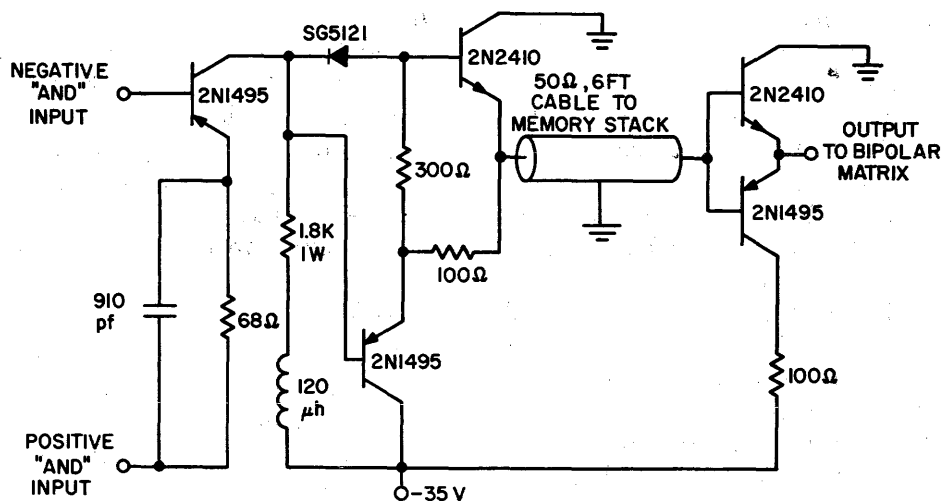


Figure 15. Switch channel.

that the characteristics of the memory stack, both with conventional core memories and with our partially automated fabrication techniques, show much tighter capacitive coupling between the network of word lines and the network of digit lines than can be made to exist between either of these networks and a ground plane. The result is a tendency for the conventional word selection matrix to introduce a very sizable common-mode noise onto the network of digit lines.

An analysis of the switch-noise injection procedure in the memory, where the word lines connected to a single switch are uniformly intersected along the terminated digit line, shows that a voltage step on the selected switch couples via the 32 word lines controlled by the switch all along the digit line simultaneously. Thus, a step at the switch generates a common-mode noise, the amplitude of which can be predicted from the inter-line and line-to-ground plane capacitances. The portion of this common-mode signal which the stack converts into the differential mode depends on the balance between the two digit lines of a pair and varies from one line-pair to another.

A pulse transformer for each word line was used for capacitive decoupling between the word selection matrix and the memory stack. The interwinding capacitance of the transformer is a maximum of 7 picofarads, whereas the capacitance between a word line and all the digit lines connected together is about 60 pico-

farads, resulting in a switching noise attenuation of about 10 to 1. Starting with a 35-volt, 30-nsec rise time step for switch selection, the half-selected word lines experience only a 3-volt step because of the isolation afforded by the transformer. This condition in turn causes a 0.5-volt common-mode spike to exist on the digit line. In the worst case this spike generates a differential noise signal almost as large in amplitude as a sense signal. This noise must be displaced in time from the sense signal by causing the timing pulse for the switch to start earlier than the timing pulse for the read driver.

As shown in Figure 12, the switch timing pulse is used to select a switch. This technique differs from normal practice, which does without a timing pulse, with the result that at least one switch is turned on all the time. In the present memory, a switch is turned on for a specific length of time to let the read and the write currents go through; otherwise, no switch stays turned on. The switch noise is appreciably reduced by holding the switches off until after the memory address register has completely settled from the address transfer transient, as otherwise a spurious selection of switches during the address transfer transient will inject additional noise into the stack. By turning off the switch as soon as the write pulse is terminated, the problem of slow switch turn-off can be easily eliminated.

Another closely associated problem is injection of noise via the half-selected word lines

controlled by the same switch during the read pulse. This is due to the passage of the read pulse through the finite impedance of the switch circuit, with its mechanism of noise injection being very similar to the one described above. This type of noise is a threat due to the fact that it always coincides with the sense signal. It is obvious that this problem can be minimized by lowering the impedance of the switch circuit. A low switch impedance is also desirable from the standpoint of matrix operation because it permits unimpeded flow of the read and the write pulses. The problem was solved by avoiding cables for connecting the switch channels and the word selection matrix altogether, and instead packaging the output stages of the switch channels at the memory stack. Here again the isolation provided by the use of coupling transformers alleviates the noise problem greatly.

Another advantage in using coupling transformers is the speed increase of the switch operation due to the capacitive isolation afforded by the transformers. Actually, this speed increase and the switch turn-on noise reduction brought about by the coupling transformers are closely related. The capacitive charging and discharging currents that must come from a switch when it is turned on and off are made small by the use of the transformers. Thus, the switch speed is increased. Since the switch noise is caused by the same charging current entering the memory stack, the noise is reduced by the transformers.

As shown in Figures 6 and 12, the read and the write pulses have two different timings, depending on the word address. This feature is necessary because the digit line delay of 20 nanoseconds from the driving and sensing point to the termination is not negligible compared with the drive current widths. The problems here are basically that of aligning the read and the strobe pulses and that of aligning the write and the digit pulses. In the present memory, the strobe and the digit pulses are fixed and the read and the write pulses are shifted according to the word address. The 1024 words of the memory are divided into two groups of 512 words. One group is closer to the digit-driving and sensing points while the other group is closer to the terminations as shown in Figure

6. The former group uses Word Pulse Timing A, and the latter group Word Pulse Timing B. Figure 12 shows that the read pulse of Timing A is delayed compared to that of Timing B, and the write pulse of Timing A is advanced compared to that of Timing B. The difference is 10 nanoseconds, which is one-half of the effective digit line delay.

## DIGIT SYSTEM

The digit system (Figure 16) is composed of the circuits that are used to detect and to write or regenerate information in each of the one hundred bits of a selected word. The circuits include 100 sense amplifiers, 100 digit drivers and the 100 flip-flops that form the memory information register.

### *Digit Driver*

During the write time, the digit driver provides a 70-milliampere current pulse into one of the two digit lines in a direction to add to the write pulse in one of the two cores of a memory bit. The digit driver consists of two identical current drivers which are under the dual control of the timing generator and the flip-flop in the memory information register. The width of the digit pulse, 75 nanoseconds, is controlled by the digit timing pulse.

The first stage of the digit driver produces a gated 10-volt pulse. The pulse is produced in one of the current drivers by the coincidence of the positive digit timing pulse and a low voltage level from one side of the flip-flop in the memory information register. The second current driver is inhibited by the positive level from the second side of the flip-flop.

The gated pulse is applied to the second stage through a capacitor that is used to give the pulse a negative level shift so that at the input to the second stage the pulse goes positive to -25 volts from a reference level of -35 volts. The second stage is a double emitter follower which is used to provide a voltage drive for the output stage.

The output stage is a nonsaturating current driver whose output current is determined by the resistance in the emitter circuit and the voltage swing at the base of the transistor. The output stage drives the center of the digit line

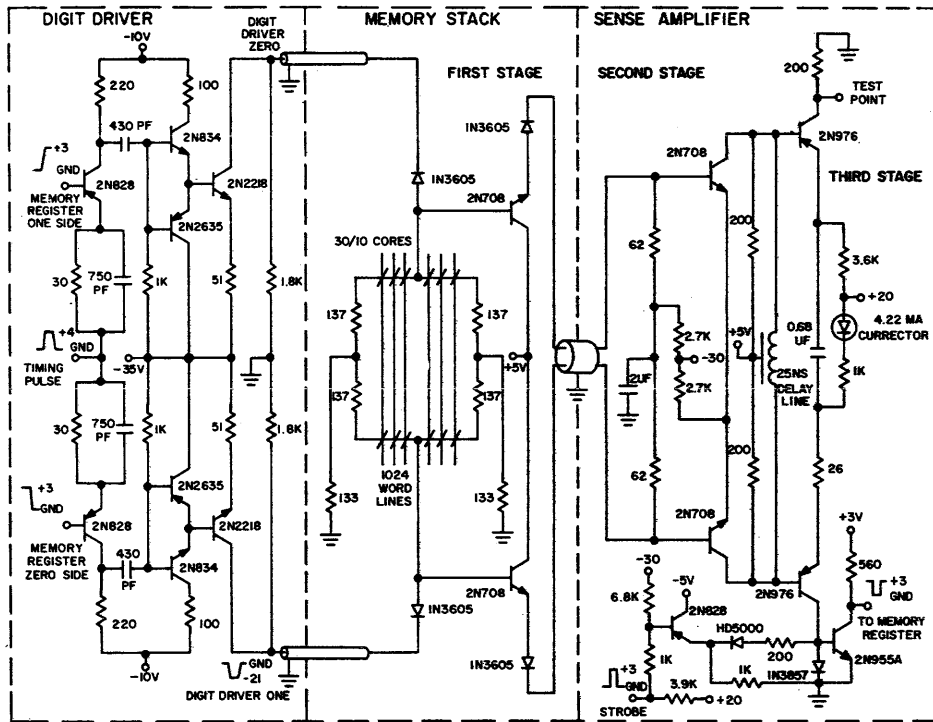


Figure 16. Digit system.

through a 100-ohm cable and a series diode, providing a 70-milliampere pulse into each half of the digit line. The diode is used at the memory stack to isolate the digit driver cable from the digit line when the driver is not in use so that low-level signals in the memory are not loaded by the cable.

*Sense Amplifier*<sup>13, 14, 15, 16</sup>

The digit lines are terminated at both ends in order to reduce the recovery time of the memory stack. As a result, only half of the difference signal from the two cores of a bit is available at the sense amplifier input. The difference signal is bipolar where one polarity represents a ONE and the other polarity represents a ZERO. The sense amplifier amplifies the difference signal and is strobed during a portion of the read time. The polarity of the sense signal at strobe time is sensed and if a ONE is detected, the sense amplifier produces a 3-volt negative-going output pulse which sets a flip-flop in the memory information register. If a ZERO is sensed, no change occurs at the sense amplifier output. During the write time, a negative digit pulse of approximately 20 volts

is applied to one of the two digit lines, depending on whether a ONE or a ZERO is being written into the memory. The sense amplifier is inhibited during the write time by the strobe circuit and recovers in less than 50 nanoseconds after the last difference-mode reflections from the digit pulse have ceased to exist on the digit lines.

The first stage of the sense amplifier consists of two emitter followers which are connected to the center of the digit lines as shown in Figure 16 and are used to provide a high input impedance so that the sense amplifier does not load the digit lines and does not interfere with the termination of the lines. The emitter followers and series diodes are physically mounted near the center of the memory stack and are connected to the plug-in board that contains the regeneration loop circuits by means of a 125-ohm shielded twisted-pair cable.

The twisted-pair cable is terminated at the input to the second stage with resistors connected to a decoupled power supply. When the negative digit pulse is applied to one of the digit lines, the corresponding emitter follower

is turned off and the transient at the input to the second stage is limited to 300 millivolts, since the currents in the cable termination are reduced to zero. The diodes are used in series with the emitter-follower outputs in order to prevent the flow of current if the base-to-emitter breakdown voltage is exceeded by the digit pulse.

The second stage of the sense amplifier is a differential amplifier, with the transistor collectors connected together through a delay line. This stage amplifies the difference between the input signals and sums the inverted amplified difference signal and the delayed amplified difference signal. This produces output voltage waveforms at the collectors that do not have a dc level shift with repetition rate variations. The output waveforms are on a well-determined reference voltage level which is determined by the current in the large resistance in the emitter circuit. Practically all of the emitter current reaches the transistor collectors and produces a constant operating voltage across the parallel combination of the collector resistors. The operating voltage is constant even if the current is not shared equally by the two transistors, since the delay line provides a dc short circuit between the collectors. The collector resistors are used to terminate the delay line so that there will be no reflections and the outputs of the second stage will recover to the reference level in a minimum of time after the end of the digit transient.

The delay of the delay line is long enough that a usable amount of the inverted amplified sense signal is passed before the output is reduced by the delayed amplified sense signal. The delay in this system is 25 nanoseconds, which is approximately one-half the base width of a sense signal.

The third stage of the sense amplifier is an ac-coupled differential amplifier. One output is used as a test point for observing amplified sense signals, and the other output drives the next stage. The third stage has a maximum output current swing that is limited by the current in the emitter current sources. This prevents the digit transient from overpowering the inhibit current in the strobe circuit.

The last stage is the strobe and pulse-stretching circuit. This stage contains a bistably biased five-milliampere germanium tunnel diode which drives an output transistor. The tunnel diode has two inputs. One input is from the third stage which provides the amplified sense signal and also provides the normal bias current for the tunnel diode. The second input is from the strobe circuit which during the inhibit time provides sufficient reverse current through the tunnel diode to keep it in the low-voltage state during the digit transient.

The operation of this stage is illustrated in Figure 17, which shows the tunnel diode volt-ampere characteristic and its load line. The tunnel diode is normally biased in the low-voltage state at point A and is unable to switch to the high-voltage state during the digit transient because of the current-limiting action of the third stage. During a portion of the read time the sense amplifier is strobed by removing the inhibit current, thereby biasing the tunnel diode in the low-voltage state near the knee at point B. A difference signal of five millivolts at the input of the sense amplifier and the polarity of a ONE signal is sufficient to trigger the tunnel diode to point C in the high-voltage state. The tunnel diode turns on the output transistor which produces a three-volt negative-going pulse used to set a flip-flop in the memory information register. The tunnel diode remains in the high-voltage state until the inhibit

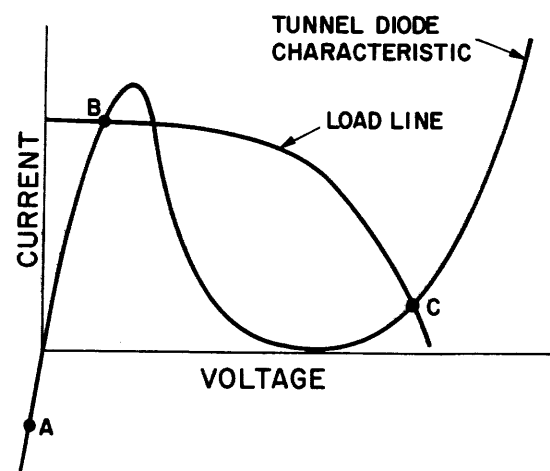


Figure 17. Tunnel diode characteristic and load line.

current is applied by the strobe circuit. The inhibit current resets the tunnel diode to point A and terminates the output pulse.

The operation of the sense amplifier is illustrated by the waveforms in Figure 18. Figure 18(a) shows superimposed the read-out signals and digit transients on the two lines of a digit line-pair. The stored information is represented in the difference between the two signals that appear on the lines at read time.

Figure 18(b) shows the signals that appear at the sense-amplifier test point when the delay line is removed from the circuit. The solid line shows reading and regenerating a ONE and the dotted line shows reading and regenerating a ZERO. The amplifier has amplified the difference in the read-out signals and has limited the digit transient. It can be observed that these waveforms have a dc component which would result in a dc shift in an ac amplifier. This shift would be particularly objectionable at high repetition rates. In addition, the waveform base line is dependent on the dc balances of the previous stages.

Figure 18(c) shows the test point signals with the delay line in the circuit. The waveforms represent the sum of the inverted amplified difference signal and the delayed amplified difference signal. These waveforms have no dc component other than the base-line voltage, which is well determined.

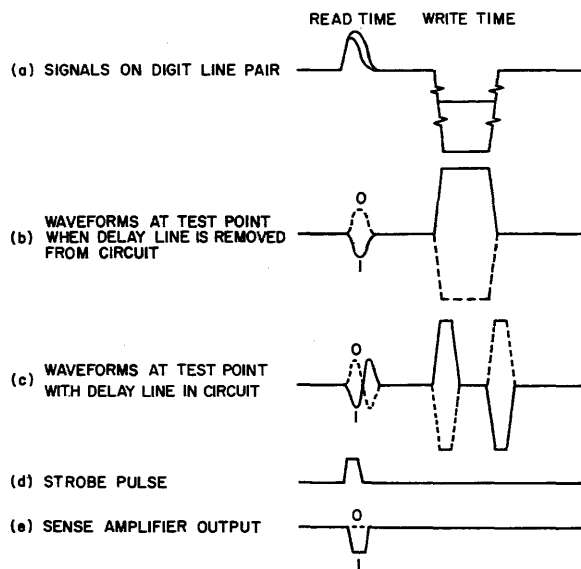


Figure 18. Sense amplifier operation.

Figure 18(d) shows the strobe pulse which is positive only during the first peak of the amplified sense signal shown in Figure 18(c).

Figure 18(e) shows the sense amplifier output. The negative-going pulse indicates the detection of a ONE.

PACKAGING

The circuitry of the memory, except for those parts that had to be near the memory stack for special reasons, is packaged in four nests surrounding the memory stack as seen in the photograph in Figure 19. Each nest has 10 removable motherboards, each of which could potentially contain up to 56 small plug-in modules. Individual modules contain such parts as logic blocks, portions of drivers, portions of the sense amplifiers, etc. When a nest is completely assembled, all of the circuitry within it is interconnected by 70-ohm-impedance printed strip lines on both sides of the motherboards and perpendicular grandmother boards. Interconnections between nests are made by coaxial cables. Some of the memory circuitry which did not lend itself to modular packaging because of power dissipation or size considerations, such as driver output stages, was packaged on specially built motherboards by removing some or all of the provisions for pluggable modules. All logic level interconnections are made via 70-ohm cables. Read and write driver outputs are transmitted to the bipolar diode matrix at the stack via 70-ohm cables. To ob-

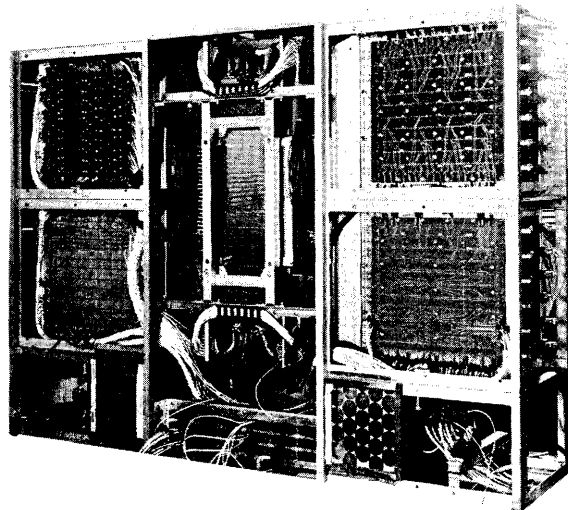


Figure 19. Memory system.

tain a lower impedance, the output stages of the switch channels are located at the memory stack. These stages are connected to the rest of the switch channels via 50-ohm cables. The digit driver outputs are transmitted to the stack via 100-ohm cables, and the sense amplifier first stages are connected to the rest of the sense amplifier by twisted-pair balanced cables having a common ground sheath and a differential impedance of 125 ohms.

## TEST RESULTS

The 1024-word two-core-per-bit memory was built with a complete word system and a full digit system of 100 bits. Also, a special memory exerciser was built to thoroughly test the memory system.

Figure 20 shows a switch voltage, read and write currents and a digit current. The switch waveform was observed at the center tap of a word line transformer primary winding (see Figure 13). The undulations on the plateau were caused by the flow of read and write currents through the switch circuit. These undulations would have been much larger if the switches had not been mounted on the memory stack. Read and write pulses of both Timing A and Timing B are shown in the figure. Note that the read and the write pulses of Timing

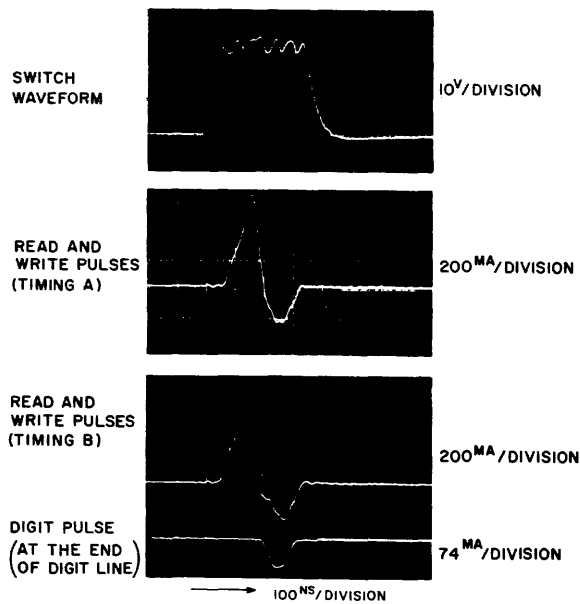


Figure 20. Switch voltage, read and write pulses ("Timing A" and "Timing B"), and digit pulse.

A are close together, while those of Timing B are slightly separated. The digit pulse was observed at the end of a digit line.

As shown in Figure 21(a), T termination networks are used to terminate digit lines. The termination impedances are  $Z_d = 137$  ohms and  $Z_s = 133$  ohms. For practicality, the same termination networks are used for all the 100 digit line-pairs, although there is some indication that the optimum value changes from bit to bit, not so much for  $Z_d$  but to some extent for  $Z_s$ . The calculated value of the common-mode termination is  $Z_c = 403$  ohms. The word lines crossing digit lines are responsible for the large difference between the differential mode and the common-mode termination impedances.

Figure 21(b) shows the voltage waveforms at the three points on a termination network as indicated in Figure 21(a). It is seen that the voltage waveform at the end of the undriven line B and that at the junction C in the T termination are the same. This is important because it means that no current flows out of the undriven digit line, which is a basic requirement for memory operation as shown in Figure 2. To make the net current propagation on the undriven line zero, there must be a voltage propagation on it. (Here, the same velocity is assumed for all the propagation modes that exist in the memory operation.) It is to be

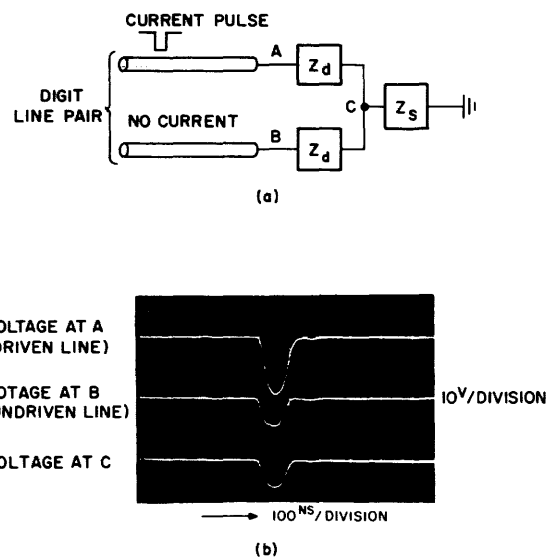


Figure 21. Voltage waveforms at a termination network: (a) T termination network ( $Z_d = 137$  ohms and  $Z_s = 133$  ohms), (b) voltage waveforms.

noted that the condition shown in Figure 21 is realized only when all the 100 digit drivers are operating.

Figure 22 shows waveforms at a sense amplifier test point, together with related waveforms. Figure 22(a) shows two bits, one at the edge of the memory stack and the other at the center, regenerating ONES and ZEROS alternately over the entire memory of 1024 words. Note that the sense signals are delayed to avoid the switch noise. The switch noise, although comparable in amplitude to the sense signal, could have been made as small as it is only by the use of coupling transformers. The negative-going sense signal represents a ONE and the positive-going sense signal a ZERO. The digit transient takes about 350 nanoseconds to die down, measured from the start of the digit pulse. This time includes approximately 300 nanoseconds attributed to the base width of the digit pulse and the stack recovery time, plus 50 nanoseconds attributed to the sense amplifier. This relatively slow recovery of the stack, even with the elaborate T

termination, seems due to the imperfection of digit lines as transmission lines.

Figure 22(b) shows regeneration of ONES on all the 1024 words. It is seen that the information is available at the memory register in about 230 nanoseconds from the beginning of the read command pulse. Figure 22(c) shows a higher repetition rate operation of about 450-nanosecond cycle time. It shows regeneration of ONES and ZEROS on alternate words over the entire memory. Here, the switch noise and the digit transient recovery are made concurrent without affecting the sense signals.

The waveforms shown above represent only a small portion of the tests performed on the memory with the aid of the memory exerciser. These tests confirmed the soundness of the design philosophy, the effectiveness of the problem solving approach, and the practicality and reliability of the memory system actually built.

CONCLUSION

Development of the present memory system evolved the method of control of wave propagation. Unless wave propagation is controlled, it is almost impossible to operate a high-speed memory. The basic requirements for control are:

1. Use of two neighboring digit lines as a pair for one-bit location
2. Equalization of coupling between the digit lines
3. Use of differential sense amplifiers
4. Termination of digit lines on both ends for all the existing wave propagations with particular emphasis on the differential-mode termination.

The last requirement is met by the present memory due to the particular digit drive scheme used. It requires careful study to choose a digit drive scheme, as otherwise, a simultaneous termination for all the possible propagations becomes a very complex problem, with no practical answer. Although not applicable to the present memory, it is preferred that only the differential-mode propagations exist. This may be accomplished by the proper selection of memory cell types and digit drive schemes, and will simplify the propagation problem

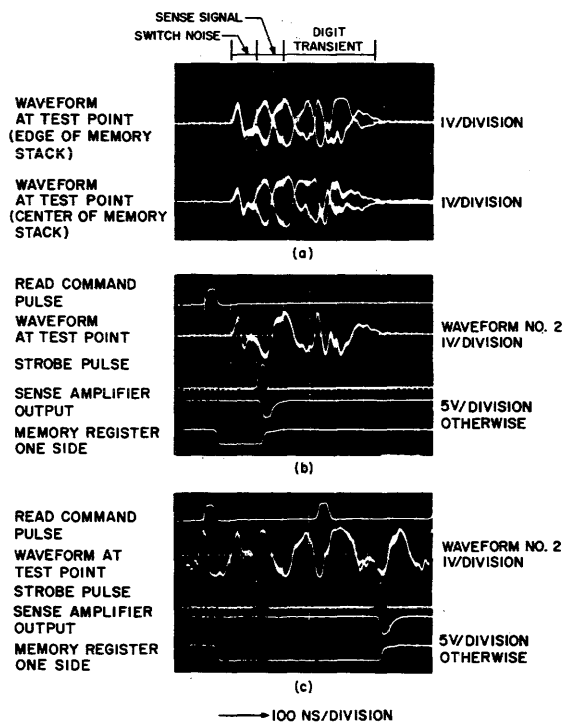


Figure 22. Waveforms at sense amplifier test points: (a) regeneration of ones and zeros, (b) regeneration of ones, (c) regeneration of ones and zeros at 450-nsec cycle time.



greatly. It should be emphasized that the approach used in this paper in treating wave propagations in a memory stack is applicable to any word-organized memory.

The use of transformers to couple read and write pulses to individual word lines proved very successful in alleviating the noise problem associated with the word selection matrix. There is still a possibility of reducing the noise further by reducing the transformer interwinding capacity, which will increase the system reliability and at the same time enable a faster access time.

The use of a delay line in a differential sense amplifier minimized the problems of dc imbalances and level shift when sensing small signals in an environment of large digit pulses. In addition, the use of a tunnel diode strobe circuit provided low-level thresholding and high-speed operation.

There seems to be no basic difficulty in building a memory with twice as many words, using the basic design described here. It is expected, however, that the cycle time will be slightly longer since the digit lines are twice as long.

## APPENDIX

### WAVE PROPAGATION ON MULTIPLE PARALLEL LINES WITH EQUALIZED COUPLING

The analysis given below shows the modes of wave propagation that can exist on a set of multiple parallel lines with equalized coupling. The following assumptions are made:

1. Propagation is in the direction of the digit lines only. This implies that the word lines are considered as contributing only to the coupling among digit lines.
2. Digit lines are distributed constant lines. This is justified because for the frequencies of interest, it is not necessary to consider the line irregularities caused by memory cells.
3. Digit lines are uniform and have no discontinuities. This assumption may not hold precisely in practice, but is made to permit a mathematical analysis.

4. A ground plane is present. This assumption is also made to permit a mathematical analysis.

A similar problem of multiple line wave propagation was studied a long time ago.<sup>17</sup> The solution given here is more general than the one given in the reference and more readily applicable to memories.

Let the number of lines be  $2n$ , where  $n$  is an arbitrary integer. From these, we form  $n$  pairs. Pair  $i$  consists of line  $i$  and line  $-i$ , where  $i = 1, 2, \dots, n$ . This is shown in Figure A-1. Pair-to-pair coupling is equalized, which implies that, when we consider pair  $i$  and pair  $j$  ( $i \neq j$ , and  $i, j = 1, 2, \dots, n$ ), the coupling is the same between line  $i$  and line  $j$ , line  $i$  and line  $-j$ , line  $-i$  and line  $j$ , and line  $-i$  and line  $-j$ . To be general, the coupling is made a function of  $i$  and  $j$ . The case in which the coupling is constant regardless of  $i$  and  $j$  has been treated elsewhere.<sup>17</sup>

Using matrix notation, the pertinent differential equations are

$$\left[ -\frac{dV}{dx} \right] = [Z] [I] \quad (\text{A-1})$$

$$\left[ -\frac{dI}{dx} \right] = [Y] [V] \quad (\text{A-2})$$

The above factors are defined as follows:

$$[V] = \begin{bmatrix} V_1 \\ V_{-1} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ V_n \\ V_{-n} \end{bmatrix} \quad [I] = \begin{bmatrix} I_1 \\ I_{-1} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ I_n \\ I_{-n} \end{bmatrix}$$

where

- $V_i$  = Voltage on line  $i$
- $V_{-i}$  = Voltage on line  $-i$
- $I_i$  = Current on line  $i$
- $I_{-i}$  = Current on line  $-i$
- $i = 1, 2, 3, \dots, n$ .

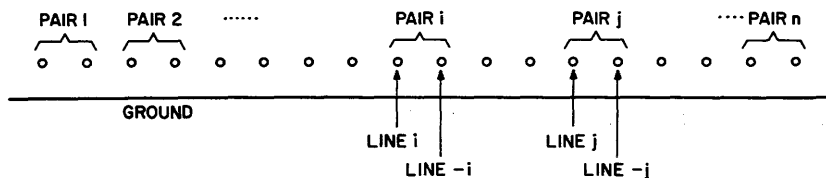


Figure A-1. Cross section of a system of 2n parallel lines.

$$[Z] = \begin{bmatrix} Z_{s1} & Z_{m1} & Z_{12} & Z_{12} & \dots & Z_{1n} & Z_{1n} \\ Z_{m1} & Z_{s1} & Z_{12} & Z_{12} & \dots & Z_{1n} & Z_{1n} \\ Z_{21} & Z_{21} & Z_{s2} & Z_{m2} & \dots & Z_{2n} & Z_{2n} \\ Z_{21} & Z_{21} & Z_{m2} & Z_{s2} & \dots & Z_{2n} & Z_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ Z_{n1} & Z_{n1} & Z_{n2} & Z_{n2} & \dots & Z_{sn} & Z_{mn} \\ Z_{n1} & Z_{n1} & Z_{n2} & Z_{n2} & \dots & Z_{mn} & Z_{sn} \end{bmatrix}$$

where

$Z_{si}$  = Self series impedance per unit length of line i or line -i,  $i = 1, 2, 3, \dots, n$ .

$Z_{mi}$  = Mutual series impedance per unit length between line i and line -i,  $i = 1, 2, 3, \dots, n$ .

$Z_{ij}$  = Mutual series impedance per unit length between either line i or line -i and either line j or line -j,  $i \neq j$  and  $i, j = 1, 2, 3, \dots, n$ . (equalized)

$Z_{ij} = Z_{ji}$   $i \neq j$  and  $i, j = 1, 2, 3, \dots, n$ .

$$[Y] = \begin{bmatrix} Y_{s1} & Y_{m1} & Y_{12} & Y_{12} & \dots & Y_{1n} & Y_{1n} \\ Y_{m1} & Y_{s1} & Y_{12} & Y_{12} & \dots & Y_{1n} & Y_{1n} \\ Y_{21} & Y_{21} & Y_{s2} & Y_{m2} & \dots & Y_{2n} & Y_{2n} \\ Y_{21} & Y_{21} & Y_{m2} & Y_{s2} & \dots & Y_{2n} & Y_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ Y_{n1} & Y_{n1} & Y_{n2} & Y_{n2} & \dots & Y_{sn} & Y_{mn} \\ Y_{n1} & Y_{n1} & Y_{n2} & Y_{n2} & \dots & Y_{mn} & Y_{sn} \end{bmatrix}$$

where

$Y_{si}$  = Self parallel admittance per unit length of line i or line -i,  $i = 1, 2, 3, \dots, n$ .

$Y_{mi}$  = Mutual parallel admittance per unit length between line i and line -i,  $i = 1, 2, 3, \dots, n$ .

$Y_{ij}$  = Mutual parallel admittance per unit length between either line i or line -i and either line j or line -j,  $i \neq j$  and  $i, j = 1, 2, 3, \dots, n$ .

$Y_{ij} = Y_{ji}$   $i \neq j$  and  $i, j = 1, 2, 3, \dots, n$ .

From Equations (A-1) and (A-2),

$$\left[ \frac{d^2V}{dx^2} \right] = [Z] [Y] [V] = [\mu] [V] \quad (A-3)$$

where

$$[\mu] = [Z] [Y]$$

$$= \begin{bmatrix} \mu_{s1} & \mu_{m1} & \mu_{12} & \mu_{12} & \dots & \mu_{1n} & \mu_{1n} \\ \mu_{m1} & \mu_{s1} & \mu_{12} & \mu_{12} & \dots & \mu_{1n} & \mu_{1n} \\ \mu_{21} & \mu_{21} & \mu_{s2} & \mu_{m2} & \dots & \mu_{2n} & \mu_{2n} \\ \mu_{21} & \mu_{21} & \mu_{m2} & \mu_{s2} & \dots & \mu_{2n} & \mu_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \mu_{n1} & \mu_{n1} & \mu_{n2} & \mu_{n2} & \dots & \mu_{sn} & \mu_{mn} \\ \mu_{n1} & \mu_{n1} & \mu_{n2} & \mu_{n2} & \dots & \mu_{mn} & \mu_{sn} \end{bmatrix}$$

and

$$\mu_{si} = Z_{si} Y_{si} + Z_{mi} Y_{mi} + 2 \sum_{k \neq i} Z_{ik} Y_{ki}; \quad i = 1, 2, 3, \dots, n$$

$$\mu_{mi} = Z_{si} Y_{mi} + Z_{mi} Y_{si} + 2 \sum_{k \neq i} Z_{ik} Y_{ki}; \quad i = 1, 2, 3, \dots, n$$

$$\mu_{ij} = Z_{ij} (Y_{sj} + Y_{mj}) + Y_{ij} (Z_{si} + Z_{mi}) + 2 \sum_{\substack{k \neq i \\ k \neq j}} Z_{ik} Y_{kj}; \quad i, j \neq j.$$

and  $i, j = 1, 2, 3, \dots, n$ .

In general

$$\mu_{ij} \neq \mu_{ji}; i \neq j, \text{ and } i, j = 1, 2, 3, \dots, n.$$

Assume a solution for Equation (A-3) of the form

$$V_i = V_{oi} e^{\gamma_i x}$$

$$i = 1, 2, \dots, n.$$

In order that the solution not to be trivial (i.e.,  $V_{oi} = 0, i = 1, 2, \dots, n$ ), the following must hold:

$$\begin{vmatrix} \mu_{s1} - \gamma^2 & \mu_{m1} & \mu_{12} & \mu_{12} & \dots & \mu_{1n} & \mu_{1n} \\ \mu_{m1} & \mu_{s1} - \gamma^2 & \mu_{12} & \mu_{12} & \dots & \mu_{1n} & \mu_{1n} \\ \mu_{21} & \mu_{21} & \mu_{m2} & \mu_{m2} & \dots & \mu_{2n} & \mu_{2n} \\ \mu_{21} & \mu_{21} & \mu_{s2} - \gamma^2 & \mu_{s2} - \gamma^2 & \dots & \mu_{2n} & \mu_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \mu_{n1} & \mu_{n1} & \mu_{n2} & \mu_{n2} & \dots & \mu_{sn} - \gamma^2 & \mu_{mn} \\ \mu_{n1} & \mu_{n1} & \mu_{n2} & \mu_{n2} & \dots & \mu_{mn} & \mu_{sn} - \gamma^2 \end{vmatrix} = 0$$

or

$$\begin{aligned} & (\mu_{s1} - \mu_{m1} - \gamma^2) (\mu_{s2} - \mu_{m2} - \gamma^2) \dots (\mu_{sn} - \mu_{mn} - \gamma^2) \\ \times & \begin{vmatrix} (\mu_{s1} - \mu_{m1} - \gamma^2) & 2\mu_{12} & \dots & 2\mu_{1n} \\ 2\mu_{21} & \mu_{s2} + \mu_{m2} - \gamma^2 & \dots & 2\mu_{2n} \\ \dots & \dots & \dots & \dots \\ 2\mu_{n1} & 2\mu_{n2} & \dots & \mu_{sn} + \mu_{mn} - \gamma^2 \end{vmatrix} = 0 \end{aligned}$$

This is a  $2n^{\text{th}}$  degree equation in  $\gamma^2$ . Let the roots be

$$\gamma_k^2 = \begin{cases} \mu_{sk} - \mu_{mk} & k = 1, 2, \dots, n \\ \text{Root of the determinant} & k = n + 1, n + 2, \dots, 2n \end{cases}$$

Consider only the forward propagation, because the backward propagation is the same except for direction. Then

$$\gamma_k = \begin{cases} -\sqrt{\mu_{sk} - \mu_{mk}} & k = 1, 2, \dots, n \\ -\sqrt{\text{Root of the determinant}} & k = n + 1, n + 2, \dots, 2n \end{cases}$$

Now the solution of Equation (A-3) is

$$\left. \begin{aligned} V_i &= \sum_k V_{ik} e^{\gamma_k x} \\ V_{-i} &= \sum_k V_{-ik} e^{\gamma_k x} \end{aligned} \right\} \begin{aligned} &i = 1, 2, \dots, n \text{ and} \\ &k = 1, 2, \dots, 2n \end{aligned} \tag{A-4}^*$$

Substituting Equation (A-4) into Equation (A-3) to find relationships among  $V_{ik}$  and  $V_{-ik}$ ,

$$\sum_{j \neq i} \mu_{ij} (V_{jk} + V_{-jk}) + (\mu_{si} - \gamma_k^2) V_{ik} + \mu_{mi} V_{-ik} = 0 \tag{A-5}$$

$$\sum_{j \neq i} \mu_{ij} (V_{jk} + V_{-jk}) + \mu_{mi} V_{ik} + (\mu_{si} - \gamma_k^2) V_{-ik} = 0 \tag{A-6}$$

\* Here it is assumed that  $\lambda_k$ 's are all single roots. Inclusion of multiple roots, however, does not change the form of Equations (A-11) and (A-12), because the terms of the form  $x^p e^{\gamma_k x}$ , where  $p$  is a non-zero integer, cannot appear in the solution.

where

$$i = 1, 2, \dots, n \text{ and } k = 1, 2, \dots, 2n.$$

By subtracting Equation (A-6) from (A-5),

$$(\mu_{si} - \mu_{mi} - \gamma_k^2) (V_{ik} - V_{-ik}) = 0$$

If  $k \neq i$ ,  $\mu_{si} - \mu_{mi} - \gamma_k^2 \neq 0$ . Therefore

$$V_{ik} = V_{-ik}, \text{ where } k \neq i, \text{ and } k = 1, 2, \dots, 2n \tag{A-7}$$

Then, Equation (A-5) can be rewritten as

$$\sum_{\substack{j \neq i \\ j \neq k}} 2\mu_{ij} V_{jk} + (\mu_{si} + \mu_{mi} - \gamma_k^2) V_{ik} + \mu_{ik} (V_{kk} + V_{-kk}) = 0 \tag{A-8}$$

where

$$k \neq i, \text{ and } i, k = 1, 2, \dots, n.$$

If  $k = i$ ,  $\mu_{sk} - \gamma_k^2 = \mu_{mk}$  ( $k = 1, 2, \dots, n$ ).

Then, from Equation (A-5)

$$\sum_{j \neq k} 2\mu_{kj} V_{jk} + \mu_{mk} (V_{kk} + V_{-kk}) = 0 \tag{A-9}$$

where

$$k = 1, 2, \dots, n.$$

For a given  $k$ , Equations (A-8) and (A-9) together form  $n$  simultaneous equations in  $V_{jk}$  ( $j \neq k$ , and  $j = 1, 2, \dots, n$ ) and  $(V_{kk} + V_{-kk})$ . In other words, if  $k$  is fixed, Equation (A-9) gives one equation and Equation (A-8) gives

$(n - 1)$  equations, because  $i$  can take  $(n - 1)$  different values. Therefore,

$$V_{jk} = 0 \quad j \neq k, \text{ and } j, k = 1, 2, \dots, n$$

and

$$V_{kk} + V_{-kk} = 0 \quad k = 1, 2, \dots, n$$

These are combined with Equation (A-7) to obtain

$$\left. \begin{aligned} V_{ik} &= -V_{-ik} & i = k = 1, 2, \dots, n \\ V_{ik} &= V_{-ik} = 0 & i \neq k \text{ and } i, k = 1, 2, \dots, n \\ V_{ik} &= V_{-ik} & i = 1, 2, \dots, n \text{ and } k = n + 1, n + 2, \dots, 2n \end{aligned} \right\} \tag{A-10}$$

Now Equation (A-4) becomes

$$\left. \begin{aligned} V_i &= V_{ii} e^{\gamma_i x} + \sum_{k=n+1}^{2n} V_{ik} e^{\gamma_k x} \\ V_{-i} &= -V_{ii} e^{\gamma_i x} + \sum_{k=n+1}^{2n} V_{ik} e^{\gamma_k x} \end{aligned} \right\} i = 1, 2, \dots, n \tag{A-11}$$

Using Equation (A-1), current equations are obtained:

$$\left. \begin{aligned} I_i &= \frac{V_{ii}}{Z_{oi}} e^{\gamma_i x} + \sum_{k=n+1}^{2n} I_{ik} e^{\gamma_k x} \\ I_{-i} &= -\frac{V_{ii}}{Z_{oi}} e^{\gamma_i x} + \sum_{k=n+1}^{2n} I_{ik} e^{\gamma_k x} \end{aligned} \right\} i = 1, 2, \dots, n \tag{A-12}$$

where

$$\gamma_i = -\sqrt{\mu_{si} - \mu_{mi}} = -\sqrt{(Z_{si} - Z_{mi})(\bar{Y}_{si} - \bar{Y}_{mi})}$$

$$Z_{oi} = \sqrt{\frac{Z_{si} - Z_{mi}}{Y_{si} - Y_{mi}}}$$

$$i = 1, 2, \dots, n$$

Equations (A-11) and (A-12) show that the possible modes of propagation are:

- 1) Independent differential mode for each line-pair
- 2) Common modes in which the two lines of a pair have identical wave propagation.

#### ACKNOWLEDGMENT

The authors wish to acknowledge the assistance given on this project by the members of the Computer Advanced Product Research Group. Credit is due to them for designing the two-level logic modules, developing the new packaging techniques, and designing and constructing the memory exerciser. Special credit is due to H. C. Nichols for his invaluable contribution in the fabrication of the memory system. Additional acknowledgment is due to Memory Products Department, RCA Electronic Components and Devices, for the construction of the memory stack.

#### REFERENCES

1. V. L. NEWHOUSE, "The Utilization of Domain Wall Viscosity in Data Handling Devices," *Proc. IRE*, vol. 45, no. 11, pp. 1484-1492, November 1957.
2. W. S. KOSONOCKY, "Memory System," U.S. Patent 3,042,905, filed December 11, 1956, issued July 3, 1962.
3. J. A. RAJCHMAN, "Ferrite Aperture Plate for Random Access Memory," *Proc. IRE*, vol. 45, no. 3, pp. 325-334, March 1957.
4. M. M. KAUFMAN and V. L. NEWHOUSE, "Operating Range of a Memory Using Two Ferrite Plate Apertures per Bit," *Journal of Applied Physics*, vol. 29, no. 3, pp. 487-488, March 1958.
5. R. E. MCMAHON, "Impulse Switching of Ferrites," *Solid State Circuit Conference Digest*, pp. 16-17, February 1959.
6. R. H. TANCRELL and R. E. MCMAHON, "Studies in Partial Switching of Ferrite Cores," *Journal of Applied Physics*, vol. 31, no. 5, pp. 762-771, May 1960.
7. R. H. JAMES, W. M. OVERN, and C. W. LUNDBERG, "Flux Distribution in Ferrite Cores under Various Modes of Partial Switching," *Journal of Applied Physics Supplement to vol. 32*, no. 3, pp. 385-395, March 1961.
8. C. J. QUARTLY, "A High Speed Ferrite Storage System," *Electronic Engineering*, vol. 31, no. 12, pp. 756-758, December 1959.
9. H. AMEMIYA, H. P. LEMAIRE, R. L. PRYOR, and T. R. MAYHEW, "High-Speed Ferrite Memories," *AFIP Conference Proceedings*, vol. 22, pp. 184-196, Fall 1962.
10. W. H. RHODES, L. A. RUSSEL, F. E. SAKALAY, and R. M. WHALEN, "A 0.7-Microsecond Ferrite Core Memory," *IBM Journal*, vol. 5, no. 3, pp. 174-182, July 1961.
11. G. F. BLAND, "Directional Coupling and Its Use for Memory Noise Reduction," *IBM Journal of Research and Development*, vol. 7, no. 3, pp. 252-256, July 1963.
12. W. T. WEEKS, "Computer Simulation of the Electrical Properties of Memory Arrays," *IEEE Transactions on Electronic Computers*, vol. EC-12, no. 5, pp. 874-887, December 1963.
13. G. H. GOLDSTICK and E. F. KLEIN, "Design of Memory Sense Amplifiers," *IRE Transactions on Electronic Computers*, vol. EC-11, pp. 236-253, April 1962.

14. T. R. MAYHEW, "The Design of a Sense Amplifier for a Thin Film Memory," Master's Thesis, University of Pennsylvania, June 1962.
15. R. T. LURVEY and D. F. JOSEPH, "RCA N7100 Microferrite Array," Application Note SMA-9, RCA Semiconductor and Materials Division, Somerville, N.J., August 1962.
16. B. A. KAUFMAN and J. S. HAMMOND, III, "A High Speed Direct-Coupled Magnetic Memory Sense Amplifier Employing Tunnel-Diode Discriminators," IEEE Transactions on Electronic Computers, vol. EC-12, pp. 282-295, June 1963.
17. J. R. CARSON and RAY S. HOYT, "Propagation of Periodic Currents over a System of Parallel Wires," Bell System Tech. Journal, vol. 6, no. 3, pp. 495-545, July 1927.



# AN ASSOCIATIVE PROCESSOR

*Richard G. Ewing and Paul M. Davies  
Abacus Incorporated, Santa Monica, California*

## 1. INTRODUCTION

This paper describes the computer system designed under an Air Force sponsored study program to develop a non-cryogenic Associative Processor organization and to study its possible use in a variety of Aerospace applications. Two approaches were considered to this problem: one in which an associative memory would be added to a more or less conventional computer and another in which a new organization would be developed around the principle of memory distributed logic. The latter approach was chosen because it appears to result in a more efficient form of parallel processor.

Because of the nature of the intended use of the processor, emphasis was placed on network simplicity, on reduction of size and power, and especially, on reliability. While the processor organization was designed in terms of a particular mechanization—wire memory and integrated circuitry—the organization and algorithms are described here in general terms, and questions of mechanization are postponed to a final section.

When the fundamental limits of electrical and optical signal propagation speeds are reached, there are just two ways to further reduce the time to perform a given computation. One of these is by making things smaller, and the other is by performing parallel processing. But efforts to achieve efficient parallel processors have encountered several difficulties. First is the problem of providing sufficient memory and computing capability within a simple module. Some parallel processors, such as the Holland<sup>1</sup> machine, have employed relatively

simple modules, but the memory capacity and computing capability of each module were limited. Others, such as the Solomon Computer<sup>2</sup>, provide greater memory capacity and computing capability in the module, but each module approaches the complexity of a small computer.

Another serious problem is that of communication. For a periodic computing structure to be useful, it is essential that there be efficient paths for the communication of control signals and operands among the modules. In some parallel processors, the communication networks are more complex than the processing modules themselves.

The associative memory suggest itself as a basis for another approach to the problem of parallel processing. Logical operations are performed within the individual memory cells of this memory, and communication within the structure is particularly efficient. Extension of these principles to permit full logical and arithmetic capability within each memory cell would provide a high degree of processing parallelism. We shall call an associative memory structure and its control logic, which is capable of performing such distributed computation, an Associative Processor.

In addition to the parallel computing capability, there are several other advantages which one may expect to achieve in the Associative Processor. These are:

1. The data storage and retrieval capabilities of the Associative Memory, which greatly simplify or eliminate such common data manipulations as sorting, col-



- lating, searching, matching, cross referencing, updating and list processing;
2. Programming simplifications based upon the possibility of ignoring the placement of data in memory and the extensive use of content addressing and ordered retrieval;
  3. The periodic structure of a large portion of the processor. Periodicity of structure lends itself to integrated circuit techniques and batch fabrication. Interconnections between components become shorter and less tangled, reducing propagation delays and simplifying layout and checkout. Since the structure is periodic, it can easily be expanded in size;
  4. Fault Tolerance. The periodic structure may permit an organization which is tolerant of memory or circuit element failures. If a cell fails, it may be possible to avoid its further use with little loss to the system capability. A program for an associative structure makes little or no reference to a unique cell so that loss of a cell would not confuse the program.

Two approaches have been taken in the past to solve the problems of parallel processing by using associative processing techniques. Rosin<sup>3</sup> and Fuller<sup>4,5</sup> have considered an associative memory under control of a general purpose computer. In Fuller's work, algorithms for a variety of arithmetic operations are built up as sequences of elementary operations performed by the rather limited word logic of the associative memory. In Davies<sup>6</sup>, more extensive word logic is provided, and the control is integrated into the associative processor. The present paper represents an attempt to achieve the higher speed of the second approach with a considerably simpler logical structure, which could be mechanized from non-cryogenic components.

The design which was adopted provides a random access memory for program storage and a bit serial associative memory for data storage and parallel processing. The ability to write tags (i.e. to simultaneously write data in a selected bit position of a number of selected words), coupled with simplified word logic networks, permits relatively efficient bit serial algorithms for many kinds of parallel searches,

parallel arithmetic and ordered retrieval. Methods were developed for treating certain classes of memory and circuit failures. For these cases, the processor can continue to operate in spite of a failure with only slight impairment of the overall system capability. In the area of communication, methods were developed for treating operand pairs in a variety of relative locations.

## 2. MEMORY DISTRIBUTED LOGIC

One of the fundamental features of the associative memory is that logical operations are performed within the memory cells. However, even in the random access memory a limited amount of logic is performed in the memory cell. The boolean function  $X_i \cdot Y_j \cdot S_{ij}$  is performed, where  $X_i$  is the selected X address coordinate,  $Y_j$  the selected Y address coordinate, and  $S_{ij}$  the bit stored at location  $ij$ . The value of the function is read out on the sense line. In an associative memory, the memory logic is extended to permit selection of a memory cell on the basis of stored data. In some associative memories, this is accomplished by the function

$$(S_1 \leftrightarrow R_1) \cdot (S_2 \leftrightarrow R_2) \cdots (S_n \leftrightarrow R_n)$$

which is mechanized in each memory word cell. " $S_i \leftrightarrow R_i$ ", the equivalence function, is the same as  $S_i \cdot R_i + \bar{S}_i \cdot \bar{R}_i$ .  $S_i$  is the bit stored in the  $i$ -th bit position of a typical word, while  $R_i$  is the corresponding bit of a reference word stored in an external register. The function selects all words whose stored contents match the reference word. This can be improved to permit masking of selected bits as follows:

$$[(S_1 \leftrightarrow R_1) + M_1] [(S_2 \leftrightarrow R_2) + M_2] \cdots [(S_n \leftrightarrow R_n) + M_n]$$

where  $M_i$  indicates whether the  $i$ -th bit is to be ignored in the comparison.

In addition to providing logic in each memory bit position, it is also profitable to have logic associated with each word cell. This is the case in certain word organized random access memories and in associative memories. In the first case, there is the word driver which may be a magnetic or semiconductor amplifier which responds to X and Y coordinate selection lines just as the typical bit cell does in a coincident current memory. In associative memories, there is usually a match detector with each

word which responds to the match logic described above. Ordinarily, the match detector has memory. These operations at both the bit level and the word level suggest the possibility of providing sufficient distributed logic to permit parallel computation throughout the memory structure.

In arriving at an Associative Processor capable of such parallel computation a number of important decisions must be made. One basic choice is whether to use a separate random access memory or the associative memory itself for program storage. The first choice is probably more practical since the random access memory is less expensive; furthermore, it will be easier to protect the associative portion from fault if the program is kept separate.

A second choice to be made is between bit parallel and bit serial operation. Certain associative operations such as the matching of fields for equality can be performed in bit parallel. On the other hand, to perform the more complex functions of arithmetic, it appears more convenient to use the bit serial approach, simplifying the bit cell by time sharing one logic module among all bits of a word.

A third problem is that of communication. To perform parallel computation, one must have access to the operands and operand pairs. In some cases, the operand pairs are stored together in the same word. In other cases, they are in adjacent words, while in still others, they are in non-adjacent words, but always some fixed number of words apart. Another common requirement involves operand pairs in which the first operand of each pair is common while the second operands are distinct. In this case, the common operand, in an external register, must be communicated to the others, stored in various memory cells. Still another communication problem is based upon the fact that while large portions of a problem may be susceptible to parallel processing, other parts may be essentially sequential. These also must be performed efficiently by the Associative Processor if they are not to offset the advantages gained in the parallel processing.

Techniques for solving these communication problems include the following:

1. Transmission of a common operand to all memory word cells.

2. Flexible control of field selection to permit operation on pairs of operands in the same words.
3. Use of shift registers for communication between words. These can be uni-directional or bi-directional and can be extended to two or more dimensions to give greater flexibility.
4. Forms of entry-exit ladder networks which permit rapid communication between non-adjacent word cells.

The following sections will describe the Associative Processor, which is based upon specific choices of these options.

### 3. ORGANIZATION

A block diagram of the Associative Processor is shown in Figure 1. It contains both a conventional random access memory (RAM) and an associative memory. The RAM provides storage for instructions and constants; it is accessed parallel by bit and serial by word. In processing operations, the Associative Memory is accessed parallel by word and serial by bit. In the organization under consideration, RAM contains 4000 twenty-four bit words, and the Associative Memory contains 500 ninety-six bit words.

Instructions accessed from RAM are transferred to the Instruction Register where they are held during execution. The D-Register,

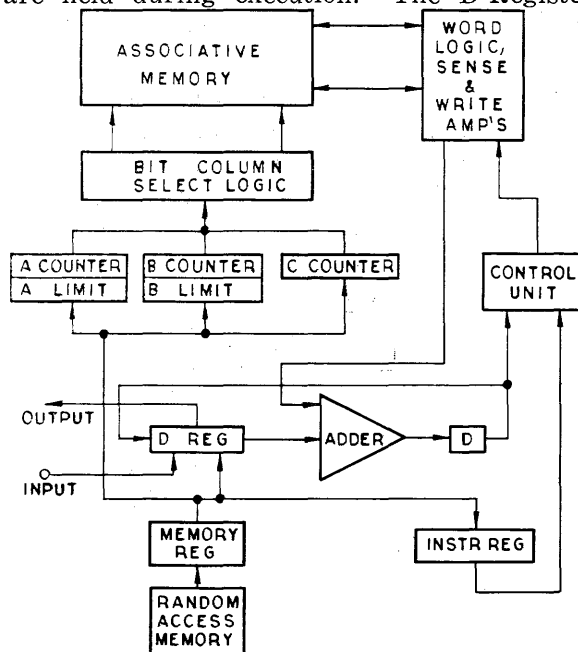


Figure 1. Block Diagram of Associative Processor.

which has the same length as a RAM word, serves as temporary storage for operands which participate in associative operations. For instance, the D-Register may hold the argument of a search, may receive data being retrieved from the Associative Memory, or may communicate with the external world. Data originating from outside of the Associative Processor can be transferred directly to either the Associative Memory or RAM. Direct input to the memories is under an automatic interrupt control.

In the Associative Memory, only one bit column at a time may be operated upon. The particular bit column is selected by either the A Counter, the B Counter, or the C Counter. Associated with the A and B Counters are the A and B Limit Registers. Each may contain a value which serves to define a maximum or minimum value of its companion counter. Together, each counter and limit register define a field which can be any length up to the number of bits in the Associative Memory word, and may overlap the field defined by the other counter and limit register.

The design of the Associative Processor is sufficiently general to permit implementation by a variety of memory elements and logic techniques. Therefore, the following description of the Associative Memory, shown in Figure 2, will present those characteristics which are essential to the design of the Associative Processor.

Storage for one bit is provided at each intersection of a word and a bit line. A pulse on a bit line causes a signal to be emitted by each bit on that line. The signals are transmitted through the word lines to the sense amplifiers.

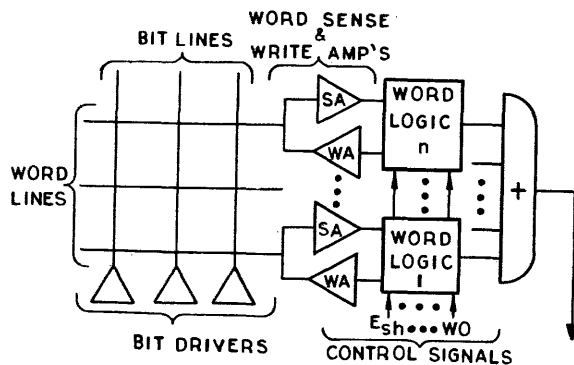


Figure 2. Associative Memory.

The equivalence function is obtained in one of two ways depending upon the particular memory element. In some memories it is sufficient to exercise control over the polarity of the interrogating pulse, thereby achieving a signal output for a match and no output for a mismatch. In these cases, the bit element itself performs the equivalence function,  $S \leftrightarrow R$ . In other memories, the stored bit is merely read out; the reference bit is transmitted to all sense amplifiers and logic associated with each sense amplifier generates the equivalence function.

Writing at a particular bit location is accomplished by passing a current through the intersecting bit and word lines. The polarity of the current in the word line, or in some cases the word and bit lines, determine the state of the written bit. By energizing all the word drivers and one bit driver, one bit of each word can be written into. The latter operation, which is sometimes referred to as "tagging", plays a significant role in the design of the Associative Processor.

The logic associated with each word gives great power to the Associative Processor. This logic is identical for all words and consists of a sense amplifier, storage flip-flop, write amplifier, and control logic. Refer to Figure 3.3. The sense amplifier is bistable and remembers the match state from one interrogation to the next. The output of the sense amplifier determines the state of the storage flip-flop in various ways as determined by the control signals,  $E_s$ ,  $E_r$ , and  $E_c$ . In addition, the contents of each storage flip-flop can be shifted to the storage flip-flop in the word above under control of the signal,  $E_{sh}$ . This provides communication between words.

One of the functions of the storage flip-flops is to control writing. In this operation, the storage flip-flops in the "1" state select the words that are to be written into, while the signals,  $W_1$  and  $W_0$ , determine whether "1's" or "0's" are written by the selected write amplifiers. In addition, the output of each storage flip-flop is "ANDed" with the output of the corresponding sense amplifier. The outputs of these AND gates are ORed together to provide an output channel from the Associative Memory to the remainder of the Processor.

Control of the word logic networks is exercised through a Control Unit. This unit interprets the contents of the Instruction Register and D Flip-flop to determine the control signals, Es, Er, Ec, Esh, W1, and W0 which are transmitted to all word logic networks.

The D flip-flop and D Register are the data link between the Associative Memory and the Random Access Memory. Data, such as the argument for a search, are transferred from RAM in parallel to the D-Register. Each bit is then shifted into the D Flip-flop where it participates in the search operation. Data retrieved from the Associative Memory are transferred through an adder to the D Flip-flop and then to the D Register.

The Associative Processor offers a variety of processing options in terms of operand location and processing speed. The following list illustrates some of the possibilities:

1.  $D + M \rightarrow D$
2.  $D + M_j \rightarrow M_j$
3.  $M_j + M_j \rightarrow M_j$
4.  $M_j + M_k \rightarrow M_k$

(1) represents an operation occurring between the D Register and one selected word in the Associative Memory. The result goes to the D Register. (2) illustrates a process between D Register and many words in the Associative Memory. The third operation occurs between pairs of operands, each pair stored in a separate word. (4) represents an operation occurring between operands in different words. The same operation may simultaneously occur in many such pairs of words. In addition to these operations, many variations are possible, e.g. the operands may be located in different words with the results going to a third word.

The capability of the storage flip-flops to act as a shift register provides the communication link between adjacent words. Another use of this shift register occurs in counting the number of words which satisfy a search algorithm. This is accomplished by operating the storage flip-flops as a shift register and counting the number of "1's" shifted out. Each "1" corresponds to a word that satisfies the search.

#### 4. COMMAND STRUCTURE

There are two types of instructions in the Associative Processor. Instructions which exercise control over the Associative Memory shall be referred to as associative instructions. Instructions which provide access to the Random Access Memory or that perform control transfers shall be referred to as non-associative instructions. A list of non-associative instructions follows:

- LA Load the contents of memory location M into the A-Counter and Limit Register.
- LB Load the contents of memory location M into the B-Counter and Limit Register.
- LC Load the contents of memory location M into the C-Counter.
- LD Load the contents of memory location M into the D-Register.
- LM Load the contents of the D Register into Memory Location M.
- TD Transfer to location M if the D flip-flop equals "zero", otherwise proceed sequentially.
- TO Transfer to location M if the output of the OR gate equals "zero", otherwise proceed sequentially.
- TI Transfer to the location specified by memory location M.
- TU Unconditionally transfer to location M.
- SH Up-shift the storage flip-flops a number of times equal to M.
- SC Up-shift the storage flip-flops a number of times equal to M. The C Counter counts the number of ones shifted into the highest level storage flip-flop.
- CD Transfer the contents of the C Counter to the D Register.
- ID Input data word from external device to the D register\*.
- OD Output data word from the D register to external device\*.

\*The input and output commands generally work in conjunction with the automatic interrupt facility. An external device requests an interruption by turning on an interrupt flip-flop. This causes the Processor to complete the present instruction, store the contents of the Instruction Address Counter in memory, and jump to an Input or Output routine. These routines can transfer I-O data between the D Register and either the RAM or the Associative Memory.

CS	CC	TC	AC	RS	IC	WC	SC	IT	RM
COLUMN SELECT	COUNTER CONTROL	TRANSFER CONTROL	ADDER CONTROL	D REGISTER SHIFT	INTERROGATE CONTROL	WRITE CONTROL	STORAGE F.F. CONTROL	INSTRUCTION TYPE	REPEAT MODE

Each associative instruction controls the processing during a single bit time, except when it is executed in a Repeat Mode. The instructions are divided into a number of fields, each of which specifies the control of a separate part of the Processor. Figure 3 summarizes these fields which are described below:

*Column Select (CS)*: The contents of this field determine what bit column of the Associative Memory is to be interrogated or written into, either by specifying the A, B, or C counter, which in turn selects the bit column, or by directly specifying one of the four bit columns. The four directly specified columns are ordinarily used for the storage of tag bits.

- A A counter
- B B counter
- C C counter
- T1 Column 1
- T2 Column 2
- T3 Column 3
- T4 Column 4

*Counter Control (CC)*: The contents of this field determine whether the counter selected in CS will be modified. A counter may be decremented or incremented by one.

- IN Increment
- DE Decrement
- NC No Change

*Transfer Control (TC)*: In general, instructions are accessed from sequential memory locations in RAM. To facilitate exiting from a subroutine, it is desirable to be able to transfer to another location when the contents of a counter become equal to the associated Limit Register.

- Ta Transfer to memory location 0 when the A-Counter becomes equal to the A-Limit Register

- Tb Transfer to memory location 1 when the B-Counter becomes equal to the B Limit Register

NC Proceed sequentially.

*Adder Control (AC)*: The contents of this field control the manner in which the output of the OR gate is transferred into the D Flip-flop.

- L OR gate output copied into the D Flip-flop.
- C Complemented OR gate output copied into the D Flip-flop.
- A OR gate output added to the D Flip-flop. The carry is stored in a flip-flop associated with the Adder.
- S OR gate output subtracted from the D Flip-flop
- NC No transfer

*D-Register Shift (RS)*: The D Register can be made to shift one bit in either direction. The shift is end around when the contents of the AC field indicate that no transfer is to take place.

- R Right shift
- L Left shift
- NC No Change

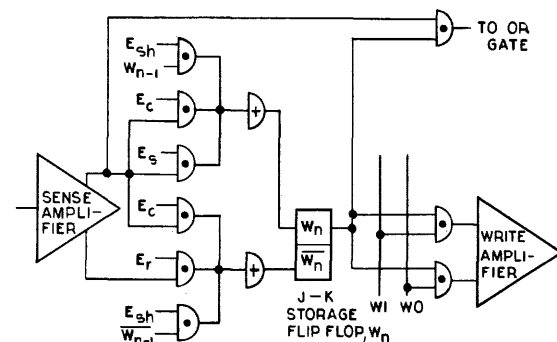


Figure 3. Word Logic.

*Interrogate Control (IC)*: Upon interrogation, the sense amplifier responds with a "1" output to a match condition between the interrogated bit and what previously has been labeled a reference bit. The IC field defines the reference bit:

- I Interrogate for "1". If the stored bit is equal to "1", the sense amplifier will be set.
- Z Interrogate for "0". If the stored bit is equal to "0", the sense amplifier will be set.
- $\bar{D}$  If the D flip-flop is equal to "1", interrogate for "1", if "0", interrogate for "0".
- D If the D flip-flop is equal to "0", interrogate for "1", if "1", interrogate for "0".

*Write Control (WC)*: This field specifies writing to occur in the words for which the storage flip-flop is equal to "1". During writing, the IC field is available to determine whether "1's" or "0's" are to be written.

- W—Write
- NC—Do not Write

*Storage Flip-flop Control (SC)*: This field specifies the state of the control signals which are common to the input logic of each of the storage flip-flops. This logic influences the transfer of data from the sense amplifiers to the storage flip-flops.

- NC 0→Es, 0→Er  
No transfer takes place.
- Es 1→Es, 0→Er  
The Storage flip-flop is set if the sense amplifier is equal to "1".
- Er 0→Es, 1→Er  
The Storage flip-flop is reset if the sense amplifier is equal to "0".
- Esr 1→Es, 1→Er  
The state of the sense amplifier is copied by the storage flip-flop.
- D 1→Es, 0→Er,  
if the D flip-flop is equal to "1".  
0→Es, 1→Er,  
if the D flip-flop is equal to "0".

- $\bar{D}$  0→Es, 1→Er,  
if the D flip-flop is equal to "1".  
1→Es, 0→Er,  
if the D flip-flop is equal to "0".
- OR 1→Er  
if the output of the OR gate is equal to "1".
- Ec 1→Ec  
The storage flip-flop is complemented if the sense amplifier is equal to "1".

*Instruction Type (IT)*: This field appears in both associative and non-associative instructions. The contents designate the instruction as being associative or not.

- A Associative
- $\bar{A}$  Non-associative

*Repeat Mode (RM)*: It is sometimes desirable to repeat an instruction during the execution of a simple search.

- $\bar{R}$  Repeat until the counter specified by CS becomes equal to its limit register.
- R Do not repeat instruction.

The time required to execute one associative instruction is measured from the time the instruction is transferred into the Instruction Register to the time the sense amplifier outputs are transferred into the storage flip-flops. During this time, the next instruction is accessed, and the previous output of the storage flip-flops can be transferred to the D flip-flop. This time will be referred to as a "bit time". Associative instructions are accessed at a rate of one per bit time. It should be noted that the AC field of the associative instruction will control the disposition of storage flip-flop data that resulted from an interrogation specified by the previous associative instruction. Non-associative data transferring instructions require two bit times for execution (Both an instruction and an operand must be accessed from the Random Access Memory). Non-associative instructions which transfer control require one bit time for execution.

## 5. MICROPROGRAMMED ALGORITHMS

The method by which associative instructions are controlled constitutes one of the

major factors contributing to the flexibility of the Associative Processor. Each field of the instruction directly specifies some control function, so that numerous associative instructions can be micro-programmed by the appropriate selection of fields. Despite the large number of options available, the central control unit is very simple since the control functions are obtained directly from the instruction fields.

Following, is a description of several important categories of associative instructions with typical microprograms; algorithms are also given for more complex data retrieval and arithmetic processes built up for microprogrammed associative instructions.

Possibly the most often used algorithm is ordered retrieval. A number of algorithms for retrieval have appeared in recent publications. Among these, the algorithm presented by Lewin<sup>11</sup> appears to be fastest, making its' implementation in the Associative Processor an attractive consideration. However, in view of its' unique hardware requirement, (i.e., the equivalent of a three state sense amplifier on each bit column), an algorithm was developed which utilizes logic of a more general nature. In fact, the development of this algorithm greatly influenced the design of the word logic. The algorithm, presented in detail later in this section, retrieves one bit of data for each bit time of execution.

The ordered retrieval algorithm is used whenever data is to be retrieved from the Associative Memory or whenever it is necessary to select a word in which to write data. Since the time required to identify a word is related to the number of bits that must be searched, it is often desirable to have stored in each word a compact address field. Having such a field also provides a convenient way to distinguish between two words which might otherwise contain the same data.

In most instances, before the execution of an associative search, it is necessary to precondition the storage flip-flops either by setting or resetting them all, or by setting those corresponding to the set of words which is to be searched. Accomplishing this last operation requires transferring the contents of the tag bit (or whichever bit position holds the infor-

mation) into the sense amplifiers and then copying the states of the sense amplifiers into the storage flip-flops. These operations can be executed with a single associative instruction.

CS	CC	TC	AC	RS	IC	WC	SC	IT	RM
T1	NC	NC	NC	NC	1	NC	Esr	A	$\bar{R}$

Setting (or resetting) all the storage flip-flops requires two associative instructions. The procedure is to interrogate the same bit column twice; once for "1's", and once for "0's". The following instructions reset the storage flip-flops.

CS	CC	TC	AC	RS	IC	WC	SC	IT	RM
1. T1	NC	NC	NC	NC	Z	NC	Er	A	$\bar{R}$
2. T1	NC	NC	NC	NC	1	NC	Er	A	$\bar{R}$

The following associative searches have been microprogrammed:

Equality	Maximum value
Less than	Minimum value
Less than or equal	Similarity
Greater than	Ordered Retrieval
Greater than or equal	

Except for Similarity, the execution time or each search is one bit time for each bit of the argument.

The object of most searches is to leave the storage flip-flops in a state which defines the locations of those words which meet the conditions of the search. However, it is possible to obtain the complementary set of words, as well as the set obtained by ORing or ANDing the results of several different searches. Following are a few examples of search microprograms:

Equality Search

CS	CC	TC	AC	RS	IC	WC	SC	IT	RM
A	IN	TA	NC	R	D	NC	Er	A	R

This instruction searches the words in memory whose storage flip-flops are initially true. At the end of the search, the storage flip-flops identify those words containing a field exactly matching the field in the D Register. The field in memory is defined by the A Counter and Limit Register. Each bit of the field is interrogated starting with the least significant bit.

The D flip-flop specifies the match conditions. A mismatch will cause the appropriate storage flip-flop to be reset.

Less Than

CS	CC	TC	AC	RS	IC	WC	SC	IT	RM
A	IN	NC	NC	R	Z	NC	D	A	R

This instruction causes a storage flip-flop to be set if the interrogated memory bit is "0" and the D flip-flop "1" and reset if the memory bit is "1" and the D flip-flop "0". Otherwise the storage flip-flops are unchanged. Essentially, this logic is the same as borrow logic with the contents of the D Register being subtracted from the contents of each memory word. When the storage flip-flop is equal to one, the memory word is less than the data register word.

Ordered Retrieval

CS	CC	TC	AC	RS	IC	WC	SC	IT	RM
A	DE	NC	L	L	1	NC	Or	A	R

This instruction transfers to the D Register the maximum value field of the set of fields which are identified by the true storage flip-flops, Starting with the most significant bit position of the search field, each bit position is sequentially interrogated for a one. When any sense amplifier indicates a match for a field still in the search set, the storage flip-flops corresponding to those sense amplifiers indicating a mismatch are reset. Each time a bit position is interrogated and is found to contain a "1" in any of the words remaining in the search set, a "1" is transferred into the D flip-flop.

Many arithmetic and logical microprograms have been developed for the Associative Processor. Below is a partial list. The operations are identified by S when they apply to an operation between a single pair of operands. An SP refers to an operation between one operand in the D Register and many operands in the memory, and P refers to simultaneous operations between many pairs of operands in memory.

The number of bit times required for the execution of each operation appears in parenthesis

Add  $M + D \rightarrow D$   
 S (1 + no. of operand bits)

Add  $D + M \rightarrow M$   
 SP (12 × no. of operand bits)  
 Add  $M_1 + M_2 \rightarrow M_2$   
 P (12 × no. of operand bits)  
 Multiply  $M_1 \times M_2 \rightarrow D, M_3$   
 S (20 × no. of multiplier bits)  
 for a 24 bit multiplicand  
 Multiply  $M_1 \times M_2 \rightarrow M_3$   
 P (no. of multiplier bits × no. of multiplicand bits)  
 Divide  $D/M_1 \rightarrow M_2$   
 S (30 × no. of operand bits)  
 Add  $(M_1 + M_2 \rightarrow M_2)$

Field  $M_1$  is added to field  $M_2$  in all words which contain a "1" in bit column (T1). Field  $M_1$  is defined by the A counter and Limit Register. Field  $M_2$  is defined by the B counter. Bit column T2 is temporary storage for the carry. Addition is executed in the following steps:

1. The  $j^{th}$  bit of  $M_1$  is transferred to the storage flip-flops.
2. The contents of the storage flip-flops are added to the  $j^{th}$  bit of  $M_2$ . The carry is developed in the storage flip-flops during the addition.
3. The partial carry resulting from the carry addition (preceding step 1) to the  $j^{th}$  bit of  $M_2$  is 0Red with the partial carry in the storage flip-flop. The final carry results in the storage flip-flop.
4. The carry is added to the  $j+1$  bit of  $M_2$ ; the resulting partial carry is stored in T2.

The following program executes this addition algorithm:

	CS	CC	TC	AC	RS	IC	WC	SC	IT	RM
1.	T1	NC	NC	NC	NC	1	NC	Esr	A	$\bar{R}$
2.	A	IN	NC	NC	NC	1	NC	Er	A	$\bar{R}$
3.	B	NC	NC	NC	NC	1	NC	NC	A	$\bar{R}$
4.	B	NC	NC	NC	NC	1	W	Er	A	$\bar{R}$
5.	B	IN	TB	NC	NC	0	W	NC	A	$\bar{R}$
6.	T2	NC	NC	NC	NC	1	NC	Es	A	$\bar{R}$
7.	T2	NC	NC	NC	NC	0	W	NC	A	$\bar{R}$
8.	B	NC	NC	NC	NC	1	NC	NC	A	$\bar{R}$
9.	B	NC	NC	NC	NC	1	W	Er	A	$\bar{R}$
10.	B	NC	NC	NC	NC	0	W	NC	A	$\bar{R}$
11.	T2	NC	NC	NC	NC	1	W	NC	A	$\bar{R}$
12.	Transfer to 1 (TU)									$\bar{A}$



Instruction one transfers T1 to the storage flip-flops. Instruction two interrogates the  $j$ th bit of  $M_1$  and resets the storage flip-flop where mismatches occur. The resulting contents of the storage flip-flops constitute the AND function of T1 and the  $j$ th bit of  $M_1$ . Instructions three, four, and five have the effect of complementing the  $j$ th bit of  $M_2$  in words for which the storage flip-flop is equal to "1". In addition, if the  $j$ th bit of  $M_2$  were equal to "1", the storage flip-flop would remain equal to "1", thereby representing the partial carry. Instruction six OR's the carry resulting from addition of the last carry to the  $j$ th bit of  $M_2$  to the partial carry in the storage flip-flop. Instruction seven clears the T2 column in preparation for the next carry storage. Instructions eight, nine, and ten add the carry to the  $j+1$  bit of  $M_2$  using the same technique as instructions three, four, and five. Instruction eleven stores the partial carry in T2. The routine is exited after instruction five has been executed and the addition of the most significant bits completed.

## 6. FAULT TOLERANCE

An interesting characteristic of this particular associative memory is its structural periodicity. Each bit driver is identical to any other, and the logic of each word is identical to that of any other. There is no addressing matrix and no ladder network. The existence of these characteristics suggest the possibility of making the system operation insensitive to local malfunctions in the memory stack and memory circuits.

There are numerous possible causes of malfunction in the Associative Memory. However, most malfunctions can be placed in one of four categories. The first is characterized by the inability of the sense amplifier to change state. The second is characterized by the inability of the storage flip-flop to change state. The third category consists of those malfunctions which cause a write amplifier to fail, and the fourth consists of those malfunctions which cause a bit driver to fail.

The procedures to be described below consist of exercising control over the functions common to the logic of each word in such a way as to gain a system tolerance to these malfunctions. Other malfunctions may occur for which

the only safeguard would be the utilization of component redundancy techniques. In the following discussion it will be assumed that no more than one type of malfunction exists in any one word.

To cope with these malfunctions, it is of primary importance to guard against spurious results in the retrieval operations. It is of little importance if, in a particular word, some other operation goes awry as a result of a malfunction. This merely produces meaningless data in that word, which is unimportant if provisions are made never to write into and never to retrieve information from such a word. Since a word is selected for writing by means of the retrieval algorithm, the fundamental problem is to guard against incorrect retrieval as a result of a malfunction.

Malfunctions of the first category cause the sense amplifier to permanently store either a "1" or a "0". If a "0" is stored, the storage flip-flop can never be set. The retrieval of data from a particular word and the ability to write into a particular word are dependent upon the storage flip-flop of that word being in the "1" state. If the storage flip-flop can never be set, the word appears to be nonexistent. If information were stored in the word prior to the malfunction, it would become irretrievable; however, as long as the malfunction existed it would be impossible for data to be inadvertently stored in that location.

A "1" locked in the sense amplifier presents a different problem. If the storage flip-flop of such a malfunctioning word is true, execution of the retrieval algorithm will result in the retrieval of a word of "1's". To prevent this, it is necessary to perform an operation prior to the retrieval algorithm which will reset the storage flip-flops in words whose sense amplifiers are locked in the "1" state without altering the states of the *other* storage flip-flops. Furthermore, this operation must not turn on a storage flip-flop in a word whose sense amplifier is locked in the "0" state. This can be accomplished as follows: Reset the sense amplifiers by interrogating a column of "0's". Then by executing the complement control  $E_c$ , complement all storage flip-flops corresponding to sense amplifiers still in the "1" state.

Malfunctions of the second category are those in which the storage flip-flop does not change state. If the storage flip-flop is locked in the "0" state, the associated word can not participate in any reading or writing operations. Such a condition will cause that word to appear to be nonexistent. A permanently stored "1", however, may cause an erroneous readout during execution of the retrieval algorithm. To avoid this, in the case of maximum value retrieval, the affected word should be loaded with "0's" prior to retrieval. A method of determining whether any storage flip-flops are malfunctioning is first to interrogate a column of "1's" so as to set the sense amplifiers, then to execute the  $E_s$  and then  $E_c$  functions. The output of the OR gate will be "1" if any storage flip-flops remain true.

The third category consists of those malfunctions which disable a write amplifier. If a disabled write amplifier can be detected and the word uniquely marked, further operations upon that word can be avoided. Detection is accomplished by writing a pattern of "1's" and "0's" in each word. An equality search is then made using the same pattern to identify any words in which the pattern was not successfully recorded. On each successive execution of this procedure, a pattern different from the last pattern must be used. If the pattern which is read out of a given word is not identical to the current pattern, then the write amplifier driving that word is malfunctioning. The erroneous pattern cannot be used again. If the length of the pattern is  $N$  bits, then  $N$  bits in each word of the memory must be relegated to storage of the checking patterns at the time of execution of the checking procedure. The number of different patterns must exceed by two the number of malfunctions which can be tolerated.

The malfunctions of the last category are associated with the bit drivers. Once a bit driver has failed there is no way of writing into or interrogating that particular bit column. Therefore, it is necessary to isolate malfunctioning bit drivers. Detection of a malfunctioning bit driver is accomplished by designating two words of memory as test words, one of which would contain stored "1's" and the other, stored "0's". Special logic on each of these two word

lines would compare the real output to the theoretical output upon each interrogation. A discrepancy would interrupt the program, thereby allowing the execution of a programmed corrective action.

## 7. MECHANIZATION

The critical problem in mechanizing the Associative Processor is, of course, the implementation of the associative memory. The critical requirements of this memory are the following:

1. Non-destructive readout. This is essential for the Processor described above; however, with slight modification, destructive readout could be tolerated provided the write time was comparable to the read time.
2. Small ratio of word write current to read signal. This significantly influences the complexity of the sense amplifier and write amplifier and limits the number of words in memory.
3. Short write cycle. This makes tagging operations practical.
4. Short interrogation cycle. This is especially important for a bit serial processor.
5. Limited power consumption.

A number of memories were analyzed for compliance with these requirements, including:

1. Plated Wire <sup>7,8</sup>
2. Laminated Ferrite <sup>9</sup>
3. Bi-core
4. Biax

At this time, the most promising of these for both the Associative Memory and RAM appears to be the Plated Wire Memory. It can be operated in a nondestructive readout mode, requires a word current of approximately 25 ma, and can be interrogated or written into at a 10 mc rate.

The closed flux path, rotational switching mode and loose coupling of the switched flux to the sense line contribute to the high ratio of read signal to word write current and to the low power consumption of each bit.

Our laboratory evaluation of the Plated Wire has indicated the feasibility of using integrated

circuitry for both the word logic and the bit drivers. The periodic structure of large portions of the Associative Processor and the requirement for a relatively small number of circuit types facilitate mechanization with integrated circuits.

The optimum sizes of the Associative Memory and the Random Access Memory depend greatly upon the application. For the class of Aerospace applications for which the Processor was conceived, the following dimensions and parameters were chosen:

RAM: 4096 words of 24 bits each.

Associative Memory: 512 words of 96 bits each.

Bit time = Memory cycle time =  $.1 \mu\text{sec}$ .

## 8. CONCLUSIONS

The Associative Processor possesses several virtues as a parallel processor. The basic processing module, i.e. one word of Associative Memory with its word logic, possesses considerable computing and memory capability for its size and complexity. This implies a large amount of parallel processing per dollar. Communication within the Processor is relatively efficient, especially where associative techniques can be employed. Most of the Processor is periodic in structure and, therefore, compatible with batch fabrication techniques and integrated circuitry. Fault tolerance techniques can be employed, at least to a degree. The non-periodic control structure of the Processor is relatively simple. And, finally, the microprogramming characteristics of the instructions permit and encourage programming experiments.

## ACKNOWLEDGEMENT

The authors gratefully acknowledge the support of the Space Systems Division of the Air Force under whose study contract the work was done and Space Technology Laboratories, Inc., who participated with Abacus, Inc. in the study.

We wish particularly to thank Mr. M. Waxman of S.T.L. for his work in evaluating the Associative Processor in several aerospace ap-

plications and Mr. T. Stupar who assisted in evaluating the technical feasibility of the system.

## REFERENCES

1. HOLLAND, J. H., "A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously," Proc. Eastern Joint Computer Conference, (1959).
2. SLOTNICK, D. L., BORCK, W. C., and MC-REYNOLDS, "The Solomon Computer," Proc. Fall Joint Computer Conference, (1962).
3. ROSIN, R. J., "An Organization of an Associative Cryogenic Computer," Proc. Spring Joint Computer Conference, San Francisco, (May 1962).
4. ESTRIN, G. and FULLER, R., "Algorithms for Content Addressable Memory Organizations," Proc. Pacific Computer Conference, Pasadena, (March 1963).
5. ESTRIN, G. and FULLER, R., "Some Applications for Content-Addressable Memories," Proc. Fall Joint Computer Conference, Las Vegas. (Nov. 1963).
6. DAVIES, P. M., "Design for an Associative Computer," Proc. Pacific Computer Conference, Pasadena, (March 1963).
7. I. DANYLCHUCK, A. J., PERNESKI, M. W. SAGAL, "Plated Wire Magnetic Film Memories," Intermag Proceedings, Washington, D.C., (April 1964).
8. K. FUTAMI, et al., "The Plated-Woven Wire Memory Matrix", Intermag Proceedings, Washington, D.C., (April 1964).
9. R. SHAHBENDER, et al., "Laminated Ferrite Memory." Proc. Fall Joint Computer Conference, Las Vegas, (Nov. 1963).
10. C. A. ROWLAND, W. O. BUGE, "A 300 Nano-second Search Memory," Proc. Fall Joint Computer Conference, Las Vegas, (Nov. 1963).
11. M. H. LEWIN, "Retrieval of Ordered Lists from a Content-Addressed Memory," RCA Review, (June 1962).

# A HARDWARE-INTEGRATED GPC/SEARCH MEMORY

*Russell G. Gall*

*Goodyear Aerospace Corporation, Akron, Ohio*

## SECTION I—INTRODUCTION

A search memory that can be operated in conjunction with a USQ-20 computer using only the standard input-output channels is being developed by Goodyear Aerospace Corporation. This approach is referred to as the peripheral search memory. On the other extreme, there is the possibility of completely integrating the search memory with the USQ-20 computer. That is, the instruction repertoire can be modified to include associative instructions, the control logic can be modified extensively, and additional registers can be added as necessary. Neither integration can be considered the optimum since the latter involves costs that are out of proportion with the advantages, and the former involves undue transfer time penalties and more complex programming.

This paper presents a method of integrating the search memory with the Univac 1206 (AN/USQ-20) computer more intimately than is possible through a standard input-output channel. The saving of search time that results approaches that of a completely integrated system. The hardware modifications required are relatively minor, and therefore increases in cost are held to a minimum. A typical four-variable search problem is formulated and its solution by the following three separate systems is analyzed: (1) AN/USQ-20 computer, (2) AN/USQ-20 computer in conjunction with a peripheral search memory tied to a standard I/O channel of the computer, and (3) the proposed hardware-integrated USQ-20/search memory. Search solution times are derived for each of the three systems. They are displayed

as curves for ease of comparison of performance of the three systems.

While the integration methods described could conceivably be applied with some study to any general-purpose computer, this document is particularly oriented toward integration of the Goodyear Aerospace Corporation (GAC) search memory with the Univac 1206 general-purpose computer. The actual military designation of this computer is CP642A/USQ-20(V). This indicates that it is one of a number of components included under the designation AN/USQ-20(V). USQ-20 is a general term used informally to identify this computer and is used throughout this paper.

## SECTION II—SEARCH MEMORY DESCRIPTION

### 1. GENERAL

The particular search memory model upon which this study is based is shown in block diagram form in Figure 1. A brief functional description is considered sufficient background for understanding the ideas presented herein. More detailed information is available in the literature.<sup>1</sup>

The memory proper has a capacity of 256 words. Each word contains 30 bits. (The word size of the USQ-20 computer is also 30 bits.) The memory is limited to three types of search:

1. Exact match ( $=$ )
2. Equal to or greater than ( $\geq$ )
3. Equal to or less than ( $\leq$ )

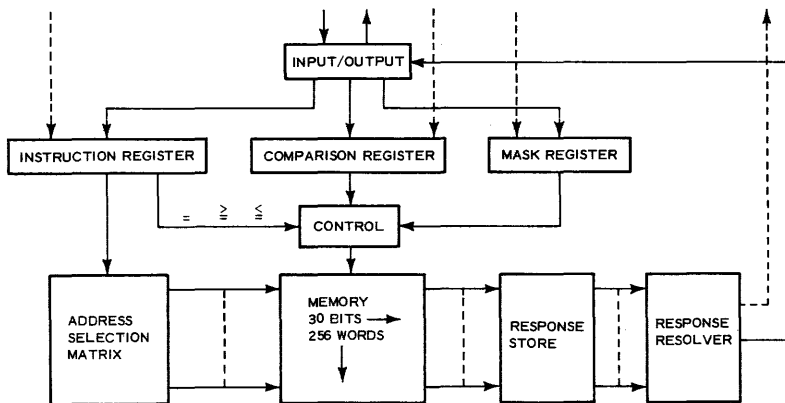


Figure 1. Search Memory Block Diagram

The search types may be combined without limit with the logic AND connective. For example, the equal-to-or-less-than search logically ANDed with the equal-to-or-greater-than search is equivalent to a between-limits search.

2. SEARCH MEMORY INPUT (FROM USQ-20)

a. Information Types

Three general types of information are transferred from the USQ-20 computer to the search memory: instructions, data, and criteria.

b. Instructions

The instruction information consists of a single instruction word, the format of which is shown in Table I. The instruction word is the only type of information that is transferred from the USQ-20 computer via the external function mode of data transfer. The configuration of the instruction word tells the search memory when to start writing data into the memory, when to erase remaining words in

memory, when to accept criteria, what type of search to perform, and what type of response is required.

c. Data

A total of (n) words ( $0 < n \leq 256$ ) of data may be transferred from the USQ-20 computer to the search memory at a maximum rate of 8  $\mu$ sec per word using the internal interrupt mode of data transfer. The USQ-20 program then will be interrupted internally after transferring the n<sup>th</sup> data word to the search memory and will jump to an interrupt routine. This routine will generate an "erase" instruction if required, load the first set of criteria in an output buffer for the forthcoming transfer to the search memory, and initiate the first search instruction. The search memory will accept a search instruction even though it is not finished erasing, although it will not act upon this instruction until erasing has been completed.

d. Criteria

The criteria consists of either (1) a mask word followed by a key word or (2) a key word

TABLE I—INPUT INSTRUCTION WORD FORMAT

		Bit						
$d_{22} \dots d_{15}$		$d_6$	$d_5$	$d_4$	$d_3$	$d_2$	$d_1$	$d_0$
Address (random write or start)	$d_7$ Response count required	Mask word follows	Response required	L/T search	G/T search	Exact match search	Stop write (erase)	Start write

transferred from the USQ-20 to the search memory. The search instruction cues the search memory as to whether the criteria will be transferred according to (1) or (2) (see bit  $d_6$  in Table I). It is considered sufficient that the criteria be transferred from the USQ-20 to the search memory in the normal buffer mode of data transfer. The internal interrupt mode of data transfer is not considered necessary in this case since the USQ-20 need not take further action until an end-of-search signal response is received from the search memory via the input channel.

### 3. SEARCH MEMORY OUTPUT (TO USQ-20)

The output of the search memory can be either response address, response count, or no response. Either may be selected by the instruction. A no-response instruction indicates the response to the present search should be saved for logically ANDing with the next search. Two address responses per 30-bit word are packed for transfer to the USQ-20. There can never be more than a single count response for any given search. It is placed in the least significant portion of the 30-bit word transferred to the USQ-20. The normal buffer mode is used to transfer address or count responses to the USQ-20. An end-of-search signal must be sent to the USQ-20 whether or not an address or count response is desired or available. This signal is transferred via the USQ-20 external interrupt. The end-of-search signal always follows the transfer of address or count responses if any have occurred.

### 4. SEARCH TIME

#### *a. Exact-Match Search*

Search time is variable, depending on the type of search. The exact-match search is performed in 5  $\mu$ sec and is independent of the memory size (number of words), number of words actually loaded into the memory, and the number of unmasked bits searched.

#### *b. Equal-to-or-Less-Than Search*

The equal-to-or-less-than search is actually the result of one or more exact-match searches. The number of exact-match searches performed depends on the number of unmasked zeros in

the key word. Therefore, the total search time is  $5m_0 \mu$ sec, where  $m_0$  is the number of unmasked zeros in the key word.

#### *c. Equal-to-or-Greater-Than Search*

The equal-to-or-greater-than search is similar to the equal-to-or-less-than search, except it depends on the number of unmasked ones in the key word. The total search time then is  $5m_1 \mu$ sec, where  $m_1$  is the number of unmasked ones in the key word.

## 5. ADDITIONAL CONSIDERATIONS

Masked shift load capability is a method of specifying  $n$  high-order bits of the 30-bit USQ-20 word, and specifying a particular field of the search memory into which this  $n$ -bit byte should be entered. Although this actual search memory model does not have this capability, for the purposes of this paper the memory is assumed to have it. Although there are several methods and variations of methods to update the search memory, the masked shift load affords the fastest way to update both the peripheral search memory and the proposed hardware-integrated GPC/search memory. It also allows less complex programming procedures in the USQ-20 computer, which will ease the analysis and evaluation to follow.

The USQ-20 is a one's complement machine. The search memory must have data in straight binary form, from smallest (most negative) value to the greatest (most positive) value to perform the equal-to-or-greater-than and equal-to-or-less-than searches properly. Therefore, the most significant bit of each data word transferred to the search memory must be inverted. This may be handled either by USQ-20 software or search memory hardware. The actual search memory does not incorporate this hardware at present, but is assumed to have this capability to simplify the ideas to be presented.

## SECTION III—PROPOSED METHOD OF INTEGRATION

### 1. SYSTEM CONSIDERATIONS

The hardware-integrated search memory will require relatively minor modifications to both the presently contracted peripheral search memory and the USQ-20 computer. The over-

all system function of the integrated search memory will be identical to that of the peripheral search memory. A block diagram for the integrated search memory is given in Figure 2. The block diagram shows that the integrated search memory system still uses an output channel of the USQ-20 computer. This output channel will provide the following functions:

1. Initial loading of the search memory by means of block transfer (masked shift load)
2. A means of providing instructions to the search memory utilizing the external function mode of data transfer
3. An additional method of updating dynamic data in the search memory by block transfer if considered appropriate in a given system application.

The external function mode of data transfer is a very efficient method of providing instructions to the search memory, since a single programmed instruction in the USQ-20 computer can effect the transfer of a single instruction to the search memory.

The main modifications involve supplying the search memory direct access to several of the

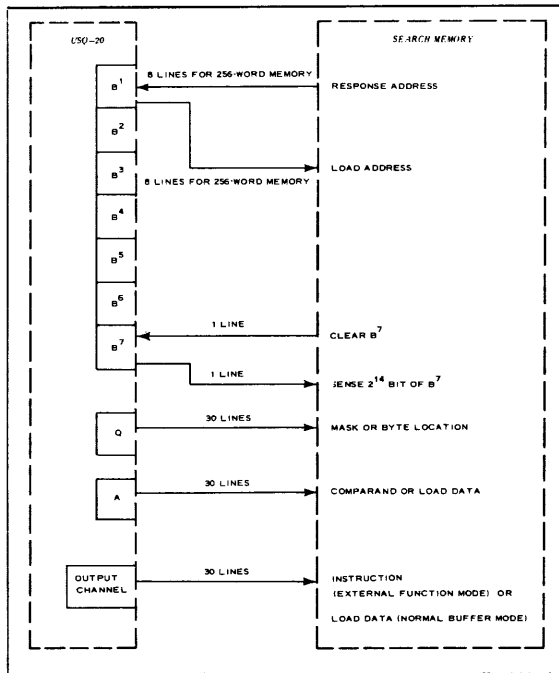


Figure 2. Hardware-Integrated USQ-20/Search Memory Block Diagram

internal registers of the USQ-20. Specifically, output access to the 30-bit A register and output access to the 30-bit Q register is required. In addition, two-way access to the  $B_1$  index register is provided. Although the  $B_1$  register contains 15 bits, only an 8-bit access is necessary for a 256-word search memory. Output access to the  $B_1$  register is not used in the time analysis that follows, but can be useful in many applications. As shown in Figure 2, there are only two lines remaining in the interface between the search memory and the USQ-20. One line allows the search memory to clear the  $B_7$  index register of the USQ-20. The other line allows the search memory to sense the most significant bit of the  $B_7$  index register. These are the only two lines necessary to effect synchronization between the search memory and the USQ-20. Absolutely no modification to the complex control circuitry of the USQ-20 computer is required. Without further explanation, one might reasonably question the fact that only two leads in the interface between the search memory and the  $B_7$  index register could effect complete control and synchronization between the two devices. The answer is, of course, that the two leads cannot alone provide the required control and synchronization of the two devices. However, in conjunction with judicious use of certain USQ-20 instructions, they provide for the control and synchronization of the two devices mainly by economical software utilization rather than expensive hardware modification. The key USQ-20 instruction for synchronization is REPEAT, described as follows: Clear  $B_7$  and transmit the lower 15 bits of Y (the operand) to  $B_7$ . If Y is nonzero, transmit (j) to r (designator register), thereby initiating the repeat mode. This mode executes the instruction immediately following the REPEAT instruction Y times;  $B_7$  contains the number of executions remaining throughout the repeat mode.

The instruction selected to follow the REPEAT instruction is the ENTER  $B_0$  instruction. It was the shortest instruction that could be found in the repertoire, and is normally used as a NO-OP instruction since there is no  $B_0$  register. In the repeat mode, the first execution of the ENTER  $B_0$  consumes 8  $\mu$ sec; each succeeding execution requires only 4.8  $\mu$ sec. The repeat mode described above and the added

single leads (Figure 2) that control and sense the status of the repeat mode synchronize the USQ-20 and the search memory during transfer of search data. Details will be clarified during the analysis to follow later.

## 2. HARDWARE CONSIDERATIONS

### a. General

The hardware considerations discussed below are based on data contributed by R. Horvath.<sup>2</sup> Since synchronization and control of these integrated devices will be handled by judicious use of the USQ-20 software, the only USQ-20 hardware modifications will be those necessary to provide the search memory two-way access with the  $B_1$  and  $B_7$  registers (18 bits), and output access from the A and Q registers (60 bits). The hardware changes that will be necessary to the present peripheral search memory to modify it for use in the integrated system are the circuitry to terminate and gate 69 additional input lines, and circuitry to drive 9 additional output lines.

### b. USQ-20 Modifications

Each additional input to a register stage in the USQ-20 must be ORed with the existing inputs to that stage. This is accomplished by removing the collector resistor and clamp diode from the output transistor of a standard USQ-20 gated input amplifier. The open collector is then directly connected to the proper output side of the given register stage.

Each additional output from the USQ-20 is handled in a straightforward manner by providing a standard USQ-20 data line driver with the feedback circuitry removed to decrease the rise and fall times. Removal of the feedback circuitry may be unnecessary.

All B registers are located in Chassis No. 7 of the USQ-20 computer. The required nine modified gated input amplifiers and the nine modified data line drivers may be located in Chassis No. 8, which has 30 spare card locations available. The interchassis connectors have a sufficient number of spare pins to allow the necessary interchassis wiring.

The A and Q registers are located in Chassis 4 and 5. The 60 modified data line drivers may be located in spare card locations in Chassis 3, 4, 5, and 6.

## SECTION IV—SEARCH TIME ANALYSIS

### 1. GENERAL

In this section, a typical search problem is defined and three separate solution methods are analyzed: (1) USQ-20 computer only, (2) peripheral search memory, and (3) integrated search memory. The resulting equations and curves represent the USQ-20 computer arithmetic time used. The over-all task can be divided into three basic time-consuming steps no matter what the search system configuration might be: (1) loading data and updating dynamic data, (2) providing search criteria and instructions, and (3) handling the resulting responses.

### 2. TYPICAL SEARCH PROBLEM DEFINITION

The typical search problem parameters are shown graphically in Figure 3. The problem is stated verbally as follows. Find the addresses of all items that are hostile AND between the limits of  $X_1$  and  $X_2$  AND between the limits of  $Y_1$  and  $Y_2$  AND equal-to-or-greater than  $Z_2$ . The solution approaches are governed by the following assumptions:

1. Six bits are used to describe each of the four variables for all items.
2. Initially, each of the 4 variables occupies a complete 30-bit word in the USQ-20 memory.
3. Mask word will always be part of the criteria supplied by the USQ-20 computer (that is, criteria will always consist of two words—mask and match).
4. The search memory capacity is 256 words and 30 bits; however, the number of words is treated as a variable in the equations to be developed.
5.  $X_1$  and  $Y_1$  are positive.
6.  $X_2$  and  $Y_2$  are negative.
7.  $Z_1$  and  $Z_2$  are positive.
8. 128 items (or half the total items) will be found between the limits of  $X_1$  and  $X_2$ .
9. 128 items (or half the total items) will be found between the limits of  $Y_1$  and  $Y_2$ .



10. 64 items (or one-fourth the total items) greater than  $Z_2$  will be found.
11. 16 items (or one-sixteenth of the total items) will prove to be hostile.
12. Only a single item will meet the logical product of all the criteria.
13. Distribution of items is shown in Figure 3.

3. USQ-20 COMPUTER SOLUTION

The USQ-20 computer method of solution uses only the USQ-20 computer without benefit of the search memory and serves as a basis for comparison of solution times with those of forthcoming solution methods.

A USQ-20 instruction-oriented flow diagram was drawn (Figure 4) to solve the typical search problem stated in Item 2, above. Note that a step of the over-all search task, that of updating dynamic data, is not included in the

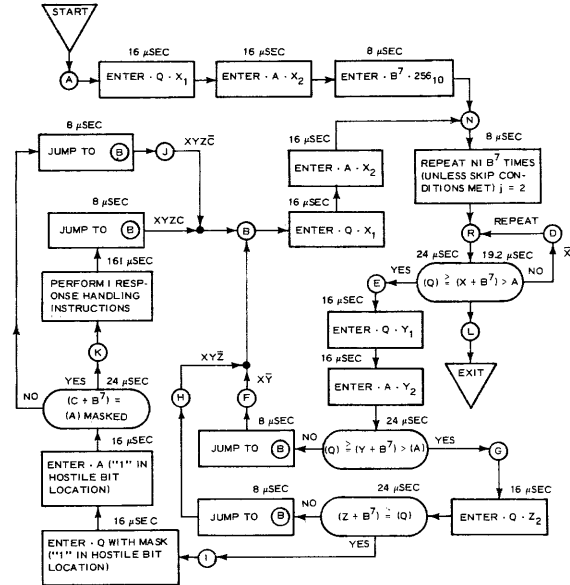


Figure 4. USQ-20 Computer Search Flow

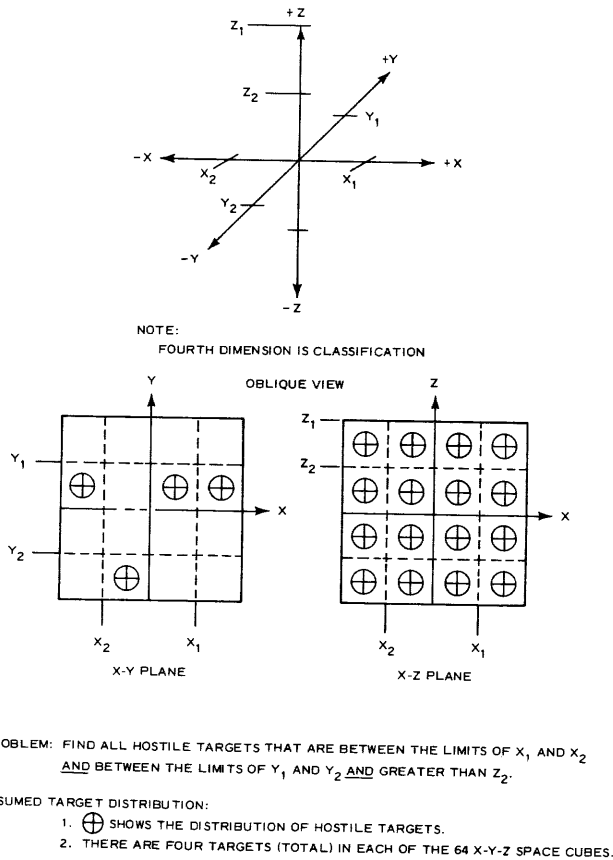


Figure 3. Test Problem Geometry

flow. When only the USQ-20 computer is used, updating is not considered part of the search task. Only when an external search memory is used, which must be updated in addition to the conventional core memory, must updating time be considered. A typical distribution of the items in the four-dimensional space was assumed as shown in Figure 3. From this assumed distribution, the number of cycles through all the loops in the flow diagram can be determined and, therefore, a very close estimate of solution time is possible. The expression for the solution time as a function of the number of items ( $N_i$ ) and the number of typical mixed searches ( $N_s$ ) is handled by four possible equations:

$$T_{c01} = [0.048 + 0.0866(N_i - m) + 0.240m] N_s + 0.016ImN_s, \quad (1)$$

$$T_{c02} = [0.048 + 0.0192(N_i - m) + 0.240m] N_s + 0.016ImN_s, \quad (2)$$

$$T_{c03} = [0.48 + 0.224(N_i - m) + 0.240m] N_s + 0.016ImN_s, \quad (3)$$

and

$$T_{c04} = T_{c02} + \frac{1}{2} (T_{c03} - T_{c02}) = [0.048 + 0.1216(N_i - m) + 0.240m] N_s + 0.016ImN_s. \quad (4)$$

With reference to the above equations,

1. T is search solution time in milliseconds.
2. The subscript  $c_0$  refers to computer (USQ-20) only.
3.  $N_I$  is the number of items.
4.  $m$  is the average number of responses per typical mixed search.
5. Equation 1 assumes the typical (see Figure 3) distribution.
6. Equation 2 assumes best item distribution for the indicated solution sequence.
7. Equation 3 assumes worst item distribution for the indicated solution sequence.
8. Equation 4 is the mean item distribution (between worst and best).

Equation 4 is plotted as Curve 1 in Figures 11 through 13. The term  $0.016ImN_s$  appears in all four equations and expresses the time required to perform some system function based on the responses.

1. 0.016 (msec) is the time per average instruction.
2. I is the number of average instructions per response.
3.  $m$  is the average number of responses per typical mixed search.
4.  $N_s$  is the number of typical mixed searches.

The term  $0.016ImN_s$ , for reasons that are explained in Section V of this paper, is disregarded when the equation is plotted.

#### 4. PERIPHERAL SEARCH MEMORY SOLUTION

##### a. Dynamic Case

As explained earlier, the peripheral search memory consists of the search memory model that interfaces with the USQ-20 computer strictly by means of the standard I/O channels of the USQ-20 computer. The typical problem will be programmed using this system configuration.

The solution for the dynamic case requires periodic search memory updating since the parameters describing each item are assumed to be variable with time. The single parameter table block-transfer method of updating will be used since this appears to be the fastest

method of updating the search memory. An equation describing search problem solution time will be developed by actually writing an abbreviated symbolic program for each of the three basic steps of the overall search problem: (1) updating, (2) supplying criteria, and (3) handling responses. Since instruction execution times are known quantities, one or more terms can be developed for each of these three basic steps of the search problem. A collection of terms results in the required equation.

The generalized functional flow diagram for this configuration is given in Figure 5. The broken line blocks are not considered part of the search problem, while the solid blocks contain the search functions that are executed from the main program. The remainder of the search functions are handled by interrupt routines that begin with the updating routine shown in Figure 6. Internal interrupt routines handle the updating, while the search routines are handled by external interrupt routines.

Updating time is determined by adding the time to perform the updating functions of Blocks F, G, H, and J in Figure 5 to the total instruction execution time for performing all the instructions found in Figure 6. In addition, every word that is transferred into or out of the USQ-20 computer via a standard I/O channel requires  $16\text{-}\mu\text{sec}$  memory-access time that otherwise could be used for arithmetic access time. The total updating time is shown in Table II.

Figure 7A displays the instruction list for supplying criteria to the search memory. The total instruction execution time required to supply criteria to the search memory is  $1176\ \mu\text{sec}$ .

TABLE II—UPDATING TIME, PERIPHERAL SEARCH MEMORY

Function	Time (msec)
Blocks F, G, H, and J (Figure 5)	0.080
Updating routine (Figure 6)	0.684
I/O output access ( $0.016 \times 4 \times N_I$ )	$0.064N_I$
Total	$0.764 + 0.064N_I$

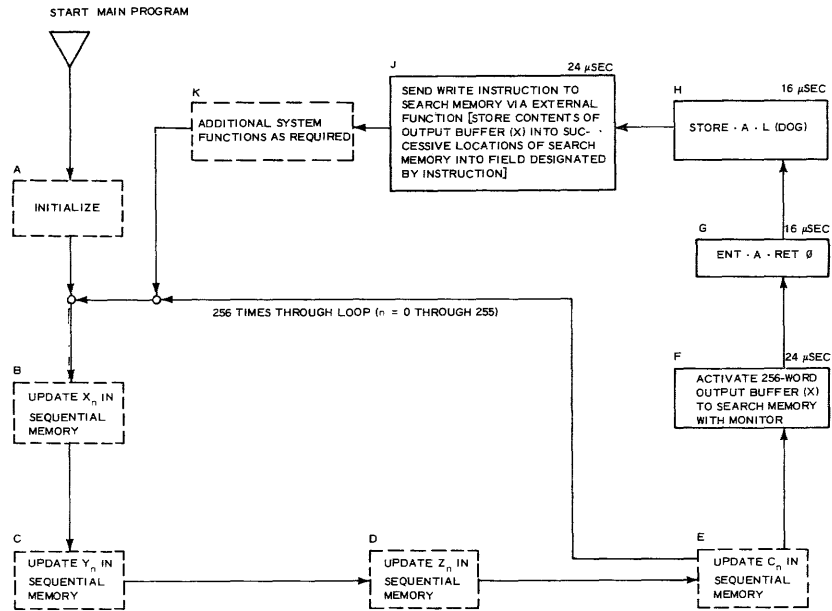


Figure 5. Peripheral Search Memory Search Flow

This is, of course, for a single search. The total execution time for  $N_s$  searches per updating therefore becomes  $1.176N_s$  (in milliseconds). Here again, every word that is transferred in or out of the USQ-20 by a standard I/O channel requires  $16\text{-}\mu\text{sec}$  memory-access time that otherwise could be used for arithmetic access

time. Since 12 criteria words are supplied for each search, the total memory access time to supply criteria to the search memory is:

$$16 \mu\text{sec}/\text{word} \times 12 \text{ words}/\text{search} \times N_s \text{ searches}$$

OR

$$\underline{\underline{0.192N_s \text{ (in milliseconds)}}}$$

TITLE UPDATING ROUTINE		PERIPHERAL SEARCH MEMORY		PROGRAMMER	
PAGE	of	CODING FORM		PLT.	EXT.
				DATE	
LABEL	OPERATOR	OPERANDS AND	INSTR EXEC TIME (μSEC)	EXEC PER SEARCH	TOTAL (μSEC)
UPDATING	HEADER TYPE	*			
INTINT	◆ RETURN JUMP* PAT		24	5	120
PAT	◆ JUMP * P + 1		8	4	32
	◆ ENTER * A · U (120 + j)		16	4	64
	◆ SUBTRACT * A · X + 255 <sub>10</sub> · SKIP · A ZERO		20	4	80
	◆ ENTER * A · U (120 + j) · SKIP		24	3	72
	◆ JUMP * RAT		8	1	8
	◆ SUBTRACT * A · Y + 255 <sub>10</sub> · SKIP · A ZERO		20	3	60
	◆ ENTER * A · U (120 + j) · SKIP		24	2	48
	◆ JUMP * TAR		8	1	8
	◆ SUBTRACT * A · Z + 255 <sub>10</sub> · SKIP · A ZERO		20	2	40
	◆ JUMP * CAN		8	1	8
	◆ JUMP * TAP		8	1	8
CAN	◆ ENTER * A · L (PAT)		16	1	16
	◆ STORE * A · L (ABLE) [FIGURE 7A]		16	1	16
	◆ JUMP * SEARCH [FIGURE 7A]		8	1	8
RAT	◆ OUTPUT * C <sup>n</sup> · MONITOR [256-WORD Y-BUFFER]		24	1	24
	◆ JUMP * PAT		8	1	8
TAR	◆ OUTPUT * C <sup>n</sup> · MONITOR [256-WORD Z-BUFFER]		24	1	24
	◆ JUMP * PAT		8	1	8
TAP	◆ OUTPUT * C <sup>n</sup> · MONITOR [256-WORD C-BUFFER]		24	1	24
	◆ JUMP * PAT		8	1	8
TOTAL					684

Figure 6. Peripheral Search Memory Updating Routine

The total computer real-time consumption necessary to supply criteria to the search memory therefore is

$$1.176N_s + 0.192N_s \text{ (in milliseconds).}$$

Figure 7B shows the instruction list for handling responses from the search memory. The total instruction execution time required to handle responses from the search memory is  $216 + 16I \mu\text{sec}$ . This is for a single search and a single response. The total execution

time for  $N_s$  searches per updating and  $m$  responses per search becomes:

$$0.216 \left[ \frac{m+1}{2} \right] N_s + 0.016ImN_s \text{ (in milliseconds),}$$

where

$$\left[ \frac{m+1}{2} \right] = \text{the greatest integer in } \frac{m+1}{2},$$

$m$  = number of responses per search, and

TITLE SEARCH ROUTINE PERIPHERAL SEARCH MEMORY PROGRAMMER  
PAGE \_\_\_\_\_ of \_\_\_\_\_ PLT. \_\_\_\_\_ EXT. \_\_\_\_\_ MS \_\_\_\_\_  
DATE \_\_\_\_\_

**CODING FORM**

LABEL	OPERATOR	OPERANDS AND NOTES	INSTR EXEC TIME (#SEC)	EXEC PER SEARCH	TOTAL (#SEC)
<b>SUPPLYING CRITERIA</b>					
EXTINT	◆ RETURN JUMP* ABLE		24	6	144
ABLE	◆ JUMP	* P + 1 [MAIN PROGRAM]	8	6	48
SEARCH	◆ ENTER	* A * RET $\phi$	16	5	80
	◆ SUBTRACT	* A * L (DOG) * SKIP * A NOT	24	5	120
	◆ JUMP	* DOG	8	1	8
	◆ STORE	* C <sup>n</sup> * CAT	16	5	80
	◆ ENTER	* A * W (CAT) * SKIP * A NOT	24	5	120
	◆ JUMP	* TARE	8	0	0
DOG	JUMP	RETn [+1]	8	6	48
	◆ JUMP	* ABLE	8	6	48
RET $\phi$	◆ EXT FUNCTION* C <sup>n</sup>	[STOP WRITE]	24	1	24
	◆ OUT	* C <sup>n</sup> * CRIT 1X <sub>1</sub> [2 WORDS]	24	1	24
	◆ EXT FUNCTION* C <sup>n</sup>	[SEARCH - NO REPLY]	24	1	24
RET 1	◆ RETURN JUMP* DOG		24	1	24
	◆ OUT	* C <sup>n</sup> * CRIT 1X <sub>2</sub> [2 WORDS]	24	1	24
	◆ EXT FUNCTION* C <sup>n</sup>	[SEARCH - NO REPLY]	24	1	24
RET 2	◆ RETURN JUMP* DOG		24	1	24
	OUT	* C <sup>n</sup> * CRIT 1Y <sub>1</sub> [2 WORDS]	24	1	24
	◆ EXT FUNCTION* C <sup>n</sup>	[SEARCH - NO REPLY]	24	1	24
RET 3	◆ RETURN JUMP* DOG		24	1	24
	◆ OUT	* C <sup>n</sup> * CRIT 1Y <sub>2</sub> [2 WORDS]	24	1	24
	◆ EXT FUNCTION* C <sup>n</sup>	[SEARCH - NO REPLY]	24	1	24
RET 4	◆ RETURN JUMP* DOG		24	1	24
	◆ OUT	* C <sup>n</sup> * CRIT 1Z <sub>2</sub> [2 WORDS]	24	1	24
	◆ EXT FUNCTION* C <sup>n</sup>	[SEARCH - NO REPLY]	24	1	24
RET 5	◆ RETURN JUMP* DOG		24	1	24
	◆ OUT	* C <sup>n</sup> * CRIT 1C [2 WORDS]	24	1	24
	◆ IN	* C <sup>n</sup> * RESPONSES [128 WORDS]	24	1	24
	◆ EXT FUNCTION* C <sup>n</sup>	[SEARCH - REQUEST REPLY]	24	1	24
RET 6	◆ RETURN JUMP* DOG		24	1	24
CONTINUED IN FIGURE 7B, HANDLING RESPONSES PORTION					
TOTAL EXECUTION TIME FOR SUPPLYING CRITERIA					1176

LABEL	OPERATOR	OPERANDS AND NOTES	INSTR EXEC TIME (#SEC)	EXEC PER SEARCH	TOTAL (#SEC)
<b>HANDLING RESPONSES</b>					
BEG	◆ ENTER	* Q * L (BUFFER COUNT)	16	2	32
	◆ COMPARE	* Q * L (BUFFER LIMIT) * SKIP			
	◆	* NI IF $\frac{1}{2}$ Q	16	2	32
	◆ JUMP	* RET 7	8	1	8
	◆ STORE Q	* L (DOT)	16	1	16
DOT	◆ ENTER	* A * U ( ) * SKIP * A ZERO	24	1	24
	◆ ENTER	* B <sup>n</sup> * A * SKIP	16	0	0
	◆ JUMP	* ODE	8	1	8
PERFORM 1 RESPONSE HANDLING INSTRUCTIONS					
ODE	◆ STORE	* Q * L (LOW)	16	1	16
LOW	◆ ENTER	* B <sup>n</sup> * L ( )	24	1	24
	◆	PERFORM 1 RESPONSE HANDLING INSTRUCTIONS	16	1	161
	◆ REPLACE	* L (BUFFER COUNT) - 1	24	1	24
	◆ JUMP	* BEG	8	1	8
RET 7	◆ RETURN JUMP* DOG		24	1	24
TOTAL EXECUTION TIME FOR HANDLING RESPONSES					216 + 16I

Figure 7. Peripheral Search Memory Search Routine

$I$  = number of instructions per response to perform some system function based on the response.

Again, every word that is transferred in or out of the USQ-20 by a standard I/O channel requires 16  $\mu$ sec memory-access time that otherwise could be used for arithmetic access time. Therefore, the total memory access-time-to-transfer responses from the search memory is

$$16 \mu\text{sec per response word} \times \left[ \frac{m+1}{2} \right] \\ \text{response words per search} \times N_s \text{ searches}$$

or

$$0.016 \left[ \frac{m+1}{2} \right] N_s \text{ (in milliseconds)}$$

where

$$\left[ \frac{m+1}{2} \right] = \text{the greatest integer in } \frac{m+1}{2} \text{ and} \\ m = \text{number of responses per search.}$$

The total real-time consumption necessary to handle responses from the search memory therefore is

$$\frac{0.216 \left[ \frac{m+1}{2} \right] N_s + 0.016ImN_s}{+ 0.016 \left[ \frac{m+1}{2} \right] N_s}$$

The final equation that expresses the complete search solution time for this system configuration can be found by combining the double underlined expressions that describe each of the three basic search steps:

$$T_{\text{PSD}} = 0.764 + 0.064N_i + 1.368N_s + 0.232 \\ \left[ \frac{m+1}{2} \right] N_s + 0.016ImN_s, \quad (5)$$

where

$T$  = search solution time expressed in milliseconds.

PSD (subscript) = system configuration (peripheral search memory, dynamic),

$N_i$  = number of items,

$N_s$  = number of typical mixed searches per search memory updating,

$$\left[ \frac{m+1}{2} \right] = \text{greatest integer in } \frac{m+1}{2},$$

$m$  = number of responses per search,

$I$  = undetermined (depends upon application) number of average (16  $\mu$ sec) instructions required to perform some system function that is based on each response address.

Equation 5 is plotted as Curve 2D in Figures 11 through 13. However, the last term ( $0.016ImN_s$ ) of the equation is disregarded when plotted for reasons that are explained in Section V of this paper.

#### b. Static Case

In some applications of the search memory, it is recognized that the search memory data could be static. Under this assumption, the equation for search solution time is similar to Equation 5 with the updating terms removed. The equation then becomes

$$T_{\text{PSS}} = 1.368N_s + 0.232 \left[ \frac{m+1}{2} \right] N_s \\ + 0.016ImN_s, \quad (6)$$

where

PSS (subscript) = system configuration (peripheral search memory, static).

The first two terms of this equation are plotted as Curve 2S in Figures 11 through 13.

## 5. INTEGRATED SEARCH MEMORY

### a. Dynamic Case

An integrated search memory has been proposed and is described in Section III above. The typical problem will be programmed using this system configuration.

The solution for the dynamic case requires periodic search-memory updating since the parameters describing each item are assumed to be variable with time. The single parameter table block-transfer method of updating will be used since this appears to be the fastest method of updating the search memory and

also since this is the same method used to update the peripheral search memory above.

An equation describing search problem solution time will be developed by actually writing an abbreviated symbolic program for each of the three basic steps of the over-all search problem: (1) updating, (2) supplying criteria, and (3) handling responses. The generalized functional flow diagram for this configuration is given in Figure 8. The broken line blocks are not considered part of the search problem; the solid blocks contain the search functions that are executed from the main program. The remainder of the search functions are handled by internal interrupt routines that begin with the updating routine shown in Figure 9.

Updating time is determined by adding the time to perform the updating functions of Blocks F and G in Figure 8 to the total instruction execution time required to perform all the instructions found in Figure 9.

In addition, every word that is transferred into or out of the USQ-20 computer via a standard I/O channel requires 16- $\mu$ sec memory-access time that otherwise could be used for arithmetic access time. The total updating time is shown in Table III.

The instructions necessary to perform the remaining portions of the over-all search problem, those of supplying criteria to the search memory and handling responses from the search memory, are found in Figure 10. The

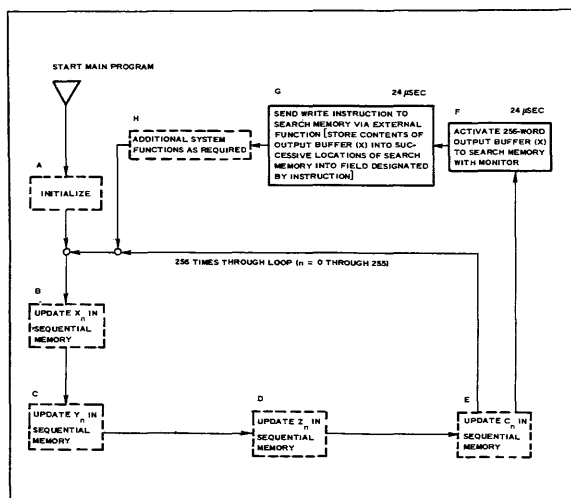


Figure 8. Integrated Search Memory Search Flow

TABLE III—UPDATING TIME, INTEGRATED SEARCH MEMORY

Function	Time (msec)
Blocks F and G (Figure 8)	0.048
Updating routine (Figure 9)	0.708
I/O output access ( $0.016 \times 4 \times N_I$ )	<u><math>0.064N_I</math></u>
Total	<u><u><math>0.756 + 0.064N_I</math></u></u>

total time is  $(481 + 24m + 16Im)$  microseconds. This, of course, is for a single search. The total execution time for  $N_s$  searches per updating, therefore, becomes:

$$0.481N_s + 0.024mN_s + 0.016ImN_s \text{ (in milliseconds),}$$

where

- m = number of responses per search, and
- I = number of instructions to perform some system function based on each response.

The final equation that expresses the complete search solution time for this system configuration can be found by combining the foregoing double underlined expressions. The equation becomes

$$T_{ISD} = 0.756 + 0.064N_I + 0.481N_s + 0.024mN_s + 0.016ImN_s, \quad (7)$$

where

T = search solution time in milliseconds,

ISD (subscript) = system configuration (integrated search memory, dynamic),

$N_I$  = number of items,

$N_s$  = number of typical mixed searches per search memory updating,

m = number of responses per search, and

I = undetermined (depends upon application) number of average (16  $\mu$ sec) instructions required to perform some system function that is based on each response address.

TITLE <u>UPDATING ROUTINE</u>		INTEGRATED SEARCH MEMORY		PROGRAMMER		
PAGE _____ of _____		CODING FORM		PLT. _____ EXT. _____ MS _____	DATE _____	
LABEL	OPERATOR	OPERANDS AND	NOTES	INSTR EXEC TIME (USEC)	EXEC PER SEARCH	TOTAL (USEC)
OUTINT	RETURN JUMP	PAT [BAT - FIGURE 10 - AFTER PER-				
		FORMING CAT]		24	5	120
PAT	JUMP	P + 1		8	4	32
	ENTER	A * U (120 + j)		16	4	64
	SUBTRACT	A * X + 255 <sub>10</sub> * SKIP * A ZERO		20	4	80
	ENTER	A * U (120 + j) * SKIP		24	3	72
	JUMP	RAT		8	1	8
	SUBTRACT	A * Y + 255 <sub>10</sub> * SKIP * A ZERO		20	3	60
	ENTER	A * U (120 + j) * SKIP		24	2	48
	JUMP	TAR		8	1	8
	SUBTRACT	A * Z + 255 <sub>10</sub> * SKIP * A ZERO		20	2	40
	JUMP	CAN		8	1	8
	JUMP	TAP		8	1	8
CAN	OUTPUT	C <sup>n</sup> * MONITOR [j WORD]		24	1	24
	ENTER	A * BAT		16	1	16
CAT	STORE	A * L (TAB)		16	1	16
	JUMP	PRESEARCH [FIGURE 10]		8	1	8
RAT	OUTPUT	C <sup>n</sup> * MONITOR [256-WORD Y-BUFFER]		24	1	24
	JUMP	PAT		8	1	8
TAR	OUTPUT	C <sup>n</sup> * MONITOR [256-WORD Z-BUFFER]		24	1	24
	JUMP	PAT		8	1	8
TAP	OUTPUT	C <sup>n</sup> * MONITOR [256-WORD C-BUFFER]		24	1	24
	JUMP	PAT		8	1	8
TOTAL						708

Figure 9. Integrated Search Memory Updating Routine

Equation 7 is plotted as Curve 3D in Figures 11 through 13. However, the last term ( $0.016ImN_s$ ) is disregarded when plotting for reasons that are explained in Section V.

#### b. Static Case

In some applications of the search memory, it is recognized that the search memory data can be static. Under this assumption, the equation for search solution time is similar to Equation 7 with the updating terms removed. The equation then becomes

$$T_{ISS} = 0.481N_s + 0.024mN_s + 0.016ImN_s, \quad (8)$$

where

ISS (subscript) = system configuration (integrated search memory, static).

The first two terms of Equation 7 are plotted as Curve 3S in Figures 11 through 13.

## SECTION V—SEARCH TIME COMPARISONS AND CONCLUSIONS

A search time analysis was performed in Section IV for each of three search system configurations. For each system configuration several equations were derived. The dependent

variable in all equations is search time,  $T$ . All equations contain the independent variable  $N_s$ , which is the number of searches per search memory updating, and  $m$ , which is the average number of responses per search. Most equations contain a third independent variable,  $N_i$ , which is the number of items carried in the system.  $T$  versus  $N_s$  is plotted for each system configuration on each of the three graphs (Figures 11, 12, and 13). Figure 11 treats  $N_i$  as a constant 256 items; Figure 12 treats  $N_i$  as a constant 512 items; Figure 13 treats  $N_i$  as a constant 1024 items. In all cases,  $m$  is assumed equal to one. The search time equations that describe all the curves are derived in Section IV. Note that one term,  $0.016ImN_s$ , is common to all the equations for all system configurations. This term describes the time required to perform some system function, based on the search response address and must be performed in the USQ-20 computer in all cases. This function is not considered part of the search but only related to the search in that it is based on the results of the search. This term is, therefore, disregarded in the plots of the equations.

Equations 1, 2, 3, and 4 describe the search time when only the USQ-20 computer is used.

TITLE SEARCH ROUTINE		INTEGRATED SEARCH MEMORY		PROGRAMMER		
PAGE _____ of _____		CODING FORM		PLT. _____	EXT. _____ MS _____	
		DATE _____				
LABEL	OPERATOR	OPERANDS AND	NOTES	INSTR EXEC TIME (#SEC)	EXEC PER SEARCH	TOTAL (#SEC)
	HEADER TYPE					
PRESEARCH	ENTER	B <sup>n</sup> • ZERO		8	1	8
SEARCH	ENTER	Q • W (X <sub>1</sub> MASK + B <sup>n</sup> )		16	1	16
	ENTER	A • W (X <sub>1</sub> + B <sup>n</sup> )		16	1	16
	EXT FUNCTION	C <sup>n</sup> [SEARCH - NO REPLY]		24	1	24
	REPEAT NI	• 10000 <sub>8</sub> [B <sup>7</sup> TIMES]		8	1	8
	ENTER	B <sup>0</sup> • ZERO [NO OP]		6.5 (AVE)	2	13
	* ON COMPLETION OF SEARCH, SEARCH MEMORY CLEARS B <sup>7</sup> IN USO-20					
	ENTER	Q • W (X <sub>2</sub> MASK + B <sup>n</sup> )		16	1	16
	ENTER	A • W (X <sub>2</sub> + B <sup>n</sup> )		16	1	16
	EXT FUNCTION	C <sup>n</sup> [SEARCH - NO REPLY]		24	1	24
	REPEAT NI	• 10000 <sub>8</sub> [B <sup>7</sup> TIMES]		8	1	8
	ENTER	B <sup>0</sup> • ZERO [NO OP]		6.5 (AVE)	2	13
	* ON COMPLETION OF SEARCH, SEARCH MEMORY CLEARS B <sup>7</sup> IN USO-20					
	ENTER	Q • W (Y <sub>1</sub> MASK + B <sup>n</sup> )		16	1	16
	ENTER	A • W (Y <sub>1</sub> + B <sup>n</sup> )		16	1	16
	EXT FUNCTION	C <sup>n</sup> [SEARCH - NO REPLY]		24	1	24
	REPEAT NI	• 10000 <sub>8</sub> [B <sup>7</sup> TIMES]		8	1	8
	ENTER	B <sup>0</sup> • ZERO [NO OP]		6.5 (AVE)	2	13
	* ON COMPLETION OF SEARCH, SEARCH MEMORY CLEARS B <sup>7</sup> IN USO-20					
	ENTER	Q • W (Y <sub>2</sub> MASK + B <sup>n</sup> )		16	1	16
	ENTER	A • W (Y <sub>2</sub> + B <sup>n</sup> )		16	1	16
	EXT FUNCTION	C <sup>n</sup> [SEARCH - NO REPLY]		24	1	24
	REPEAT NI	• 10000 <sub>8</sub> [B <sup>7</sup> TIMES]		8	1	8
	ENTER	B <sup>0</sup> • ZERO [NO OP]		6.5 (AVE)	2	13
	* ON COMPLETION OF SEARCH, SEARCH MEMORY CLEARS B <sup>7</sup> IN USO-20					
	ENTER	Q • W (Z <sub>2</sub> MASK + B <sup>n</sup> )		16	1	16
	ENTER	A • W (Z <sub>2</sub> + B <sup>n</sup> )		16	1	16
	EXT FUNCTION	C <sup>n</sup> [SEARCH - NO REPLY]		24	1	24
	REPEAT NI	• 10000 <sub>8</sub> B <sup>7</sup> TIMES		8	1	8
	ENTER	B <sup>0</sup> • ZERO [NO OP]		6.5 (AVE)	2	13
	* ON COMPLETION OF SEARCH, SEARCH MEMORY CLEARS B <sup>7</sup> IN USO-20					
	ENTER	Q • W (C MASK + B <sup>n</sup> )		16	1	16
	ENTER	A • W (C + B <sup>n</sup> )		16	1	16
	EXT FUNCTION	C <sup>n</sup> [SEARCH - REQUEST REPLY]		24	1	24
REP 6	REPEAT NI	• 10000 <sub>8</sub> B <sup>7</sup> TIMES		8	1 + m	8 + 8m
	ENTER	B <sup>0</sup> • ZERO [NO OP]		8	m	8m
	* ENTER RESPONSE ADDRESS FROM SEARCH MEMORY INTO B <sup>1</sup> REGISTER AND CLEAR B <sup>7</sup>					
	REGISTER					
	PERFORM (I)	RESPONSE HANDLING INSTRUCTIONS		16	1m	161m
	JUMP	• REP 6		8	m	8m
BAT	JUMP	• P + 1		8	0	0
	BSKIP	• B <sup>n</sup> • SEARCH COUNT		16	1	16
	JUMP	• SEARCH [FIGURE 8]		8	0	0
	JUMP	• PAT   FIGURE 22A		8	1	8
TOTAL EXECUTION TIME FOR PERFORMING SEARCH				481 + 24m + 161m		
* THESE ARE NOT USO-20 INSTRUCTIONS; THEY ARE SYSTEM FUNCTION STATEMENTS.						

Figure 10. Integrated Search Memory Search Routine



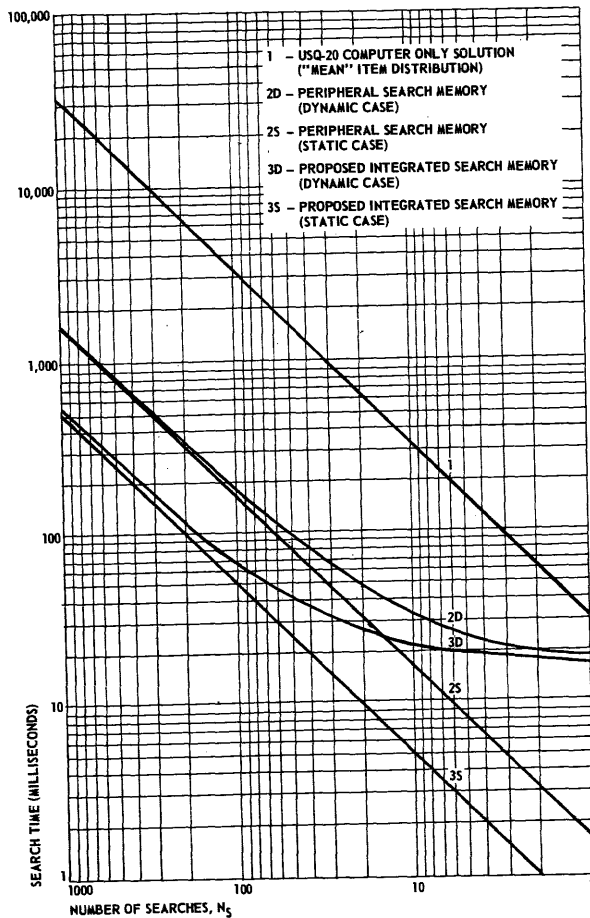


Figure 11. Search Time (T) versus Number of Searches per Updating ( $N_s$ ), 256-Item Case,  $m = 1$

Since the USQ-20 computer must perform the search sequentially, the resulting search time is highly sensitive to item distribution (within the space constraints) for any given solution sequence. Equation 1 assumes the typical distribution found in Figure 3. Equation 2 assumes best case distribution for the solution sequence used; Equation 3 assumes worst case distribution for the solution sequence used; and Equation 4 is the calculated mean between Equation 2 and 3. Since the item distribution changes in a dynamic situation, the mean distribution is the most meaningful. Therefore, Equation 4 was chosen to be plotted (as Curve 1 in Figures 11 through 13) to display the problem solution resulting from use of only the USQ-20 computer. Updating is not included in the search time, using the USQ-20 computer only, since it is assumed that updating must occur to meet other system requirements.

Curves 2 and 3 display the problem solution time of the peripheral search memory and the proposed integrated search memory, respectively. Curves 2D and 3D handle the dynamic problem situation where the search memory must be updated in addition to the conventional memory. Curves 2S and 3S represent a static problem situation where updating is not required. The curves displayed in Figures 11 through 13 allow unlimited performance comparisons to be made between the three system configurations. The conditions, however, must be specifically stated. Two representative comparisons will be made here (see Figure 12). The conditions are as follows:

$$N_I = 512$$

$$m = 1$$

$$N_s = 1000.$$

Results are

- Peripheral search memory (Curve 2D) provides a performance increase by a fac-

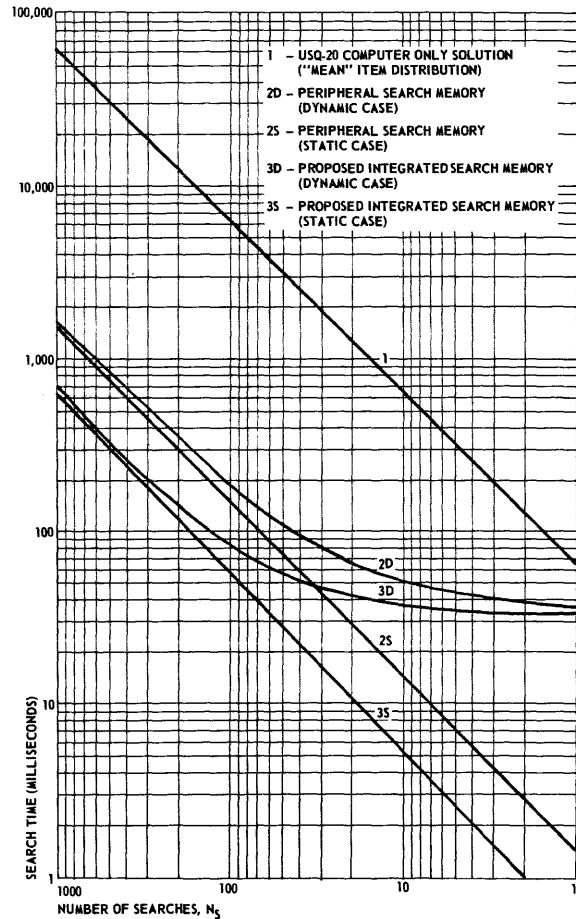


Figure 12. Search Time (T) versus Number of Searches per Updating ( $N_s$ ), 512-Item Case,  $m = 1$

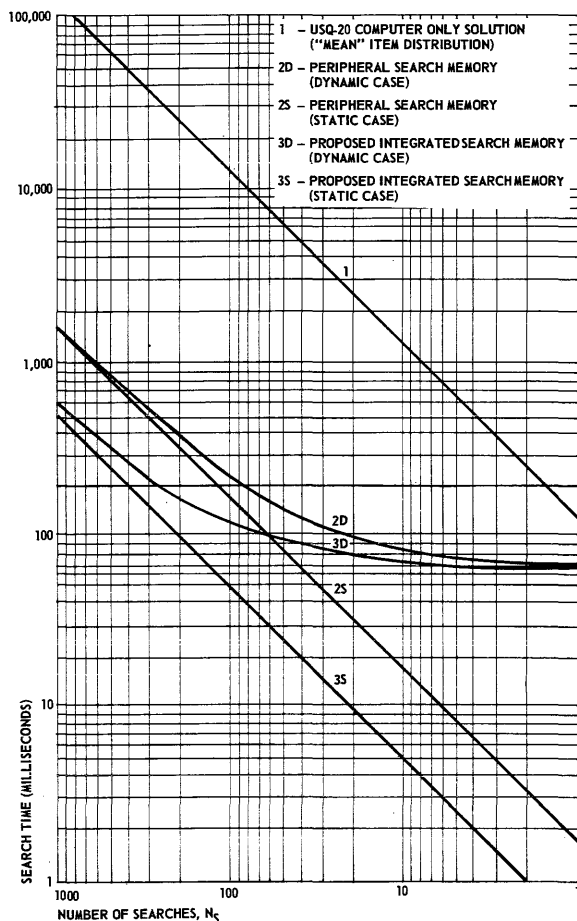


Figure 13. Search Time ( $T$ ) versus Number of Searches per Updating ( $N_s$ ), 1024-Item Case,  $m = 1$

tor of 38 over the conventional computer only (Curve 1) solution.

2. Integrated search memory (Curve 3D) provides a performance increase by a factor of 116 over the conventional computer only (Curve 1) solution.

Note that the curves show only system search time of the three system configurations. The proposed integrated search memory has an additional advantage over the peripheral search memory—less complex programs. Comparison of the list of instructions for the two system configurations (given in Figures 7 and 10) will indicate that the proposed integrated search memory approach requires fewer instructions and allows more familiar and direct programming techniques than for the peripheral search memory.

An important point to remember is the fact that the updating terms of the search time equations, for the system configurations that involve the search memory, only need be considered if the search memory must be updated in addition to the conventional core memory of the USQ-20 computer. If only one of the two memories requires updating then the updating terms may be ignored and, therefore, only the straight line plots of Figures 11 through 13 need be considered.

Another consideration is the fact that the feasibility of transferring large amounts of data over the standard USQ-20 I/O channels for one purpose (search memory in this case) is unpredictable unless the word rate requirements of all the I/O channels are known and considered. In a complex system, there is the possibility that the I/O is already heavily loaded and that the peripheral search memory I/O requirements could result in overload of the I/O. The proposed integrated search memory, on the other hand, does not have to depend on standard I/O channel data transfer, and so this unpredictable I/O situation can be avoided. Related to this consideration is the definition of solution time. The solution times analyzed and displayed in this paper are based on computer memory access time. In an approach where standard I/O channel transfers are not involved, computer memory access time is identical with real or actual solution time. Where standard I/O transfer time is involved, memory access times are not adjacent and the actual solution time is greater than the solution time based on computer memory access time alone. This then points to another advantage of the proposed integrated search memory approach.

#### LIST OF REFERENCES

1. GER-10857: *Collection of Technical Notes on Associative Memory*. Akron, Ohio, Goodyear Aerospace Corporation, 9 October 1962.
2. HORVATH, R.: *Integrating the Search Memory with the USQ-20 Computer*. GER-11621. Akron, Ohio, Goodyear Aerospace Corporation, 4 June 1963.

# A BIT-ACCESS COMPUTER IN A COMMUNICATION SYSTEM

*Dr. Edmund U. Cohler and Dr. Harvey Rubinstein*  
*Sylvania Electronic Systems*  
*A Division of Sylvania Electric Products, Inc.*  
**APPLIED RESEARCH LABORATORY**  
*40 Sylvan Road*  
*Waltham, Massachusetts, 02154*

## 1.0 INTRODUCTION

Systems having computers and communications subsystems are increasing in number. The application of such systems span such diverse fields as process control, message switching, command and control, and multi-user on-line computer installations. In these systems, a significant portion of the information processed is brought to and sent from the computer on a large number of communication lines, carrying peak bit rates generally from 75 bps to 4800 bps.

Often, failure of a portion of the system to provide services can entail serious consequences to the system users. Thus severe reliability standards are placed on the system hardware. Many of these systems must be capable of providing service to a range in the number of users and must be able to grow as the system finds more users. Thus, one finds the need for modularity to meet these demands. Finally, as these systems are used, they must be capable of change so that they can be adapted to the ever changing and wide variety of requirements, problems, formats, codes and other characteristics of their users. As a result, general-purpose stored program computers should be used wherever possible.

Past approaches toward meeting these operating requirements have been made by utilizing

two computers (full redundancy) to obtain the required reliability and availability. One computer stood by while the other processed data on line. When it failed, the computers were interchanged. To handle the incoming data, many of these past systems were designed with costly complex fixed programmed bit and character buffers, and message assemblers. The buffers operated in such a way that a failure in one of them could prevent usage of a number of transmission lines. As a rule, the fixed programs wired into these units did not permit rapid changes of the characteristics of the line it handled.

In this report, a design for a low-cost multi-processor system is described which alleviates these past deficiencies. This system performs the store-and-forward operations of a message switch. A unique design of the input and output interface is central to meeting these objectives, and is the primary topic of this paper.

### 1.1 *Operational Objectives of the Design*

The basic objectives of the work described were to design a message switch which provides:

- A. Improved operational reliability,
- B. Greater economy, both in the initial installation and in operation, and

- C. Greater adaptability to a wide variety of communication environments and procedures, initially and during operation.

These goals were achieved through the design features which are summarized below:

1. The design is made modular through the use of equipment pools. These pools of processing and storage equipment lead to a very high order of reliability with low initial and maintenance costs. In particular, a method has been evolved for using a number of small digital computers to provide message switching functions with a large amount of flexibility and modularity in operation and in installation.
2. A unique method of interfacing lines with processors has been invented which decreases buffering costs, failure interaction, and bit processing equipment requirements. The method is a combination of hardware, logic and software which ideally suits the problem at hand. It makes possible to use general-purpose processors in transferring and processing messages and thus provides great adaptability to changing environments and requirements. The direct access to memory of information lines provided by this technique allows much greater equipment efficiency in handling incoming and outgoing information in a class of multiuser computer systems extending well beyond message processing. The message processing example, however, shows sufficient details to support the claims of better efficiency.
3. An efficient method for the orderly storage and retrieval of messages in a modular drum (or other medium-access-time) storage system has been evolved. This method reduces the cost of core storage by allowing frequent drum accesses and reduces the cost of the drum system by making efficient use of the storage required by the drum.

### 1.2 *Achieving Operational Reliability*

To obtain the reliability offered by redundancy without the impoverishing costs of du-

plexing, we have turned to the use of modular equipment pools. As an example, the processor pool serves incoming and outgoing lines. Each of these lines has a usable service connection to three separate processors. Thus, if a single processor fails completely, its lines can be serviced by other processors that are not completely occupied. Because there are four processors in the pool, we need only 25 percent overcapacity (redundancy) in each processor to assure no loss of system capacity on a single processor failure. Many computer-centered systems require 100 percent redundancy (complete duplexing to achieve the same result. Other such systems do not use the pool concept so that despite similar degrees of modularity, a loss of a single module causes complete loss of service to a group of lines until it is either repaired or replaced. A similar pooled approach has been taken in the message storage area where any of three selection units can give a processor access to the message storage drum modules.

The modularity of this design also improves maintenance times by reducing the time required to isolate faults and by simplifying the training of maintenance personnel. Shorter times to find a difficulty and correct it result in greater system reliability.

An important requirement in pool operation is that failures on one side of an interface do not cause failures on the other. The magnetically coupled interfaces used in the direct-access-to-memory avoid this difficulty. The magnetic coupling is sufficiently loose that the failure of an active component cannot affect other equipments across the interface.

### 1.3 *Achieving Greater Economy*

We have achieved economy in this system primarily by the invention of a new type of line access to a processor which makes the processor more than five times more efficient in the acceptance and assembly of bits from a serial line. In all past such switching systems either external equipment was assigned to the task of bit assembly or it was done in the computer at great cost in number of memory cycles per bit. By making possible a single instruction time for the handling of a single bit, we have

been able both to eliminate external equipments and to make efficient usage of our computer in handling the bit transfers and assemblies as well as the more complex but less frequent jobs of switching.

Furthermore, the modularity of the processing pool allows us to choose the switch capacity to suit traffic conditions and numbers of lines in various installations thus minimizing the required equipment. In addition, the pooled approach results in much less redundancy so that we can expect an almost two-to-one cost reduction over other duplexed systems even at their optimum capacity.

1.4 Achieving Greater Operation Adaptability

The advantage of using a programmed processor for flexibility and adaptability to meet new requirements and situations over the older techniques of wired-in operations is now well recognized in the industry. Error correction and detection schemes may be implemented. Very sophisticated priority disciplines can be adopted on a moment's notice to suit the situation at hand. Changes in codes, formats and routing indications can be handled. With pooled design, we can re-assign lines not only under failure conditions but under conditions of changing traffic patterns because line assignments are made electronically and each line can be assigned to any of three separate processors.

In talking about the adaptability to change, we should also speak of protection against unwarranted change. We observe that this system, being primarily under programmed control, permits protection from tampering by making initial program entry possible only from protected devices, while subsequent modification through the console or other external devices would be solely under the control of some internal program.

2.0 Description of the System

2.1 Description of Switch Operation

The block diagram of Figure 1 shows the equipment pools and their interconnections. The most important pools in the normal on-line operation of the switch are the processor pool, the message drum pool and the processing

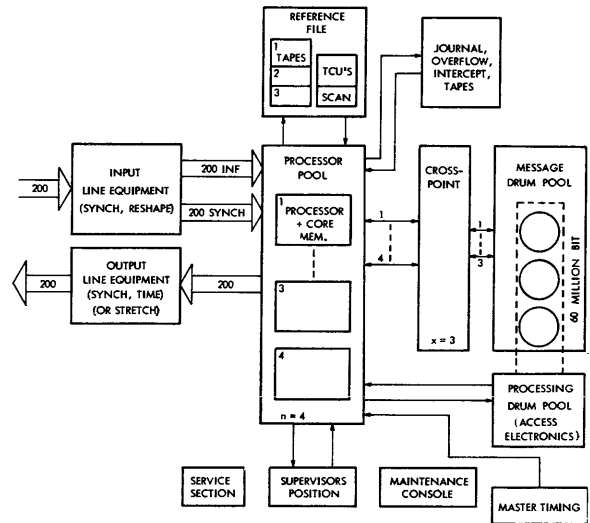


Figure 1. 200-Line Message Switch Major Subsystems Diagram (Maximum Lines).

drum pool. The tape and console pool play a subsidiary job as they are only partially utilized in the routine switch operation. Briefly describing the input processing, the incoming bits for each line are sent to three processors in the processing pool. A supervisory program has previously assigned each line to one of these three processors. As the bits arrive, the processor assembles them into characters and checks the characters for special system coordination and control information. Included in these control information groups are the routing indicators which identify the message destination and precedence characters which indicate the priority of the message. When these arrive, an access is made to the processing drum pool by the processor to translate these groups into outgoing line numbers. When the outgoing line numbers and the precedence of the message are known to the processor and the message has fully arrived, an entry is made into a table (queue list) to alert the outgoing line that a message is awaiting transmission. In addition, the processor enters somewhat different information onto a ledger, which maintains an account of the message status; i.e., those lines on which the message is to be transmitted, those on which it has been transmitted and those which have acknowledged the transmission. Simultaneously, the processor trans-

fers the incoming message onto the 'in-transit' message-drum pool and onto the reference-tape pool. This it does in fixed-size bins. The initial entries into a journal are also made in the course of the input message processing. The journal is a chronological listing of the actions taken on a message while it is in the switch.

In output processing when a processor finds that one of its outgoing lines is no longer busy (or at fixed intervals after a nonbusy condition), it refers to the queue lists, which are identified by line and precedence, to obtain the message-drum address of the next message to be transmitted on that line. It then makes arrangements to retrieve that message from the drum and to send out the characters one bit at a time. In the meanwhile recordings of these actions are made upon the journal tape. When the message is completely transmitted, additional entries are made in the ledger to indicate transmission. When all transmissions of a message have been made the in-transit message and the ledger are erased.

## 2.2 *The Processor Pool Functions*

The processor pool accepts the bits from a line, assembles them into characters, disassembles them and sends them to a line. Secondly, it examines the incoming characters and performs a variety of routing, queuing and surveillance functions based on these. Thirdly, it stores groups of characters in its core memory for buffering other storage pools (primarily the message drum pool). Finally, during slack time and routinely, the processor evaluates switch operation and traffic for maintenance and adaptation purposes. It is evident that a general-purpose processor can handle all of these functions, and if it is time shared, efficient hardware usage can be achieved along with flexible operation.

In Section 3.1, it is demonstrated that bit and character processing dominate the other processes in computer usage thus making the interface techniques important in improving processor efficiency.

Because this paper is primarily on the interface technique which we have used between the communication lines and the processing pool, our discussion will center on this pool.

## 2.3 *Message Drum Pool Functions*

The major message drum pool function is the storage of messages to accommodate line availability/demand variations. This variation shows up as messages stored in "in-transit" storage awaiting lines to become free for transmission. It is clear that this function requires orderly and efficient storage of messages.

Orderly storage of information on the message drum and efficient transfer to and from the processing pool have been achieved by the use of list processing techniques. Because of properly chosen accessing procedures, all bins of information in the processor memory are of the same size, so that the information may be stored on the drum in fixed-size bins and successive bins chained to previous ones. A complete empties list keeps track of all available storage space remaining on the drum, thus permitting very efficient storage filling. Because all messages are stored in fixed-size bins, the problem of cross-office speed conversion is automatically accomplished. A full discussion of the message drum pool techniques is beyond the scope of this paper.

## 2.4 *Processing Drum Pool Function*

The processing drum pool stores the lists and registers used in the processing job by the processor pool. Its lists, as a matter of fact, are used in common by all of the processors of the processing pool to provide a record of: where the message will be kept on the message drum pool, on what lines the message is to be transmitted, in what sequence the message is to be transmitted, to what lines the message has been sent, which message to transmit next, where the message is located; and to update the ledger entry of lines to which the message has been sent. Even though normal operation of this pool is independent of the message drum pool, it makes use of the same drums for storage. This is possible and efficient because both storage capacities are determined by the maximum probable queue build up.

## 2.5 *Other Equipment Groups*

The other equipment groups within the switch are not as central to the normal operation of the switch, and will not be covered in this paper.

3.0 SYSTEM DESIGN

3.1 *The Processor Pool*

3.1.1 *Processing Jobs*

Messages on various lines and trunks may differ in code, bit rate, and message format, but in each case, the message consists of a header, text and ending. The header includes routing and message priority. Some messages are divided into 80 character blocks for transmission and reception purposes.

In a message store and forward system, processing of two types are encountered. The first type centers about the acceptance, storage and transmission of messages and the second type about the control of the switching system.

Tables I, II and III indicate typical message processing functions. Routine functions are classified in these tables as either of a bit, character block or message type.

System control functions keep the switching system performing effectively by supplying the switching center supervisor with data useful in the management of the store and forward center and network, and as an aid in maintaining and testing the switch, its programs, and data base. They are generally not performed

regularly or very frequently and are given in Table IV.

3.1.2 *Discussion of Store and Forward Switch Functions*

An examination of the list of functions given in Tables I-IV shows the store and forward switch functions fall into four classes: data formatting, system operation, signal acceptance and transmission and recording (storage) operations.

In order to obtain an order of the importance of these functions to message switching, it is desirable to classify them in the order of their frequency of occurrence. The acceptance and forwarding of bits are the most frequently occurring functions. They occur at the incoming and outgoing bit rates. The next most frequently occurring functions are those associated with each and every character which enters or leaves the system, for example, control character check. Most functions are not performed on each character. Header validations and entries in queue lists for example are performed on entire messages independent of the number of characters they contain. The character functions occur 1/b times as frequently as the bit function, where b is the number of bits per character and averages almost 7 bits per char-

TABLE I—INPUT PROCESSING—ROUTINE

	Bit	Character	Block	Message
Accept bits and assemble characters	x			
Check parity of characters	x	x		
Detect system control characters		x		
Assemble characters in words and bins		x		
Write messages in "In-Transit Store"				x
Initiate preemption for flash messages				
Verify header				x
Perform routing				x
Enter incoming message data in ledger				x
Check block parity			x	
Write message on reference tape				x
Enter data in journal				x
Acknowledge accepted messages				x
Count blocks			x	
Enter data in queue lists				x
Assign serial number for processing				x

TABLE II—OUTPUT PROCESSING—ROUTINE

	Bit	Character	Block	Message
Use queue lists to initiate message transmission				x
Make journal entries				x
Update ledger				x
Retrieve messages from "In-Transit Store"				x
Convert formats				x
Convert codes		x		
Construct block parity		x		
Check security for each block			x	
Remove messages from storage which have been transmitted				x
Convert messages to a bit stream	x	x		

acter. The next set of operations in the order of descending frequency of occurrence, are those which occur for each block. They occur  $1/cb$  as often as the bit functions. Here,  $c$  is the number of characters per block and is about 80 characters. The remaining functions occur once per message or so. If there are  $m$  characters per message, the message functions occur  $1/mb$  as often as the other functions where  $m$  is approximately 2000. Then each character, block or message function occurs respectively 8, 640, 16,000 times as infrequently as a bit function.

The most frequent functions then are the bit functions. When a bit arrives, it must be stored in the proper place in a character, used to update the character parity count, and counted to establish the arrival of a full character. When a full character is received, it is transferred to another location in memory for further processing. A similar per line process occurs in reverse order when information is disassembled for forwarding. These bit functions are performed on each line at a rate determined by the information rate on the line. In designing an equipment to perform this function, various

TABLE III—MESSAGE PROCESSING—NON-ROUTINE

Control errors.
Print or display messages.
Initiate service messages.
Manually retrieve message.

line rates (from 75 bits per second to 4800 bps), and various bits per character (from 5 to 8 depending on their code) must be considered.

The character operations are somewhat more complex. Typically, characters must be examined to determine if they are system control or coordination characters, and if their parities are correct. When transmitting, the character codes may require conversion and the block parity must be determined. The characters also must be counted to determine block length.

Two characteristics of these bit and character functions should be noted. The first is that there exists a variety of functions and the

TABLE IV—SYSTEM CONTROL PROCESSING TASKS

Maintain in-transit storage status.
Maintain status of traffic.
Activate overdue message alarms.
Activate queue threshold alarms.
Execute maintenance routines.
Control program maintenance routines.
Activate and control start-up.
Activate and control recovery.
Check confidence levels on equipments and links.
Provide line synchronization.
Allocate hardware.
Accept supervisor initiated commands.
Provide statistical analyses of traffic.



second that these same functions are performed on all lines. The latter assumptions imply time sharing of equipment while the former implies a reasonably complex assortment of equipment.

### 3.1.3 *The Processor Interface With Input and Output Lines*

Because the most frequent operations in the message switch are the acceptance and the delivery of bits of information, earlier switch designs used special equipment to accept bits from a line and assemble them into characters for use by the processor. That approach involved considerable equipment which was peculiar to a particular line type. Recent designs have made this equipment sufficiently flexible (generally by pluggable programs) to be suitable for a wide variety of such lines. However, in so doing, any efficiency derived from special-purpose equipment was lost. Furthermore, an efficient method of providing alternate capability (redundancy) in case of equipment failure was not provided. Seeking economy, redundancy, flexibility and simplicity in handling bits, we use a general-purpose machine, taking advantage of its high speed to perform service for a number of lines. A direct and inexpensive interface is made to each line. Each line interfaces with a number of processors so that in case of failure, assignments can be made electronically for other processors to take over the lines formerly served by the inoperative processor.

The communication lines interface directly with computer *memory cores* in our design. A single instruction (one memory cycle in length): (1) accepts a bit from the communication lines; (2) puts it in the proper bit location in a memory word which is employed as a character buffer for that particular line; (3) checks to see if a complete character has yet arrived; and (4) computes the parity bit for the character. The operation is accomplished almost entirely with existing equipment in the main computer memory. Additionally, it provides alternate servers for each line with sufficient decoupling to assure that no failure on one side of the interface can cause a failure on the other side. Both the method of entry and the handling of the bits within the machine will be described in what follows.

#### 3.1.3.1 *Bit Handling By A New Instruction*

The lines coming into the computer are actually wired into the memory of the computer as described in the next section. Each incoming communication line will be accompanied by a synchronizing line which specifies timing. Each of these wires is wired into a memory core at a location which is permanently reserved for that communication input. A program will cause the line termination memory locations to be scanned at times specified by interrupts from a real-time clock and will then put the received data bits into the proper position in that word. When the word is full, it will be transferred to another location and character processing will begin. A new type of instruction in the processor puts the bit into the proper place in the memory location for the line, checks to see if the location is full and computes the character parity. The entire instruction takes just one memory cycle of the computer. One additional memory cycle must be used to determine the line to be scanned next. This latter instruction is just an unconditional branch instruction whose address portion is determined when scan lists are made up.

The new type of instruction is externally determined which means that its effect is not determined at the time the program is written but rather is determined by subsequent input. This is not quite the same as a branch or skip instruction which merely constitutes a choice of where the next instruction is taken based on post-programming inputs.

With the direct interface it allows inexpensive appropriate control of a processor by a number of external users.

For purpose of accepting bits the instruction nature is determined by an incoming synch signal and by a marker bit which determines the end of character. However, the instruction is programmed in the normal manner as part of a subroutine which performs line scanning. The instruction format is as shown in Figure 2. The instruction code part of the instruction word contains a partial code and two externally set bits. The address field part of the instruction word contains the operand for the instruction. When the scan program causes this instruction to be read-out from the memory, the

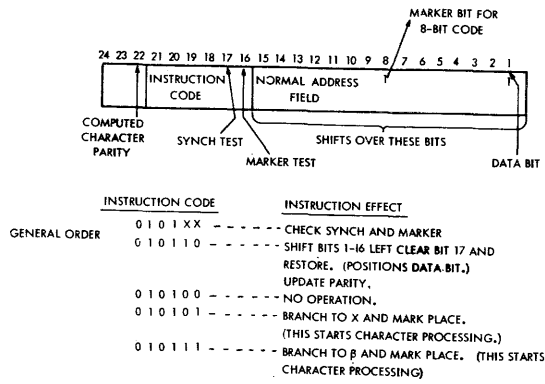


Figure 2. Instruction Formats for Bit Processing.

operation which is executed will then depend upon the two indeterminate bits. For the time being, let us assume that the second indeterminate bit is a zero. The first indeterminate bit is the synch bit, which will be a one if a new data pulse has come in since the line was last scanned. The full instruction code is then 010110 which (see Figure 2) entails a shift operation upon part of the instruction word. The data bit which was in the last significant bit of the word is shifted left one position and, thereby, entered into the partly assembled character. Simultaneously, the character parity is updated. In addition, the synchronization bit is cleared and the entire word restored in the same memory location. If when we had read the line word out, no synch bit had come in, then the instruction (010100) would be interpreted as a no-op and the word restored without modification.

The use of the marker bit is fairly simple. When we arrive at the end of a character, we would like to know about it so that the entire character can be moved to another location in core memory thence to enter on character processing. To do this, the program sets a "1" into a particular bit of the input line instruction in the normal address field. Since the address field is initially all zeros, the marker bit will be the first one to show up in the second indeterminate bit of the instruction code (due to a succession of shifts). Thus, whenever a "1" appears in this position, it indicates that a complete character has been received, and the instruction becomes a branch instruction which branches to a sub-routine to take care of transferring the character. Two different branches

are indicated here because the asynchronous nature of the system may allow a data bit to come in after the character was filled. On the other hand, there may be no new data bit that has come in. In one case, a single bit must be preserved in the memory location and in the other case, it need not be.

While this is one type of externally determined instruction, there are others possible, and in fact, the line equipment used in making the asynchronous to synchronous conversion can be embodied in an additional indeterminate synchronous bit possible in the instruction code. The full power of such an instruction particularly in message switching and command and control has not yet been realized.

### 3.1.3.2 Description of Interface Electronics

The basic system of entry into the memory is illustrated in Figure 3. Each incoming line is wired into a core which is effectively part of the main memory. These cores are all in the same bit position in the memory. The information coming into the core is written into each core on the basis of a coincidence of input-current and a write-current from the computer; the latter being supplied on every memory write cycle (by the y-drivers in Figure 3).

The information is read out of the core by the standard read cycle of the memory. Thus, once having been written in, a data bit is available for read-out with the rest of the word using the standard memory equipment already in the computer (with some exceptions to be covered later).

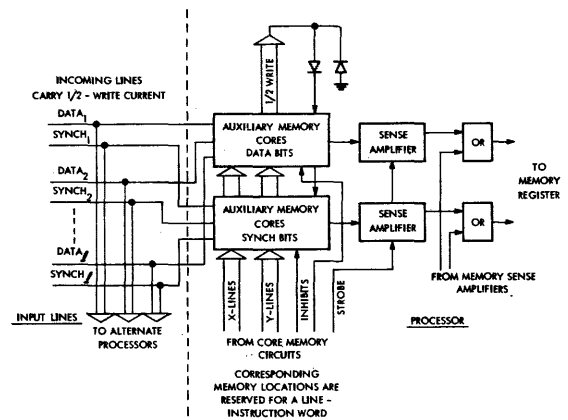


Figure 3. Processor-Input Line Interface.

However, we recognize that a single bit could be written in and read-out many times since input pulses are much wider than memory cycles. Furthermore, a read-out of a zero data bit is ambiguous since it may indicate no input rather than zero-input. Thus, we must provide a synchronizing channel for each line, which will indicate when a data bit is available for reading. A similar input to another bit of the same word will be used for such synchronization purposes. However, the synchronization pulse must be timed to a single write pulse of the computer. This means there must be an asynchronous to synchronous converter timed from the master timing source (which also times the computers) for each incoming line. While we have devised a method of employing a third channel to obviate the need for the conversion equipment, we will not discuss it here because the economic tradeoffs are not clear.

Actually, the input lines are not wound through the cores which are normally located in the main memory stack. Instead, two additional very small memory planes (Figure 3, Auxiliary Memory Cores) are provided which are the storage locations for those particular bits of memory. These planes are wired in with the x and y and z lines of the main memory. A coincident current memory will provide two half-writes, x and y. Either the x or the y write current will be provided to the external cores as a 1/2 write common to all auxiliary cores. Thus, coincidence with the external 1/2 write signal will write in a "1". A separate sense winding will be provided to prevent interference with the normal words of memory and thus a separate sense amplifier will be provided for each of these two planes. The output will be logically added to the output of the normal sense amplifiers. Thus, all the input lines can be implemented with the addition of the auxiliary planes, two sense amplifiers, a few diodes and two gates.

While each line has been described as threading a single memory, in actuality it would thread cores in three separate processor memories. Thus, any one of the processors coupled to this line could service the line if its programmed scan included the line.

On-line program modification could take care of reassignment if it became necessary. Be-

cause the input line is magnetically coupled to the memory, no processor failure can disable it; i.e., it will still deliver its write current to the other processors with which it is associated. Furthermore, if a portion of the line equipment fails, it disables the particular line but in no way prevents the processors from servicing other lines. Thus, this magnetic coupling has the sort of ideal loose coupling described earlier.

### 3.1.4 *Programming System*

A master control program schedules operating programs and provides for hardware and line assignments. Consequently, it organizes routine and non-routine activities. The portion of the control program which refers to the routine functions resides in the core memory of each processor and has the facility to call the remaining program or portions of it from the processor drum memory to core storage when required. The operation of the control program is tied to interrupt signals from a real-time clock. These signals occur as often as required for the processor to sample its incoming lines for signals and to supply information to its outgoing lines. The processor need not keep track of elapsed time.

The operation of the control program and the operational programs for the functions proceed as follows: the control program, on the basis of information describing the lines assigned to it, schedules groups of lines to be scanned at a time. When a real-time interrupt occurs the program transfers control to an appropriate program for handling the line scanning functions. If during the line scan a full character is found to have entered the machine, the character will be entered into core memory with others in its message. If the character should be a system control character appropriate action will be noted in a list kept for scheduling by the control program. When all the lines have been scanned, control will be returned to the control program. At this time, the control program decides what its next course of action will be through an examination of its scheduling list. It might examine the control characters to determine their significance. If one of these was a start of message, it would initiate a header verification and then have control

returned to it for user action. As characters are accumulated in memory or transmitted from memory by the operating routines, they would signal the control program to initiate a program which would bring more information from the drum for transmission or would have information taken from core memory for storage.

Programs for the handling of change in line assignments due to hardware failures will either be manually or automatically initiated. The manual initiation will occur from the supervisor's console. From this position, a computer will be selected and a message sent to this computer to initiate a program that would remove a computer from service and reassign its lines. This program will be retrieved from the processor drum, together with a list describing line assignments and characteristics. It will then determine another group of assignments based on an algorithm previously decided on, which is judged to minimize the overloading. Queue lengths would be available to this program if required. The change in line assignments requires no hardware change.

### 3.1.5 Processor Rate and Storage Requirements

The total number of memory cycles required for the processing job can be divided by the number of input bits to give a figure of merit which is independent of the capacity of the system. At times one sees such a figure expressed as instructions per throughput bit. However, the number of output bits exceeds the number of input bits in a system where multiple addresses are allowed. According to one estimate for a large system, the average output rate will be 75% higher than the average input rate. Thus, the proposed figure seems more natural and allows one to evaluate the required processor memory speed.

Other system factors do influence the proposed figure of merit. For example, the average number of bits per character, characters per block and blocks per message will determine the number of instructions per bit used in character and message processing. These parameters have been chosen as discussed in Section 3.1.2.

Trial programs of bit and character functions have been worked out to obtain the data

on which to assess the processor requirements. In estimating program complexity, a prototype instruction code containing twenty instructions was used. The input bit processing takes two memory cycles per bit of direct input processing and approximately 0.5 memory cycle per bit attributable to the control program. The former load may be reduced by limiting the flexibility of the scan cycle, and in fact may be reduced to 1 memory cycle per bit for a fixed or nearly fixed scan. The output bit processing is similar in memory cycle usage.

Input character processing can be done in 36 memory cycles per character and the output processing in 28 memory cycles per character. To determine the full processor rate required for each incoming bit, the bit and character process rates must be augmented by the memory cycles required for the block and message functions. Our analysis of the drum transfer and other routine block and message functions indicates that 2500 instructions per message received and 50 instructions per block received is a generous allowance for these functions (i.e., 5000 and 100 memory cycles respectively).

The total number of memory cycles per input bit is then conservatively fixed at

$$2.5 + 1.75 \times 2.5 + \frac{36}{7} + \frac{28}{7} \times 1.75 +$$

$$\frac{5000}{2000 \times 7} + \frac{50}{80 \times 7} = 19.5 \text{ memory cycles}$$

This is equivalent to ten instructions which is what most systems require for a single interrupt to process one input bit.

The core memory associated with each processor is used to hold the currently executed programs (control programs, operating programs, and maintenance programs when required), the data base for the execution of this program (code conversion tables, empties-registers, queue entries by precedence for each line, address of next ledger entry, etc.) and the messages prior to storage on the drum.

It is expected on the basis of preliminary estimates that the programming and its data would consume less than 2000 core memory words.

The message buffer is required to hold about 70 words per line. If a double buffer scheme is used to prevent buffer overlay before transfer to in-transit storage, for a computer handling 67 lines, about 10,000 words for message buffers are required per computer.

The most important facets of the processor, the interface with the communication lines and the memory size and rate have been discussed. The remaining features of the processors are of conventional nature.

#### BIBLIOGRAPHY

1. HELMAN, D. A., et al, "VADE: A Versatile Automatic Data Exchange," IEEE Trans. Communications and Electronics, No. 68, p. 478, September 1963.
2. HARRISON, G., "Message Buffering In A Computer Switching Center," IEEE Trans. Communications and Electronics, No. 68, p. 532, September 1963.
3. POLLACK, M., "Message Route Control In A Large Teletype Network," J. ACM, Vol. 11, No. 1, p. 104, January 1964.
4. GENETTA, T. L., GUERBER, H. P., RETTIG, A. S., "Automatic Store and Forward Message Switching," WJCC, San Francisco, AFIPS, Vol. 17, pp. 365-369, May 1960.
5. HELMAN, D. A., BARRETT, E. E., HAYUM, R., WILLIAMS, F. O., "Design of ITT 525 'VADE' Real-Time Processor," FJCC, AFIPS, Vol. 22, pp. 154-160, 1962.
6. SEGAL, R. J., GUERBER, H. P., "Four Advanced Computers—Key to Air Force Digital Data Communication System," EJCC, Washington D.C., pp. 264-278, December 1961.
7. WOLF, F. G., "Application of A Modular Data Processor to Store-and-Forward Message Switching Systems," Proceedings Ninth National Communications Symposium, pp. 198-207, 1963.
8. "Message Switching and Retrieval In A Real Time Data Processing System," Comm. Catalyst of Progress, National Communications Symposium, Ninth, pp. 190-197, 1963.



# VERY HIGH SPEED SERIAL AND SERIAL-PARALLEL COMPUTERS HITAC 5020 AND 5020E

*Kenro Murata and Kisaburo Nakazawa  
Hitachi Ltd.  
Tokyo, Japan*

## 1. Introduction

HITAC 5020 family consists of general purpose computing systems designed to solve a wide variety of problems for both scientific and business data processing.

HITAC 5020 system is a medium-scale junior version of this family, and would have the same performance characteristics as IBM 360/40 – 50.<sup>1</sup> A purely serial logic construction is the remarkable feature of this system.

HITAC 5020E (5020 ENHANCED) system is a large scale senior version of this family, and would have as much performance as IBM 360/62 – 70. But this system is constructed in serio-parallel logic form for economical reasons.

The central processing unit of this family is designed to rapidly and economically perform fixed or floating point arithmetic operations in either single or double precision, and even more to be able to process bit-wise variable length data.<sup>2,3</sup> It contains 18 MC, 2-phase serial transistor-diode logic circuits and helical transmission lines<sup>4</sup> for accumulators, index registers and other various registers.

Our design goals of 5020 family are as follows:

### (1) High Performance per Cost.

We have realized this feature by introduction of helical transmission lines combined with the 18 MC serial logic system.

### (2) Program Flexibility.

The refined and flexible instruction system, in conjunction with a number of multi-purpose registers to be used as accumulators, index registers, and many input-output control registers, gives powerful possibilities to the programming activities.

### (3) Simultaneity and Multiprogram Activity.

The memory time sharing, the concurrent operation of various control unit, the automatic program interruption, the memory protection, and the introduction of priority mode are prepared.

This paper reviews the engineering design of HITAC 5020 and 5020E systems with primary concentration on central processing unit.

## 2. Outline of HITAC 5020 and 5020E System

The 5020 system is organized along four basic lines, Main Memory, Arithmetic and Control Unit, I/O Channels and I/O Devices.

Figures 1 and 2 show those 5020 and 5020E system configurations, respectively, and Figures 3 and 4 are pictures of the 5020 system.

### 2.1 The Main Core Memory

The core storage of the 5020 has a capacity ranging from 8,192 words to 65,536 words (32 bits each), and is directly accessible from the

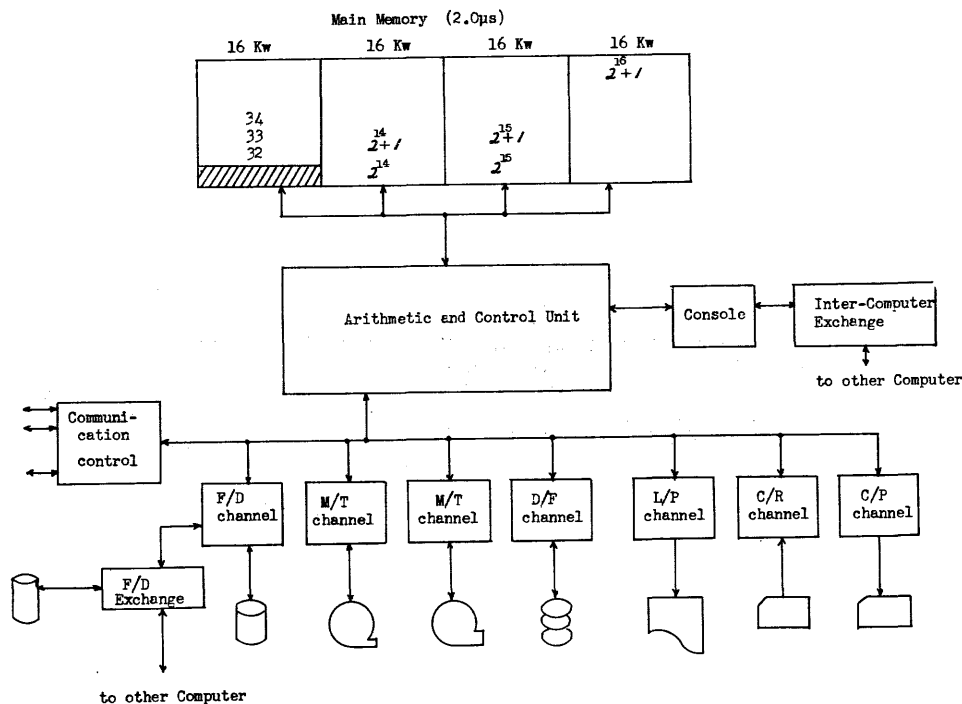


Figure 1. The System Configuration of HITAC 5020.

central processor and input-output control channels, with read-write cycle of 2.0 sec per 32 bits.

The memory addresses 0 – 31 are transmission line registers and so absent in the main core memory. The read-write control of the 5020 Arithmetic and Control Unit or Channels is rather simple one, not overlapped, and the access to the memory by the central processor is delayed only if any of the input-output channels or the central processor are attempting to access the storage at the same time.

In the case of the 5020E, the core storage can contain any one of 16K, 32K, 49K, or 65K words as a unit. Although the core storage cycle time is 1.5 microseconds, the effective increase of the access speed is realized through the completely independent access to each of the separate banks into which the whole core storage is effectively divided. (i.e., two banks in case of 16K words).

Two words (64 bits) in each bank of 16 KW capacity are referred at a time, so the effective memory speed is doubled. Moreover, each block, which is made up of four separate banks, is addressed as shown in Figure 2 (i.e. address 0, 1 in bank 0; 2, 3 in bank 1; 4, 5 in bank 2; 6, 7

in bank 3; 8, 9 in bank 0; and so on). This increases independent operation efficiency. In other words, instructions, operands and I/O informations, etc. are referred through the respective exclusive memory refer controls (four in total as seen in Figure 2). With these features the effective storage speed is doubled, while the controls are referencing the separate banks, since more than two banks are simultaneously accessed.

Instructions, operands and I/O data, etc., are, through their respective controls, transmitted via the Core Memory Multiplexer which time-shares the data flow between the core storage and either the processor or the I/O channels. In addition, during one storage cycle, data of two word length (64 bits), or three or four word length (96 bits or 128 bits), in case of the operand, can be transmitted in parallel, thereby increasing the storage access efficiency.

The 5020 instructions can designate up to 65K words (16 bits address field) of the core storage addresses. However, the 5020E is so designed that the enlargement of the core storage is possible. A field conversion up to 260K words of the store capacity is feasible.



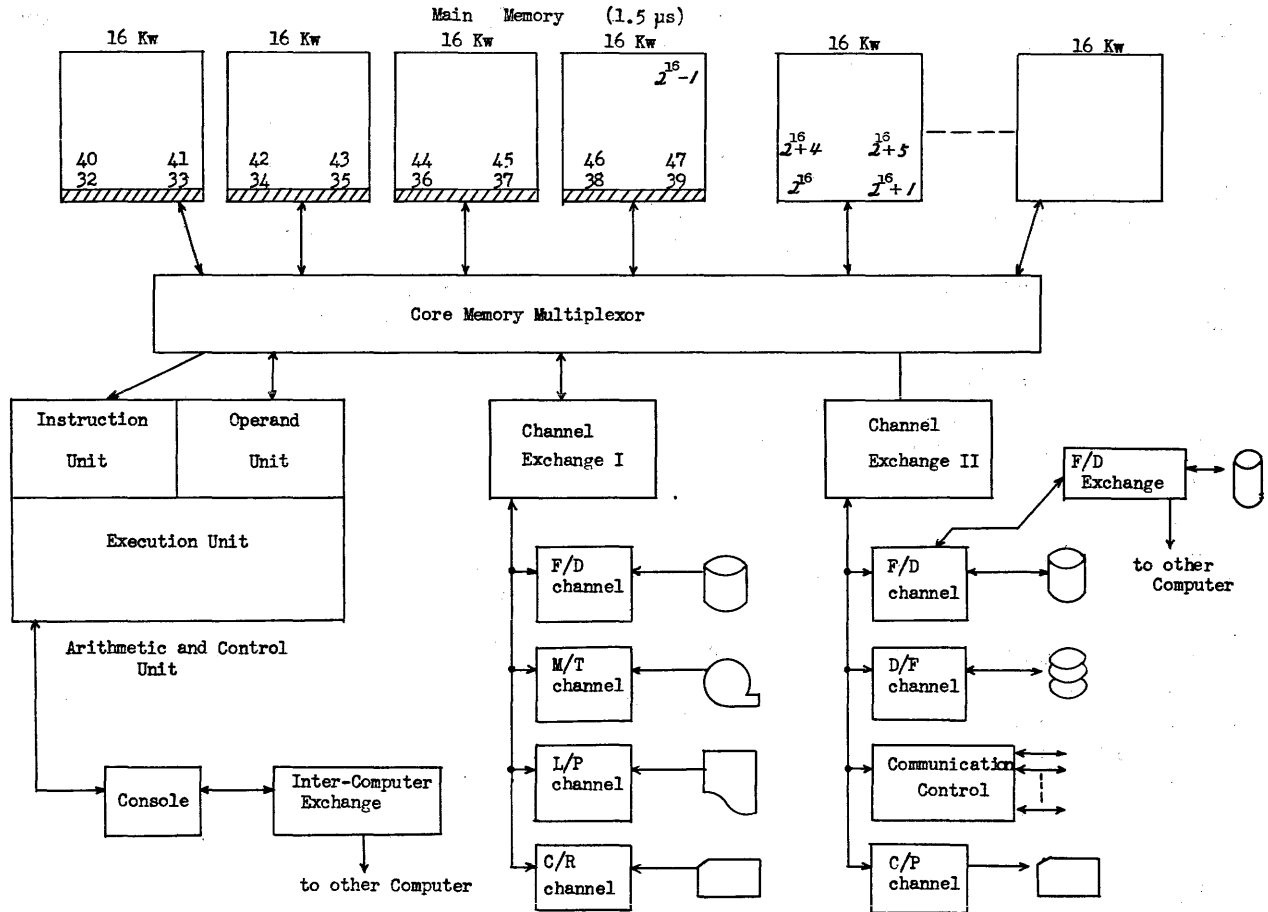


Figure 2. The System Configuration of HITAC 5020E.

In the case of more than 65 KW capacity of the 5020E, every 32 KW memory module enlargement is possible.

The effective address field of 16 bits is expanded to 18 bits (theoretically 21 bits) by

means of a preset instruction and a LMM (large memory mode) indicator bit.

The preset instruction prepares 21 bits address information which is added to index modified address field of next instruction, and

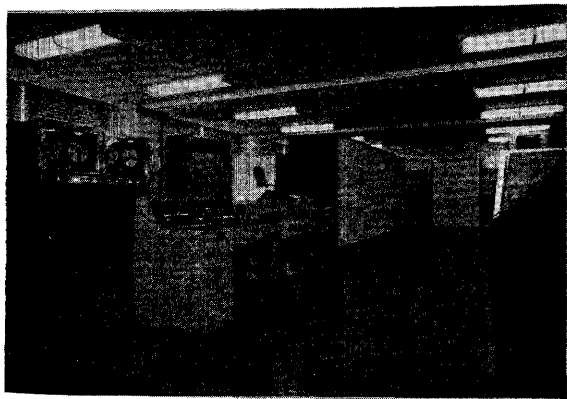


Figure 3. Front view of HITAC 5020.

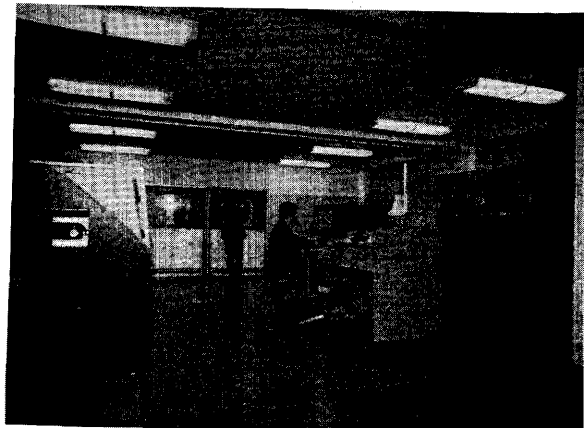


Figure 4. Console and I/o Devices.

generates the effective address field of 21 bits. When the LMM is off, the index modification is performed on 16 bits field so the information is performed seen as 0 to  $2^{16} - 1$ . When the LMM is on, the modification arithmetic is executed on 16 bits information seen as  $-2^{15} \sim +2^{15} - 1$ . Our choice of the above mentioned conversion is done for assurance of complete compatibility between the system of less than 65 KW capacity and the more.

## 2.2 Arithmetic and Control Unit

The Arithmetic and Control Unit executes and processes stored programs and data. In the 5020, the arithmetic and logical operations are all performed in serial logic. Basic cycle time is  $2\mu\text{s}$  for 32 bits information and 4 bits unavailable bits (these are used for multiplication acceleration) of 18 MC.

In contrast to the 5020, the 5020E is designed to adopt a four-bit parallel logic configuration based on the high speed circuit which has been proved to be completely feasible with the 5020 systems. By this alone, therefore, approximately four times increase in operation speed over the 5020 is readily possible.

In the 5020 the multiplication of 32-bit by 32-bit word is executed in 8 basic cycles using

four adders, whereas the 5020E can perform the same multiplication in 2 basic cycles (1.0 microseconds) using a multiplication unit.

The 5020E processor takes instructions and operands in advance, that is, three controls: the control for instructions, the control for operands and the control for executions are constantly operating in parallel. Therefore, the instruction and the operand access times would scarcely appear in the total execution time required for the instruction.

This is a kind of "advanced control" facility and will not present any trouble, since the instructions will always be executed sequentially and the interruption facility will be the same as the 5020.

Specific comparisons of the internal speeds of the 5020 and 5020E are illustrated in Table I. Comparative capacities indicate that the 5020E is approximately 8 to 12 times faster than 5020 in scientific applications and 8 times in the other applications.

Table I also shows the performance characteristics of other typical computer systems, for comparison.<sup>1,5</sup>

Furthermore, the 5020E can accept the object programs which are written in the 5020

Table I. HITAC 5020, 5020E—Execution Times (in  $\mu\text{sec}$ )  
(Including instruction read and index-modification time)

Operations	type	single or double	5020	5020E	360/40	360/50	360/62	360/70	CDC 3600 (48 bit)
		-length							
Add and subtract	fixed-point	single	8	0.75 ~ 2.25	11.88	4.0	1.87	1.05	2.07
		double	12	1.25 ~ 2.25					
	floating-point	single	14 ~ 24	3.0 ~ 3.75	18.66	6.88	3.2	1.13	4.25
		double	16 ~ 26	3.0 ~ 3.75	27.66	9.69	3.22	1.13	6.38
Multiply	fixed-point	single	24	2.0 ~ 2.5	84.72	28.52	6.12	2.8	6.70
		double	80	3.5 ~ 3.75					
	floating-point	single	36 ~ 38	2.75 ~ 3.75	80.63	21.5	6.12	2.2	6.4
		double	72 ~ 74	4.25 ~ 5.25	259.38	38.0	9.62	4.2	26.9
Divide	fixed-point	single	42	7.5 ~ 8.0	186	33.25	11.2	5.7	14.9
		double	148	20.0 ~ 23.0					
	floating-point	single	72 ~ 80	9.5 ~ 12.75	128	22.25	8.1	4.1	13.0
		double	132 ~ 142	17.0 ~ 21.0	477	69.5	17.62	7.3	27.2
Shift	single	6 ~ 10	1.0	11.25 ~ 25	4 ~ 9	1.12 ~ 1.62	0.4 ~ 1.2	1.25 ~ 2.63	
	double	6 ~ 10	1.5	12.5 ~ 43	4 ~ 15	1.62 ~ 2.87	0.6 ~ 1.4	1.37 ~ 2.63	
Jump	jump	4	0.5 ~ 1.5	8.13	4.75	1.53	1.0	2.25	
	non-jump	8	1.5	8.76	3.75	1.44	1.0	1.25	
Store	single	10	1.25 ~ 2.5	12.5	4.0	1.44	1.26	1.88	
	double	16	1.25 ~ 2.5	17.5	5.75	1.44	1.26	4.5	
Inner-loop of the polynomial calculation $Px + a_1 = P$	fixed-point	single	36	3.5 ~ 5.5					
		double	96	6.0 ~ 7.5					
	floating-point	single	64	6.0 ~ 7.0					
		double	100	7.0 ~ 8.0					
Inner-loop of the matrix multiplication $\bar{S} + a_1 b_1 = \bar{S}$	fixed-point	single	48	5.0 ~ 7.5	121	44	11.0	4.9	15.76
		double	112	8.0					
	floating-point	single	86	8.0 ~ 10.0	125	39	12.3	5.0	15.76
		double	116	9.0 ~ 11.0	317	59	16.1	7.1	
Inner-loop of the matrix inversion by a direct method $a_1 + \alpha b_1 = a_1$	fixed-point	single	54	6.0 ~ 10.5					
		double	124	8.0 ~ 10.5					
	floating-point	single	96	10.0 ~ 12.5					
		double	138	11.0 ~ 13.5					

machine language, as well as the programs written in external languages. Consequently, the various capabilities and functions of the 5020E are identical to those of the 5020 except for the increased capacities and several expanded functions.

The 5020E is upwards compatible with the 5020, which means that the 5020E can accept all 5020 programs directly as compiled on the latter system.

The 5020E is also completely downwards compatible with the 5020, which permits 5020E programs to be directly applied to 5020, and which also means that the 5020 is provided with the "deduction" ability (stored logic feature) of 5020E hardware instructions, such as quadruple arithmetic.

General Features of HITAC 5020 family<sup>6</sup> are as follows:

#### (1) *Arithmetic Operations*

The central processing unit is designed to perform fixed or floating point arithmetic in either single or double precisions.

Especially the double length floating point arithmetic is to be operated faster than in other conventional computers.

In the fixed point arithmetic operations, for large scale scientific problems requiring fine precision, instructions such as ADD WITH PRECEDING CARRY and SUBTRACT WITH PRECEDING BORROW are prepared. Moreover, in MULTIPLICATION and DIVISION, the 4-word-length product and the 2-word-length quotient with the 2-word-length remainder are available respectively.

These instructions give great conveniences to the scientists and engineers for obtaining their results with high precision. In the floating point arithmetic operations, the operand in accumulator is always given in double length even in the "single" length floating point arithmetic.

Therefore, the errors caused by the accumulative arithmetics are usually smaller than with the conventional com-

puter. This fact has been demonstrated by simulation for comparison of this HITAC-5020 arithmetic with conventional arithmetic using a single length accumulator, by means of HITAC-5020.

#### (2) *Variable-Length Data Handling and Repeat Mode*

Bitwise addressing is one of the most remarkable features in this system. All bits within the main memory can be addressed and bitwise variable length data of 1 to 64 bits length can be assigned as operand as well as fixed length data.

In the variable length instructions the bitwise address (normally 21 bit address field) modification is possible and if the repeat mode is defined by a bit in the single instruction, "scatter" or "gather" of data within memory and "table look-up" for equal, greater or less condition can be performed in the rather universal form.

These facilities, variable length data handling and repeatability, can be used in bit-based or character-based operation for sorting, merging and compiling.

#### (3) *Multi-Purpose Registers*

The HITAC 5020 system contains 14 index and arithmetic registers. Six of them serve as index registers and accumulator registers, and the other 8 registers are used as accumulator registers. These registers are also addressed as if they were in the standard core storage, but physically, they are helical-wired transmission line registers.

Since the index registers may also be used as the accumulator registers, their contents are subject to the arithmetics as well as the address modification.

This is a powerful feature in compiling and data processing programs. In accumulative operations, intermediate partial result in accumulators can be used in ensuing operations without returning them to the main storage, therefore the storing-fetching time and the round-off error are eliminated.

(4) *The Remarkable Instructions* of the 5020 family are as follows:

(a) The floating point arithmetic includes four-word length (8-bit exponent, 120-bit mantissa) operations, namely,

$$(a, a + 1, a + 2, a + 3) * (\tilde{m}, \tilde{m} + 1) \rightarrow a, a + 1, a + 2, a + 3$$

$$(a, a + 1, a + 2, a + 3) * (\tilde{m}, \tilde{m} + 1, \tilde{m} + 2, \tilde{m} + 3) \rightarrow a, a + 1, a + 2, a + 3$$

\* denotes an arithmetic code for any one of addition, subtraction, multiplication or division.

(b) Integer arithmetic instructions have been added, that is;

$$(a) * (\tilde{m}) \rightarrow a - 1, a$$

$$(a) \div (\tilde{m}) \rightarrow a, \text{remainder} \rightarrow a + 1$$

(c) Operations are included to shift data up to four-word length.

(d) Bit search instructions which allocate the specific bit address, whose content is one, to the specific memory address.

(e) A preset instruction (Modify Next Instruction) which utilizes any of the memory addresses as if they were index modifiers of next instruction.

(5) Functions, as shown below, are *optional features* of our system, such as,

- (a) Binary to Decimal Conversion,
- (b) Decimal to Binary Conversion,
- (c) Translate by Table,
- (d) Edit,
- (e) Decimal Arithmetic Operations, etc.

(6) *Adaptability to Multi-computer System*

For both the 5020 and the 5020E, considerations are given to the multi-computer system configurations as well as the single computer system configurations based on each of the systems. For instance,

(a) Function which allows a program to turn on and off the magnetic tape

switches, or the magnetic drum switch share.

(b) Function to interrupt other computers or transmit a communication information.

(c) Main memory share capability has been provided.

### 2.3 Input-Output Control Channel

Input-output units are linked with the memory and the central processor through the I/O control channels. After an initial instruction from the computer, the channel controls information transmission between the I/O unit and main core storage in accordance with the "commands." Up to 12 channels are available, and an arbitrary arrangement is permitted. To a certain channel, however, the same type of I/O units are to be connected and the maximum number of units which can be connected is assigned.

### 2.4 Input-Output Units

Peripheral units, such as magnetic drums, magnetic tape units, printers, etc., can be connected directly to the input-output control channels.

Tables II and III review the typical I/O devices of the 5020 and the 5020E systems.

## 3. Instruction System

### 3.1 Number Representation

Figure 5 shows the internal representation of numbers in the HITAC 5020 family. Negative numbers in fixed point operand or floating point fraction part are represented in 2's complement form, and negative exponents in 256's complement form. Bitwise variable length data are referred by instruction with the left most bit address (21 bits information) and the bit length (6 bits information) of the field.

### 3.2 Memory Address Assignment

Figure 6 shows the memory address assignment, and Figure 7 shows the bit assignment of special registers. As seen in these, almost all information that represents the internal state of the 5020 or 5020E system is assigned in the particular bit in the main memory.

Table II

Type #	Type	Capacity/Device	Transfer Rate		Av Access or Start Stop Time	RPM or Tape. Vel.	Max No.	Max No. of Device per CH.
			(6 bits)	(3 bits)				
H-366-1	D/F	22 Mch	32 Kc/s	6 Kw/s	125 ms	1,200 rpm	2	4
-2	"	44 "	" "	" "	"	"		
-3	"	66 "	" "	" "	"	"		
-4	"	88 "	" "	" "	"	"		
H-179-1	F/D	32 Kw		51 Kw/s	10 ms	3,000 rpm	2	8
2	"	49 "		" "	"	"		
3	"	65 "		" "	"	"		
H-582	M/T		66.7 Kc/s	12.5Kw/s	5.5ms	2.54 m/s	8	6
H-175	M/T		24 Kc/s	4.5Kw/s	7.0ms	3.0 m/s	8	6
H-3485 *	M/T		30 Kc/s	5.6Kw/s	3 ms	3.81 m/s	8	6
			83.4 "	15.6 "		"		
			120 "	22.5 "		"		

I/O Device (1)  
\* 8 bits (9 tracks)H-3485 is also available

D/F: Disc File  
F/D: File Drum  
M/T: Magnetic Tape

3.3 Instruction Set and Instruction Format

The instruction set of the HITAC 5020 may be classified as follows:

- (1) 6 types of short (32 bits) instructions for fixed length data. These include (a) immediate, (b) jump, (c) fixed point arithmetic types including logical instructions, (d) floating point arithmetic types, (e) store, and (f) miscellaneous.
- (2) 2 types of long (64 bits) instructions for bitwise variable length data. These include (a) jump instructions, and (b) 2-address arithmetic or logical instructions.
- (3) Long (64 bits) instructions for input and output "command."

The short instructions for fixed length data handling may be called "local 2-address instruc-

tions." Their format is shown as Format I in Figure 8.

where,

- F; function
- B; specify a modifier (index register)
- I; indirect addressing bit
- A; specify a working arithmetic register or index register
- V; variation; short (32 bits) or long (64 bits) (for the length of operand) left (0-15) or right half (for the A-register) etc.
- M; memory address part or immediate number

When M = 0, I = 0, B = 0, the operand address of this instruction is not #0 address but next location of this instruction.

(a) M ⊃ A ⊃ B

M refers to all core memory addresses and helical delay line register addresses, address numbers 0 through 65,535. A refers to addresses 0 through 15 which are the helical delay line registers and serve as fast access multi-purpose registers. Thus, A designates an arithmetic register or an index register. In the case of "Jump on Indicator" instruction, A refers to the local bit address of the

Table III

Type #	Type	Speed	No. of CH	Max No. of CH	Max No. of Device per CH
H- 333	L/P	1000 l/m	120 ch	2	2
		800 "			
H- 329	C/R	1,470 "	80 Col	2	2
H- 334	C/P	100 "	80	2	2
H-3436	C/P	300 "		2	2

I/O Devices (2)

L/P: Line Printer  
C/R: Card Reader  
C/P: Card Punch

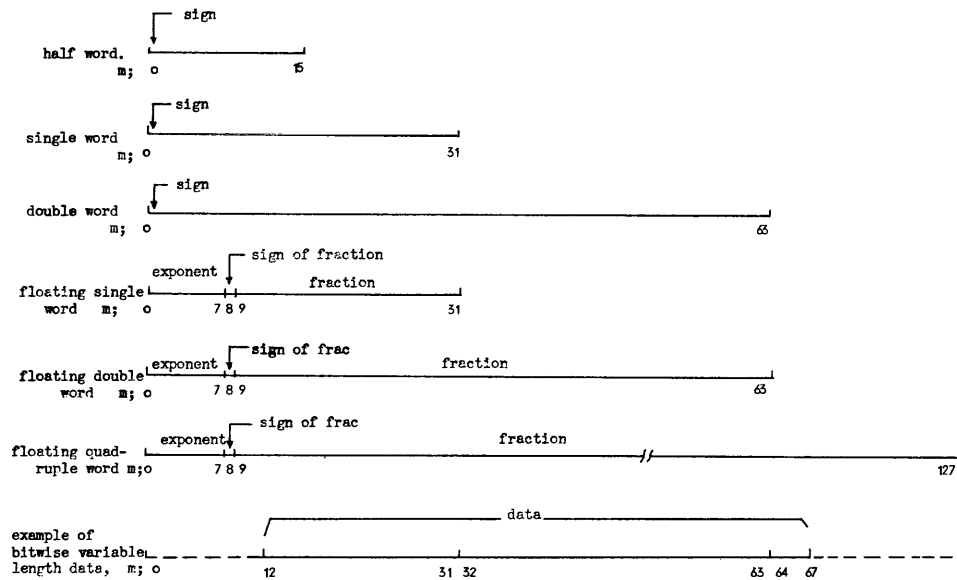


Figure 5. Number Representation.

indicator bit which is accommodated in the address 17 register. B refers to the index registers which serve as "zero access modifiers" of the M-address of an instruction.

- (b) Each instruction operates on A-addresses indiscriminately, i.e., there is no distinction between index arithmetic, pure arithmetic and logical operation in the addresses from 0 to 15. For example, "add" instruction may be used for an addition of two index or arithmetic registers. Some advantages of this instruction system are:

- (i) All registers are always efficiently available in the program sequence.
- (ii) Arithmetic results may immediately be used as index modifiers. This is remarkably useful in the compiling and data processing programs.
- (iii) In many cases, several program sequences may continue without store instruction. This will shorten the execution time and eliminate the round-off error.
- (iv) The operation of a program sequence may be reduced by using A- or B-registers in two-address way.

The long instructions for bitwise variable length data are divided into two categories, Formats II and III in Figure 8, bitwise variable length data handling and variable length conditional jump instruction, respectively.

where,

F; function example VA (variable length add); VBM (variable length bitwise multiply) (AND) etc.

$B_1, B_2$ ; modifier of first operand address

$B_2, I_2$ ; indirect bit of first operand address

$M_1^*$ ; bit address of first operand

L; length of operands

W; repeat mode bit

$M_2^*$ ; bit address of second operand

A; A-address in which there is a number to be compared

$M_2$ ; jump address

V; variation bit (absolute or not)

Some specific features of this instruction are:

- (i) Bitwise addressing and bitwise variable length data handling.
- (ii) Complete 2-address.
- (iii) Bitwise modification.
- (iv) Repeatability by W-bit.

In the repeat mode, many data may be scattered or gathered from memory to memory in

the most universal form, where source address interval  $d$ , destination address interval  $d'$ , count number  $n$  and current count-up  $p$  are all assigned in the addresses 2 and 3 as shown in Figure 9(a) for Format II, (b) for Format III.

Repetition stops when  $n = p$  or the jump condition is fulfilled, and when the repetition is the case of jump instruction (table look-up), the bit address of the current address is stored

from 0 through 20 bit address of address 3. After this the jump occurs. This operation is used in the "table look-up."

The instruction format for the input and output "command" is shown in Format IV in Figure 8.

where,

F; Function

B; Modifier of DA parts

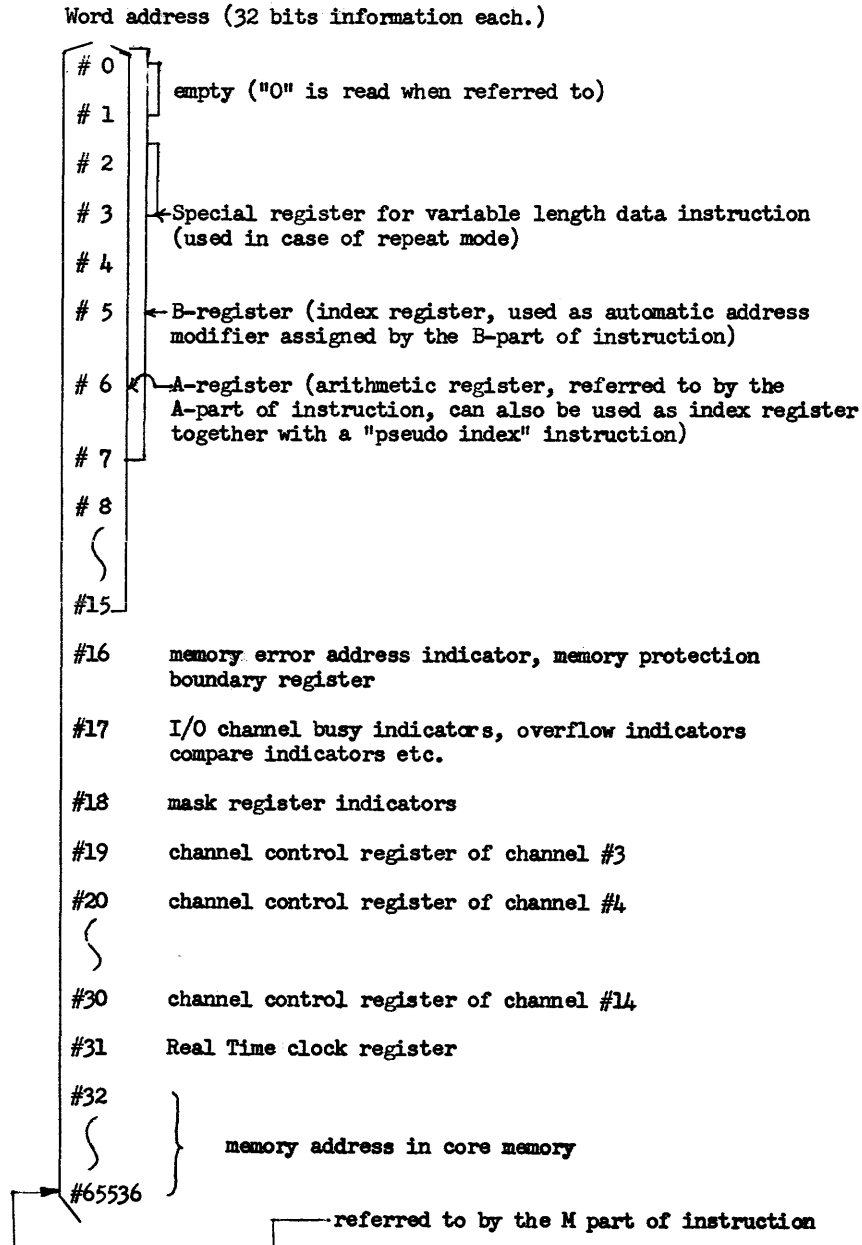


Figure 6. Memory Address Assignment.

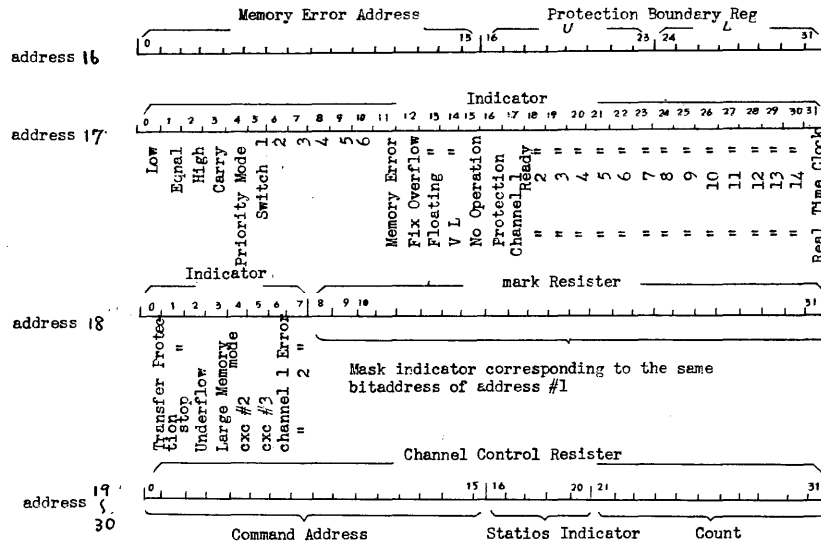


Figure 7. Assignment of Special Register Bit.

- I; Indirect address bit
- A; Channel number specification
- B<sub>2</sub>; M<sub>2</sub> part modifier
- I<sub>2</sub>; Indirect address bit
- M<sub>2</sub>; "Command" location specification
- D; Device number specification
- W; Mask bit setting
- L; "PS" instruction variation

#### 4. Input-Output Channel and Its Control

The I/O control of the 5020 family is executed by means of I/O channels, and the Arithmetic and Control Unit or the channel time-shares main memory as occasion arises. An I/O channel is exclusive to a certain type of I/O devices, and the simultaneous operation of maximum 12 channels is allowed.

The channel operation is initiated in Arithmetic and Control Unit by I/O instruction (Format IV), which is valid when and only when the Priority Mode Indicator (the 4th bit of address 17, of Figure 7) is on.

The L part of an I/O instruction assigns sub-functions such as Read, Read Backward, Write, Advance, Advance Backward, Rewind, etc.

The channel having received the control, starts the designated unit and transfers informations between I/O unit.

The transmission area is defined by "commands" (as shown in Figure 8), which are initially addressed by a I/O instruction and are fetched from the main core memory also by use of memory read facility of the channel. The command operations can be chained by designating a bit in a command.

Each part of a command shown in Figure 8 represents the following:

- M; initial main memory address or drum address
- N; number of word or block to be processed
- R; transfer or skip
- S; variation of operation
- T; disconnect or proceed

The adoption of commands in the 5020 and 5020E systems makes the I/O operation more efficient, as so called scatter read, gather write or symbol control is possible.

Having left the I/O control to the channel, the central processor processes ensuing main program simultaneously with the I/O operations of the channel. Its memory refer process is delayed only if any of the I/O channels are attempting to access the main storage at the same time.

The status of a channel in operation or after operation is indicated in the channel Control Register as shown in Figure 7 and can be referred by main control programs.



5. Program Interruption and Protections

The program interrupt conditions are:

- Memory error
- Overflow in fixed point, floating point or variable
- Length arithmetic operations
- Memory protection and No operation
- Ready conditions of I/O control channels
- Real-Time clock interrupt
- Manual Console switch, etc.

These conditions are indicated in the address 17 Indicator register (Figure 7).

The built-in circuits continuously and automatically check for above conditions and cor-

responding contents of mask register (address 18, as seen in Figure 7).

These program interrupt conditions cause an interruption only when there is a "one" bit both in Indicator Register and Mask Register, and when the Priority Mode Indicator (4th bit of address 17) is off.

If the program interruption occurs, the main program routine is turned off, the Priority Mode Indicator is forced to turn on, the current content of SCC is stored to the address 32 with the instruction word count, and a forced jump is made to the address 33 from where begins Master Control Program, the routine handling the specific conditions.

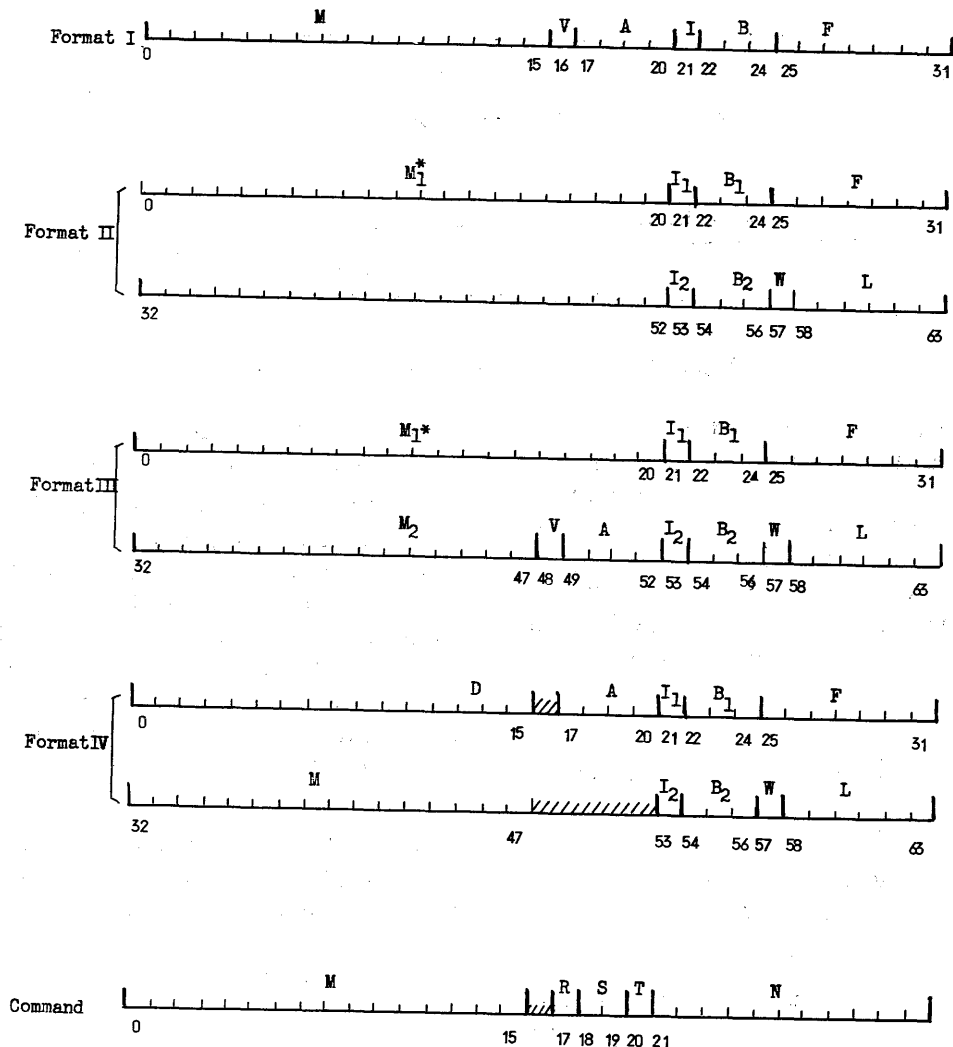


Figure 8. Instruction Format.

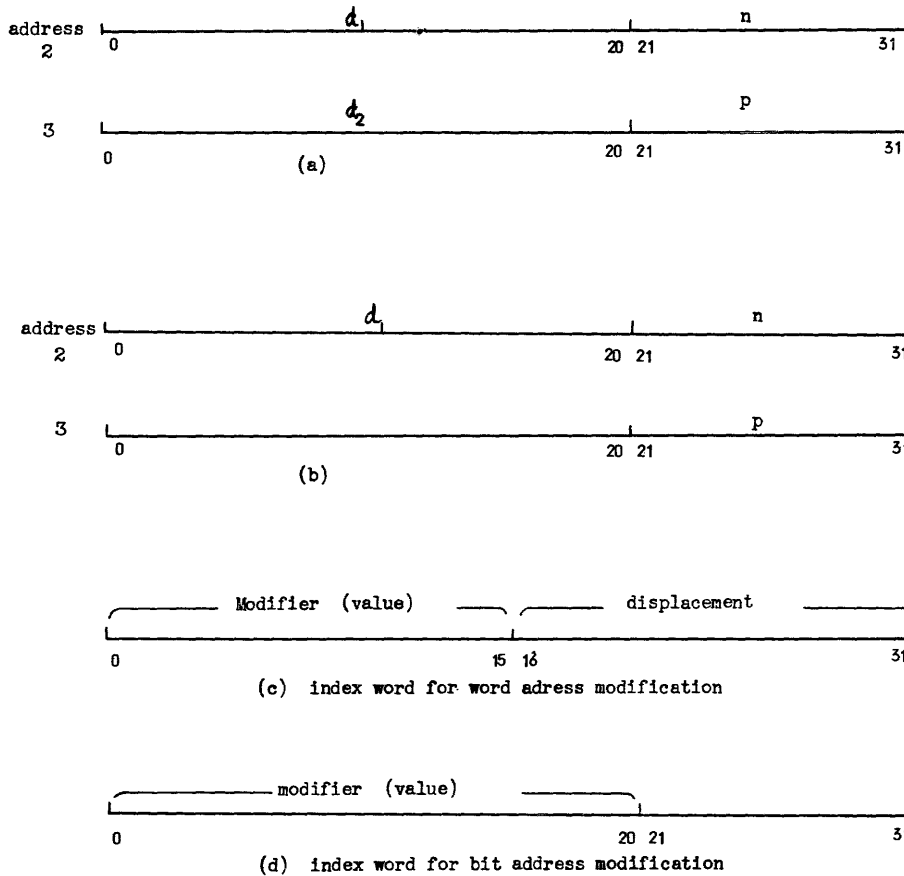


Fig. 9. Repeat Mode Modifier and Index Word.

The right half word of address 16 is the memory Protection Boundary Register (as shown in Figure 7), U or L assigns the upper bound group number or the lower bound, respectively, where a group represents every 256 words memory locations, each group having an address that is a multiple of 256. (When the main memory capacity is over 65 KW in a 5020E, every 1024 words.)

The protected memory address X, whose group number is X, is assigned by U and L, such as

- if  $U > L$  ,  $U > X \geq L$
- if  $U = L$  , no region is assigned
- if  $U < L$  ,  $X < U$  or  $L \leq X$

When and only when Priority Mode Indicator is on, all the instructions which will have store operation to the above-mentioned protected area are suppressed from storing, and this is indicated by Transfer Protection Indicator (0th bit in address 18) and Protection

Indicator (16th bit in address 17). So the program interruption will occur.

The purpose of this protection is to eliminate the interactive of programs in multiprogramming operation. Especially when the system is operating under supervision of the monitoring system, it is necessary to safeguard the monitor from harm of supervised program, for these situations cause not only a single program but all the system to stop. Moreover, stop protection facility is provided in the 5020 family.

Thus, automatic program interrupt ion system and protection facility have made effective multiprogramming possible.

## 6. Logical Structure

### 6.1 Logic Circuit

The logic circuit of the 5020 family is fully synchronous, 2 phase (B phase) static circuit. Figure 10(a) shows a basic regenerative ampli-

fier or flip-flop, in which the connection A is specially added to eliminate the hazard of misoperation and to have room of the clock phase adjustment. Figure 10(b) is the symbolic representation of circuit (a). (c) in the same figure shows the rules of worst case logical connection, that is, two levels of pair logic are permitted. Max number of fan-in of logic circuit is 7 and max fan out of logic is 6. (d) shows the helical-wired transmission line (delay line) register of 36 bit information in the 5020 system.

6.2 Arithmetic Unit of the 5020

In the 5020 arithmetic unit, the pure serial logic and the delay line registers feature reveals especially its simple characteristics in multiplication, division and shift operation. The multiplication and the division in the serial computer are performed by a series of additions, subtractions and shifts. Therefore, if the most simple procedure is adopted, it will take

32 words cycle time to execute the multiplication of one word by one word. Particularly for scientific uses it is important to reduce the execution time of the multiplication. The summarized procedures of the multiplication and the division are as follows.

(1) Multiplication

The HITAC 5020 has five binary Adder & Subtractors. In the multiplication, four Adder & Subtractors of them are connected in series. The multiplier is divided into eight parts, each of which contains 4 bits information. When the multiplicand passed through four Adder in series according to 4 bits information, the partial product of one-word by one eighth word is obtained. Therefore the multiplication of one word by one word is performed only in 8 words cycle time (16  $\mu$ sec).

In the case of Integer Multiply (IM), five Adders are used. So it takes only 1 word cycle

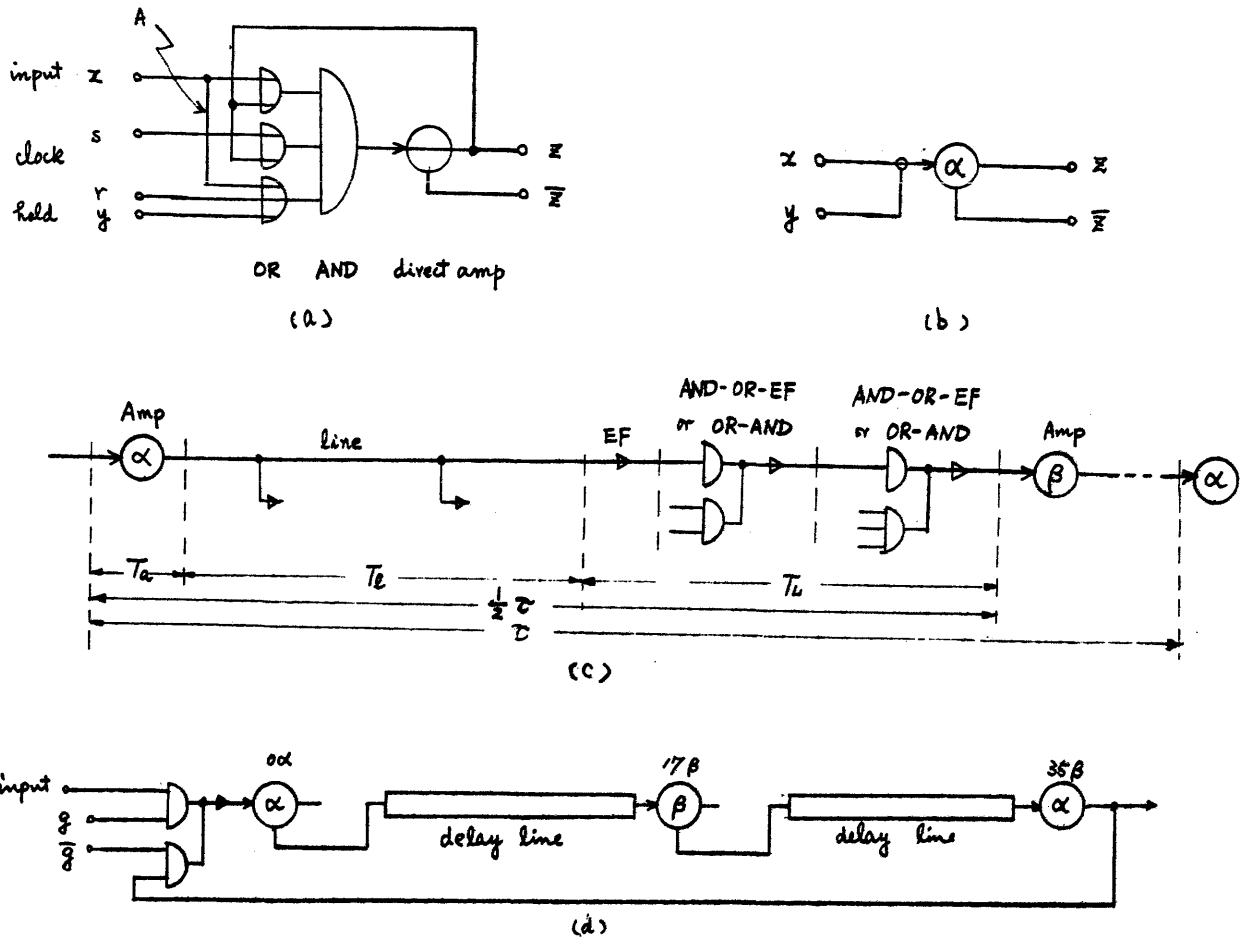


Figure 10. Logic Circuit.

time to carry out the multiplication of one word by the integer less than 32. This scheme is illustrated in Figure 11.

(2) *Division*

Generally the non-restoring method is suitable for the procedure of the division in computers. According to the test result of signs of the divisor and the partial remainder, the divisor is added to or subtracted from the partial remainder shifted one place to the right.

In the HITAC 5020, this method is more improved to reduce the execution time of the division by using three binary Adder & Subtractors. (See Figure 12.)

The result of the addition or the subtraction is transferred to the Adder 2 and the Adder 3, in which the divisor is added to or subtracted from the result in the Adder 1, respectively. These two results appear in the Auxiliary Registers 1 and 2. At the end of this operation, the true result is selected automatically by checking a sign bit of the result in the Adder 1. Thus, the division of one word by one word is performed in 16 words cycle time (32  $\mu$ sec).

(3) *Shift Operation*

In the HITAC 5020, the shift operation is performed by means of the shifter. Owing to very high clock frequency, the delay lines can be used as various registers instead of the transistorized register, and the use of delay lines is very eminent from the viewpoint of economy and reliability. The Shifter, of course, is made of delay lines. For instance, n bits left shift is carried out by making the contents of a register pass through the delay lines of n bits

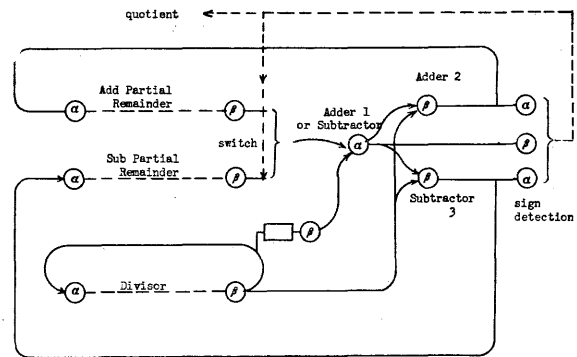


Figure 12. Division Unit of the 5020.

length. The Shifter of the HITAC 5020 consists of delay lines of 0, 1, 2, 4, 8, 16, and 36 bits length. When the number to be shifted n places is set in the Shift Control Register (SCR), these delay lines are connected automatically to make the total length n bits. Every kind of shift operation, thus, can be performed by very simple logical circuits. Although the one-word information of the HITAC 5020 is 32 bits, the one-word delay line registers have 36 bits length for certain reason (ex. multiplication). Therefore, the recirculation time of delay line register is 36 bits cycle. Some consideration must be taken to shift the two or more-word information because of 4 bits spare time. For this purpose, the coincidence circuit between the shift control register and the counter is provided, which controls two output of the shifter (see Figure 13).

6.3 *Arithmetic Unit of the 5020E*

(1) *Multiplication Scheme of the 5020E*

To perform the multiplication of one word by one word only in two words cycle time (1.0 $\mu$ s in the case of the 5020E), we use eight 4-bit-parallel-adders, A1, A2 . . . . A8 (see Figure 14). The multiplicand is passed through the fifteen black boxes which are n time circuits, named N1, N2, . . . . N15 and we get 15 outputs (i.e. y, 2y, 3y, . . . . 15y) simultaneously. Now, they are fed into eight gates numbered G1 through G8. On the other hand, the most significant four bits of the multiplier control the gate G1. That is, they select one of the 15 outputs, mentioned above, or inhibit all of them when the 4 bits are all zeros. The next more significant 4 bits of the multiplier

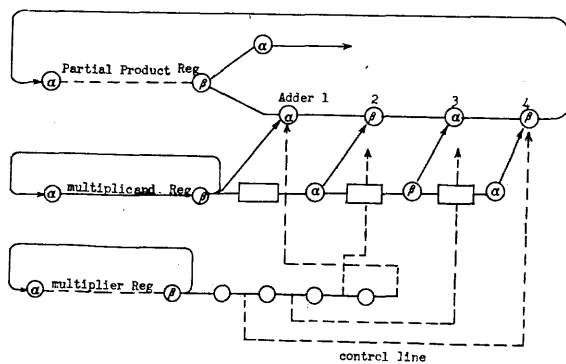


Figure 11. Multiplication Unit of the 5020.

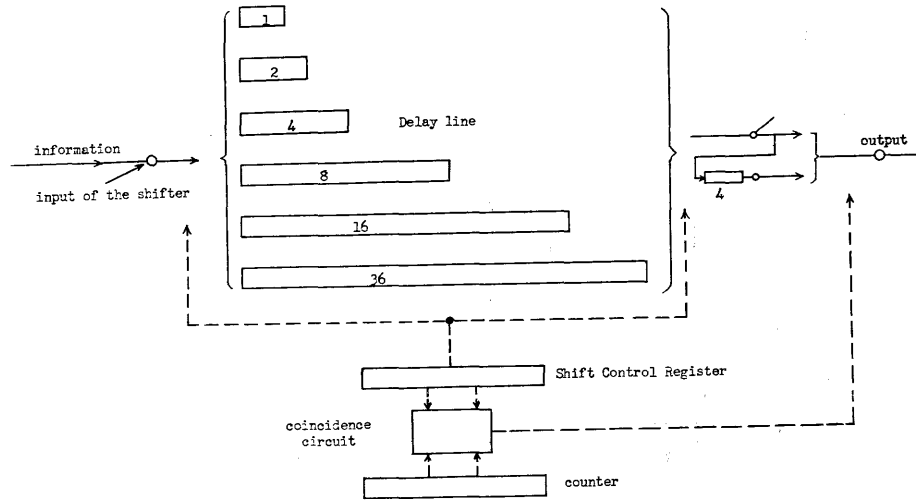


Figure 13. The Schematic Diagram of the Shifter of the 5020.

control the gate G2 and so on. The outputs of the gates are fed into the corresponding adders and the adders are connected in series. So we can get the two-word product from the output of the adder A8 in 2 words cycle time.

(2) Division Scheme of the 5020E

To speed up the division process, we modify the non-restoring method slightly and obtain 4 bits quotients in one word cycle (0.5 μs).

In this method, comparing the sign of the divisor (y) with the sign of the partial remainder (r<sub>i</sub>), we can determine the next quotient bit (q<sub>i</sub>) and whether to add or subtract next. But still there are eight possibilities left. That

is, how many times of the divisor should be added to (or subtracted from) the 16 γ<sub>i</sub>.

$$r_{i+4} = 16 \gamma_i \pm (2k + 1) y \quad (k = 0, 1, \dots, 7)$$

Comparison of the most significant 5 bits of the divisor (y) with the most significant 6 bits of the partial remainder (r<sub>i</sub>) allows us to restrict the above eight possibilities to only two. (The divisor must have been normalized.) Suppose that we could find k to be n or n+1 (n = 0, 1, 2, . . . 6). Then, perform three additions (or subtractions).

$$r_{i+4} = 16 \gamma_i \pm (2n + 1) y$$

$$r_{i+4} = 16 \gamma_i \pm (2n) y$$

$$r_{i+4} = 16 \gamma_i \pm (2n + 3) y$$

Test the sign of r<sub>i+4</sub> and we can determine which of the two possible remainders (r<sub>i+4</sub>, r<sub>i+4,2</sub>) is right. And, of course, we can obtain the right quotient bits. This method is schematically illustrated in Figure 15 where N1, N2, . . . N15 are the n times circuits which are used also in multiplication operation; G1, G2, G3 are gates which choose the n multiple of the divisor by the information of the most significant 5 bits of divisor (D1, D2, . . . D5) and of the 6 bits of the partial remainder (R1, R2, . . . R6). AS1, AS2, AS3 are the Adder-Subtractor and P is a circuit to select the right partial remainder out of two possible ones.

7. Circuitry

Recent advancement of transistor technique is remarkable and enables us to easily realize

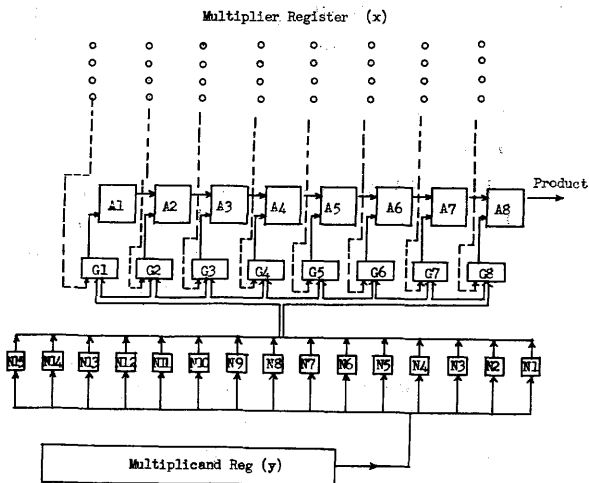


Figure 14. Schematic Diagram of the Multiplication unit of the 5020E.

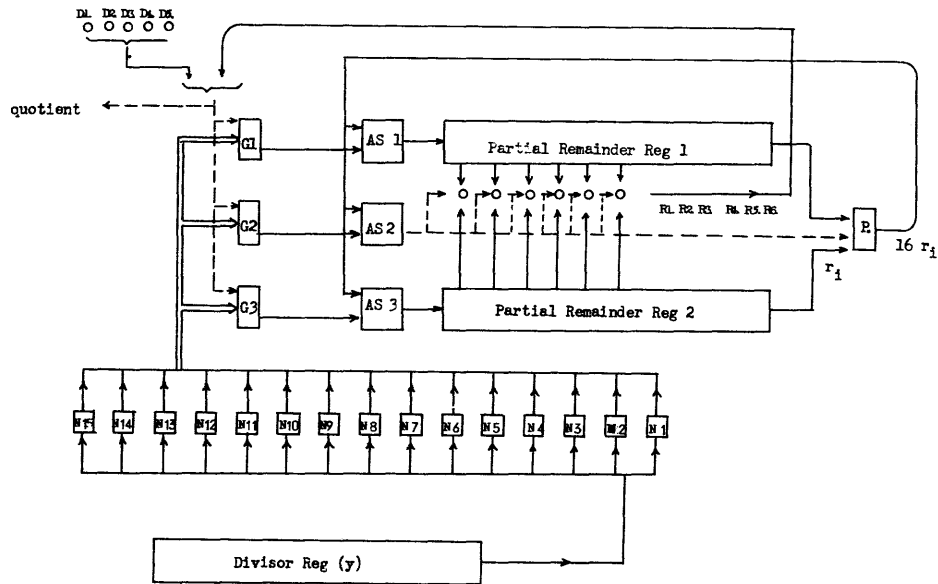


Figure 15. Schematic Diagram of the Division Unit of the 5020E.

the circuit that operates at over 10 MC clock frequency. Moreover, we have utilized the current switch mode circuit to make the most of its high speed feature. For AND-OR logic circuit, however, the diode logic and emitter follower fan out is adopted, based on economy and logic density vs. speed studies.

Figure 16 illustrates the circuit configuration of Amp (which is the same circuit as shown in Figure 10(a), (b)) (a), AND-OR logic (b), and OR-AND logic.

The "zero" level of signal is ground and the "one" level is + 3 V as is done in other current switch mode circuitry. The collector of the amplifier directly drives the signal transmis-

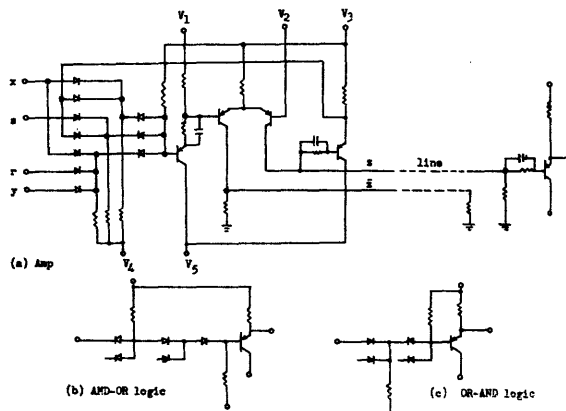


Figure 16. Circuitry of the 5020 Family.

sion lines, which are twisted pairs or coaxial cables, and the collector load is the matching resistance (75 or 100) of these lines. The AND-OR logic circuit is driven by EF's (emitter followers) which are almost uniformly distributed on signal lines, and transform the source impedance to low. The reasons why we do not drive the line by EF, but from collector directly, are as follows:

- i) The maximum fan out of EF is confined by its  $P_c$  (power dissipation of transistor).
- ii) The direct connection of logic input to transmission line disturbs the line characteristics.
- iii) The problem of damping oscillation of EF is serious.
- iv) It is necessary to decrease the collector load resistance as far as the characteristic impedance of line in order to attain very high speed switching of transistor, so our configuration has no losses.

The signal transmission lines are the foamed polyethylene twisted pair lines and coaxial cables, whose length, for the worst case of half phase 2 level logic, must be less than 1.8m. The output lines of EF are single lines less than 0.5m length.

The computer is fully taken care of ground construction and power distribution, for these may otherwise cause very serious problems in very high speed circuit.

The circuit data of Figure 16, or Figure 10 are shown below.

T<sub>a</sub>; delay time of Amp including logic witch clock, 6.5 ns

T<sub>a</sub>; delay time of Amp including logic witch line, 7.5 ns

T; delay time of two levels of pair logic, 8.5 ns

These, all added up, have sufficient room for half phase of 18 MC, clock period, that is 27.8 ns. Actual computer system has been operating with sufficient margin, and proved to be completely feasible.

#### 8. Core Memory

The 5020 Magnetic Core Memory is a word-arranged, linear-select system utilizing one ferri-rite core per bit in a partially-switched mode.

The basic memory module contains 8,192 words (33 bits per word including one parity check bit) arranged in an array of 4,096 access lines of 66 bits (2 words) each. This enables the transfer of 2 words or up to 64 consecutive bits per module in a single memory cycle which is 2.0 microseconds for 5020 and 1.5 microseconds for 5020E. Expansion of memory capacity can be made in modules or banks, as described in 2.1. Each 30-18 mil high speed core in the memory array is threaded by three windings: One winding each for the word read and write currents, one winding for the common sense-digit line whose winding scheme is such that optimum noise cancellation is achieved and the rather liny line is properly terminated in view of the high speed operation involved. Word selection is accomplished by means of a large steering diode matrix in conjunction with a transistorized cross-bar switch; drive current are derived from the constant current switch, flow into the selected steering diode, word winding and finally the voltage switch. The access control and information flow control for the memory modules are provided for every 16 KW as a unit. The built-in checkout circuit serves to quick maintenance

and troubleshooting without the aid of the central processor.

#### 9. Summary

The paper has described the outline of hardware aspect of the HITAC 5020 and 5020E. The design goal of this family, such as high performance per cost, is achieved by 18MC fully synchronous 2 phase logic circuit, helical transmission lines for various purpose registers such as accumulators, and serial or serio-parallel logic structure. 10 systems are now in production, the largest system of which will be installed at the Tokyo University and consists of a 5020E and two 5020's. This will be one of the most advanced integrated systems in Japan, and in the world as well, which is expected to play an important role and contribute to the advancement of Japanese sciences and engineering with its full-fledged power.

#### ACKNOWLEDGMENT

The authors wish to acknowledge Prof. T. Simauti, St. Paul's University, Tokyo, for his considerable contribution to the development of system philosophy and software considerations. Thanks are also due to S. Anraku, Hitachi Central Laboratory, for his works in the logical design, especially in the design of 5020E control unit, to Y. Onishi and M. Tsutsumi, Hitachi Central Laboratory, for their memory design works, and to many other people associated with the 5020 and 5020E project.

#### REFERENCES

1. Standard EDP Report, "IBM System/360" (Auerbach Corporation, Philadelphia, Penna., June 1964).
2. E. BLOCH, "The Engineering Design of the STRETCH Computer," Proc. EJCC, No. 16, p. 48, 1959.
3. W. BNCHHOLZ, *Planning a Computer System* (McGraw Hill Book Co., Inc., N. Y., 1962), chap. 7, p. 75.
4. I. A. D. LEWIS and F. H. WELLS, *Milli-microsecond Pulse Techniques* (Pergamon Press, London, 1959), chap. 2, 3, p. 47.
5. Standard EDP Report, "CDC 3600," *ibid.*
6. Hitachi Ltd., *The Instruction Manual of HITAC 5020* (Hitachi Ltd., Tokyo, 1963).





# IBM SYSTEM/360 ENGINEERING

*P. Fagg, J. L. Brown, J. A. Hipp, D. T. Doody*

*International Business Machines Corporation, Poughkeepsie, New York*  
and

*J. W. Fairclough*

*International Business Machines Corporation, Winchester, Hampshire, England*  
and

*J. Greene*

*International Business Machines Corporation, Endicott, New York*

## INTRODUCTION

The cornerstone of the IBM System/360 philosophy is that the architecture of a computer is basically independent of its physical implementation.<sup>1</sup> Therefore, in System/360, different physical implementations have been made of the single architectural definition which is illustrated in Figure 1.

One of the initial decisions was the number of processors to implement. Specifically, should it be four or five (considering the Model 60/62 as one). The original decision of five was based upon a planned increase of about 2.5 to 3 in internal performance between one model and the next.

Another fundamental decision was to provide full compatibility, both upward and downward, over the entire range of the IBM System/360. This decision was motivated primarily by the advantages, both to IBM's customers and to IBM, of the interchangeability of software.

It was clear to Engineering that the cost targets for each model in System/360 would be feasible only if a significant breakthrough were made in costs of building transistorized computers utilizing the IBM SMS technology. Therefore, we decided in 1961 to utilize the

micrologic components that were then being developed by the IBM Components Division.

The most significant features of this Solid Logic Technology (SLT) are the module, which replaces discrete transistors, resistors, diodes, etc., and the two-layer printed wiring which replaced most of the discrete wires. The modules are assembled on small cards in groups of 6, 12, 24, or 36. The small card is the basic replaceable unit. These small cards, in turn, plug into large multilayer printed circuit cards (approximately 8.5 x 13 inches). Interconnections between large cards are made by flat multiconductor tape cables that plug into large cards in the same way that the small cards do, and which run in channels between the large cards. See Figure 2.

Experience with read-only storages was derived from an experimental computer built in 1960-1961 at the IBM Hursley Laboratory in England. There were two major reasons for the general adoption of read-only storages in System/360.

1. *Assist downward compatibility due to the cost advantages.* Read-only storage (ROS) showed an advantage in cost over the circuitry which it replaced. ROS is used primarily in the control section of the system and its advantages be-

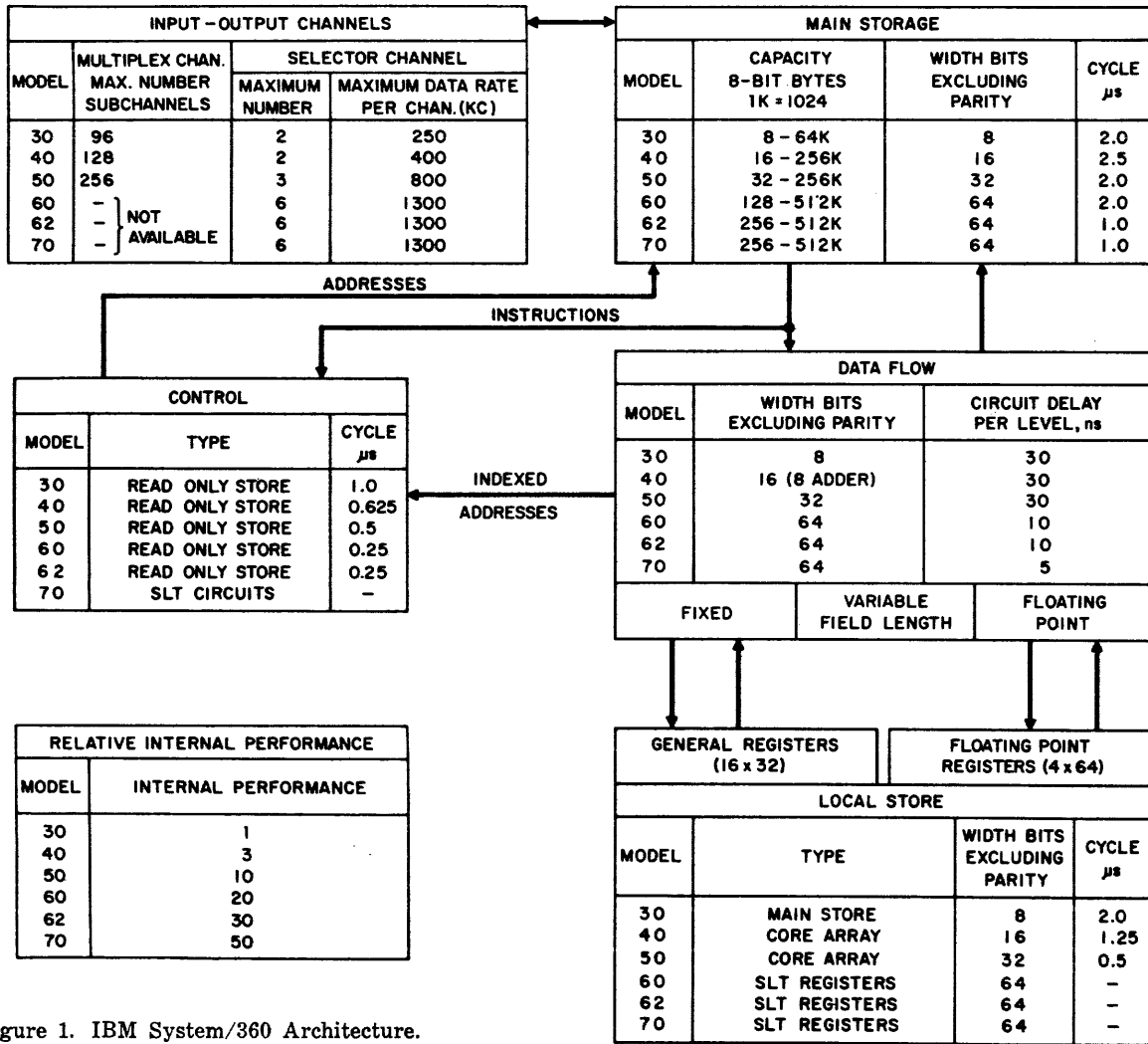


Figure 1. IBM System/360 Architecture.

comes more pronounced when more functions are to be performed. Therefore, ROS units showed a method of maintaining full compatibility by allowing complex function even in the smallest systems.

2. *Flexibility, such as the implementation of compatibility features with other IBM systems.* A unique flexibility is achieved by being able to add to the control section of the computer, when implemented by a ROS, without significantly affecting the remainder of the system. One result is to allow certain System/360 models to operate as another computer, such as the 1401, by adding to but without redesigning the system. This ROS concept can be combined with software

to offer almost any performance from pure simulation (no ROS) to maximum performance (no software). When this can be done with performance equivalent to the original system, it greatly simplifies the programming conversion problem by offering essentially two computers in one. Note that we have various technologies for the read-only memory control, including card capacitor, balanced capacitor, and the transformer approach. Certain choices were made for initial implementation, but it should be made clear that the choices, especially in the slow speed unit are not critical and that more than one technique could be used to give the desired performance.

The concept of architectural compatibility was carried a step further in the input/output area by the decision to attach the various units through the same electrical interface. The major reasons for this decision were the added flexibility to the IBM customer, and the greatly reduced number of different engineering and manufacturing efforts which would be involved in producing both the System/360 I/O channels and the many I/O units.

Other decisions concerned reliability and maintainability. The primary improvement in reliability involved the advantages of SLT over SMS and obtaining more performance from a given number of components by using high-speed circuits and storages. An objective in maintainability was to have hardware and software not only detect failures, but to localize them to small areas, such as five specific small cards. A programming system was created which takes the machine logic, analyzes it, and automatically produces a set of programs with the proper patterns and expected results for that specific logic. These fault locat-

ing programs (FLT's) are then capable of being entered to the appropriate computer (Model 50, 60/62, or 70) and executed with the assistance of special hardware. This hardware allows setting up the proper patterns in the various registers, advancing the clock a controlled number of cycles, logging these registers into main storage, and comparing the actual versus expected results. The programming system allows for updating these FLT's with engineering changes, and they offer a powerful diagnostic tool in localizing failures.

### SYSTEM/360 MODEL 30

Model 30, the smallest member of the System/360 line, was designed for the market area currently served by the IBM 1401, 1440, 1460, and 1620. The design objective was, of course, complete function and compatibility with other System/360 models. However, System/360 architecture includes 142 instructions, decimal, binary, and floating-point arithmetic, complete interruption facilities, overlapped channels, storage protection, and other features normally found in more expensive computers. This made maintenance of full compatibility,

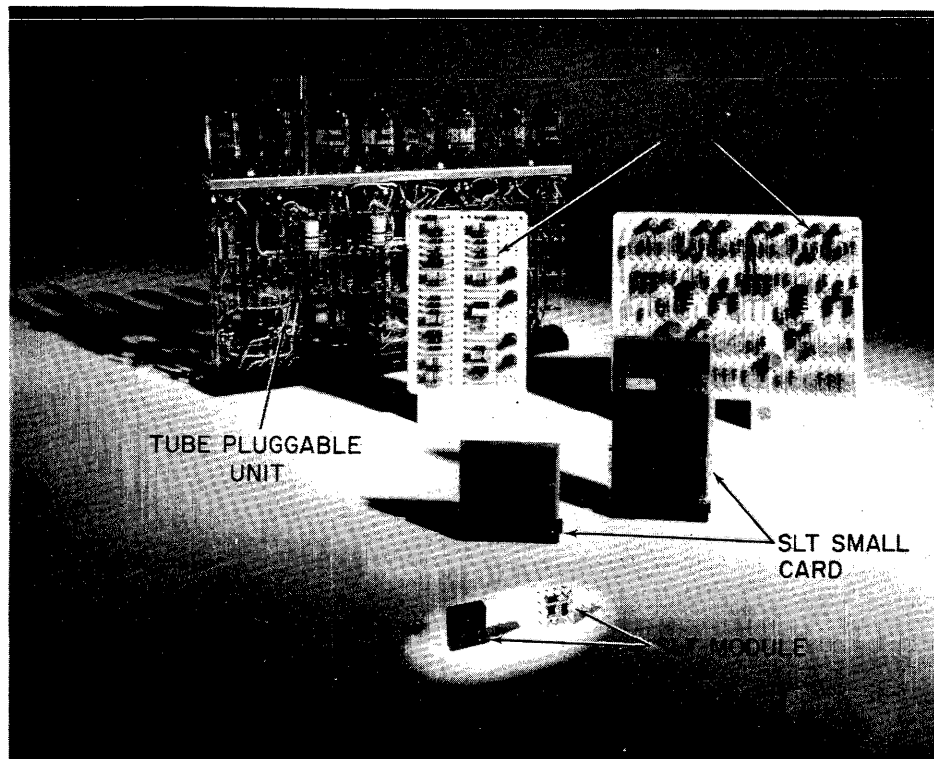


Figure 2. Solid Logic Technology (SLT).

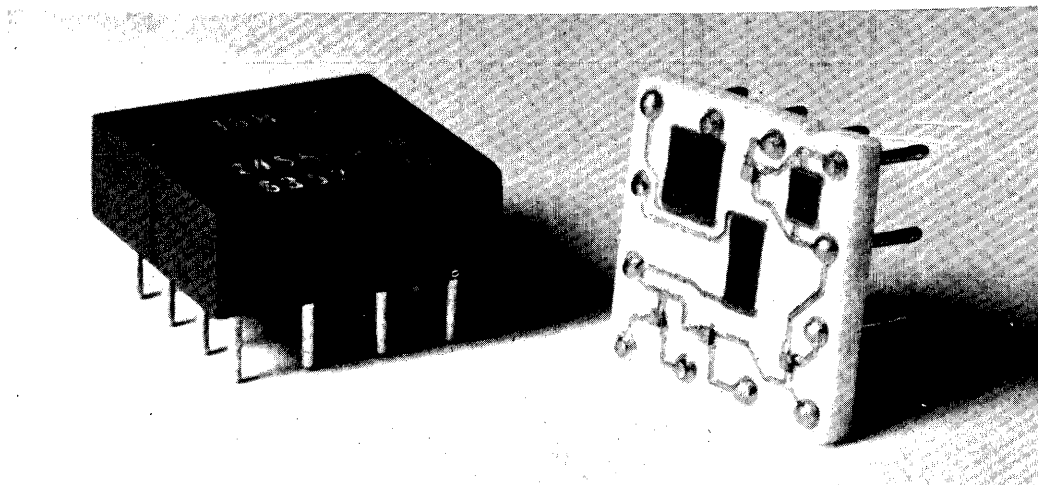


Figure 2b.

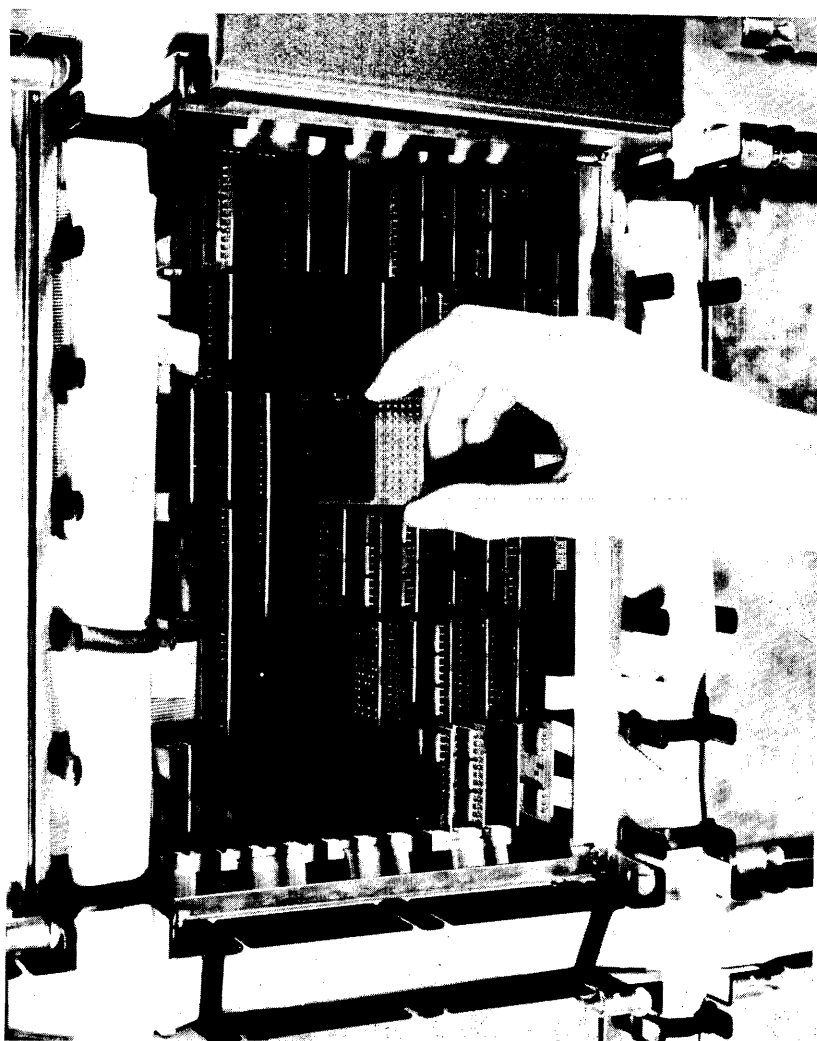


Figure 2c.

while achieving the required cost and performance objectives, a very difficult task.

Therefore, a number of different hardware configurations were examined. Data paths and storages from four to sixteen bits wide; table lookup addition and logical operations; registers built in SLT hardware, in main storage and in a separate high-speed storage, were studied in detail to arrive at a configuration that would meet the cost performance objectives. The availability of the 2- $\mu$ sec. storage at an acceptable cost was also established at this time and became an overwhelming factor in our selections.

The key engineering decisions which established Model 30 characteristics were:

1. Selection of an 8-bit wide, plus parity, 2- $\mu$ sec storage unit, 64K bytes maximum,
2. Use of 8-bit wide, plus parity, data path,
3. Use of 30-nsec SLT circuits,
4. Implementation of "local storage" in main storage,
5. Use of a 1- $\mu$ sec ROS for control,
6. Provide integrated, time-shared, multiplex and selector channels,
7. Provide for complete processor growth in in a single frame,
8. Utilize "byte" packaging to facilitate checking and fault location.

"Byte" packaging means placing all the circuits required for eight data bits plus parity on one pluggable small card. This concept, along with micro-program diagnostics, permits fault location to a resolution that averages five small cards.

A simplified data-flow diagram of the Model 30 processor is shown in Figure 3. The fourteen 8-bit registers, with the 8-bit ALU and 8-bit storage, provide all the functions for the CPU and multiplex channel. Additional registers or buffers are required for selector channels, direct control, interval timer, and storage-protection features.

Figure 4 is a map of local storage, which is actually an extension of the main storage unit. The first 256 bytes are utilized by the processor for general registers, floating-point registers, interim storage during multiply time shar-

ing, and other scratch-pad functions. The second 256 bytes are used by the multiplex channel for channel-control words. Additional control-word storage is available in larger storage sizes.

The multiplex channel time shares the processor registers by interrupting the microprogram, holding the return address, storing the registers in local storage, and then at completion of the I/O operation, restoring the registers to their original state. This is analogous to a macroprogram interrupt.

#### TIMING AND ROS CONTROL

The basic timing is established by the 1- $\mu$ sec read-only storage. The main storage provides the read-write cycle in 2  $\mu$ sec and a read-compute-write cycle in 3  $\mu$ sec. Within the 1- $\mu$ sec ROS cycle are four 250-nsec timing pulses.

The read only storage in the Model 30 is a card capacitor ROS containing a maximum of 8,000 words with 64 bits per word.

The capacitor ROS consists of a matrix of drive lines and sense lines with capacitors at the intersections where a one is required, and no capacitors at those intersections requiring a zero. The voltage change on a drive wire will cause capacitive current to flow in those sense lines which are coupled to that particular drive wire by a capacitor. In the card capacitor store, the 64 sense lines and one plate of each capacitor are printed on an epoxy glass board bonded to a sheet of dielectric material. The drive wires and the other plate of the capacitors are printed on mylar cards (program cards) the size of a standard IBM card (see Figure 5).

Each program card is punched with the information pattern specified by the micro-code and contains 12 ROS words. A capacitor plate is punched out at an intersection where a zero is to be stored; thus, an unpunched card will give all ones and punching a hole gives a zero at that bit in the word. Microprogram changes can be made by inserting new program cards.

#### MICROPROGRAM

A single microprogram instruction can initiate a storage operation, gate operands to the

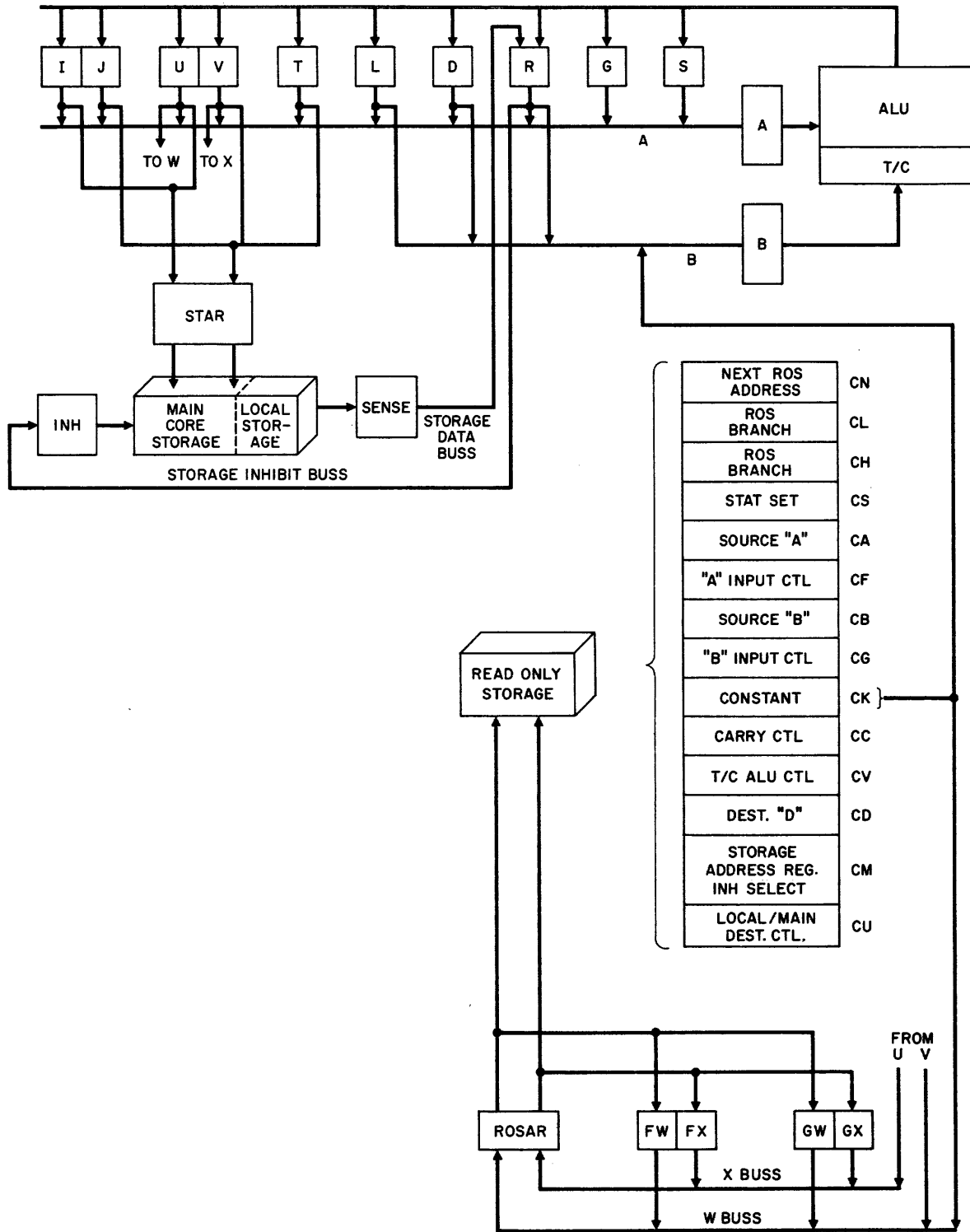


Figure 3. System/360 Model 30 Data Flow.

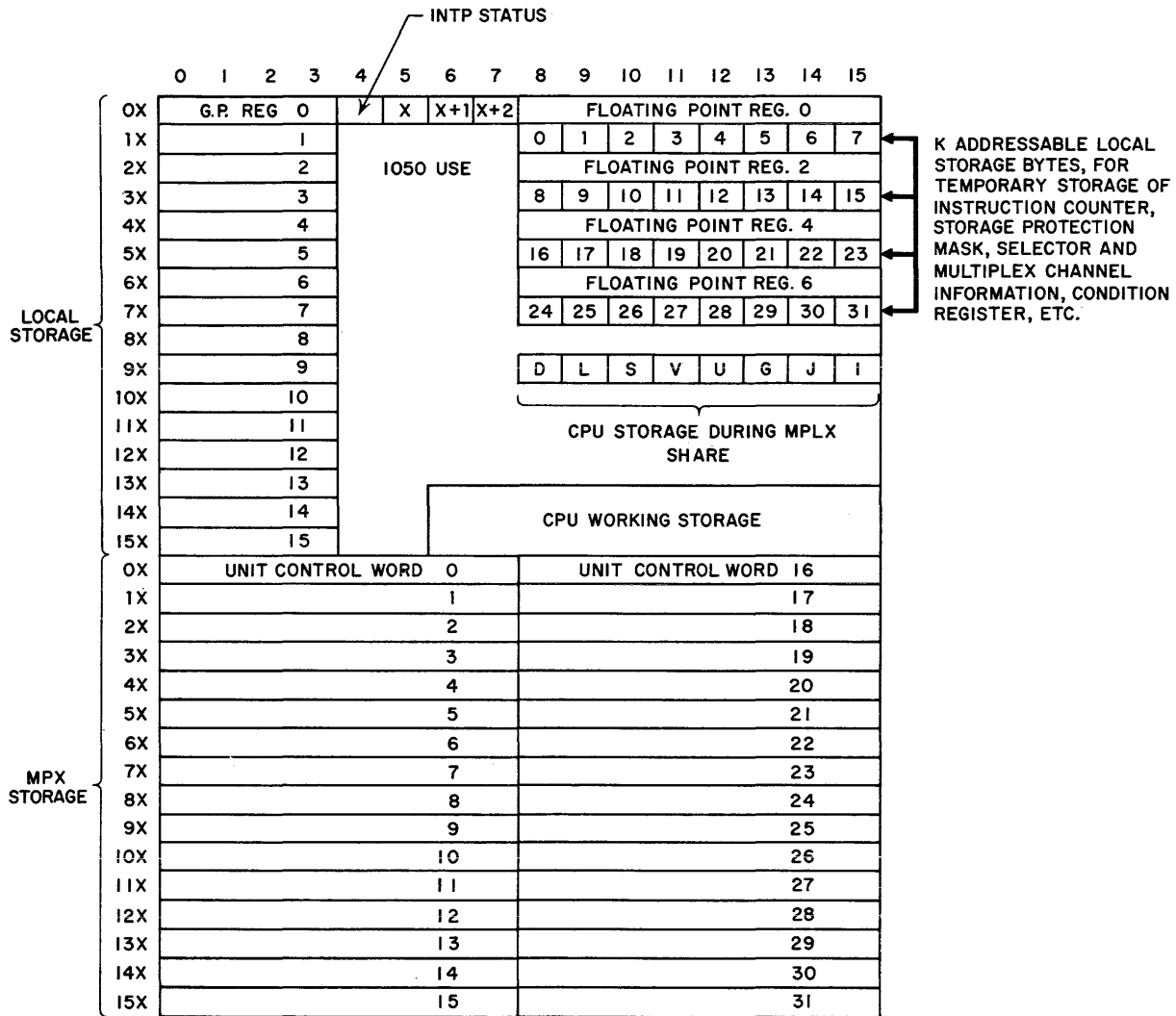


Figure 4. Model 30 Local Storage.

ALU input registers, select the ALU function, store the result in the destination register, and determine the next micro-word to read from read-only storage. Each ROS word is decoded to operate the gates and control points in that system. A brief description of the branch, function and storage-control fields in each ROS word follows.

*Branch Control*

The branch control fields provide the address of the next ROS word to be executed. A ROS address is a 13-bit binary number. Normally the branch control group provides only 8 bits (leaving 5 bits unchanged) of

next address information. Of these 8 bits, the low-order 2 are called "branch" bits and the remaining 6 are called "next address" bits. The 6 "next address" bits are specified directly in a 6-bit field. The two "branch" bits are specified by two 4-bit fields. These two fields are decoded and used in masking and extracting machine conditions and status conditions contained in data-flow registers G and S.

Another 4-bit branch control group provides the function of setting several variables to desired values for later use in microprogram branching.

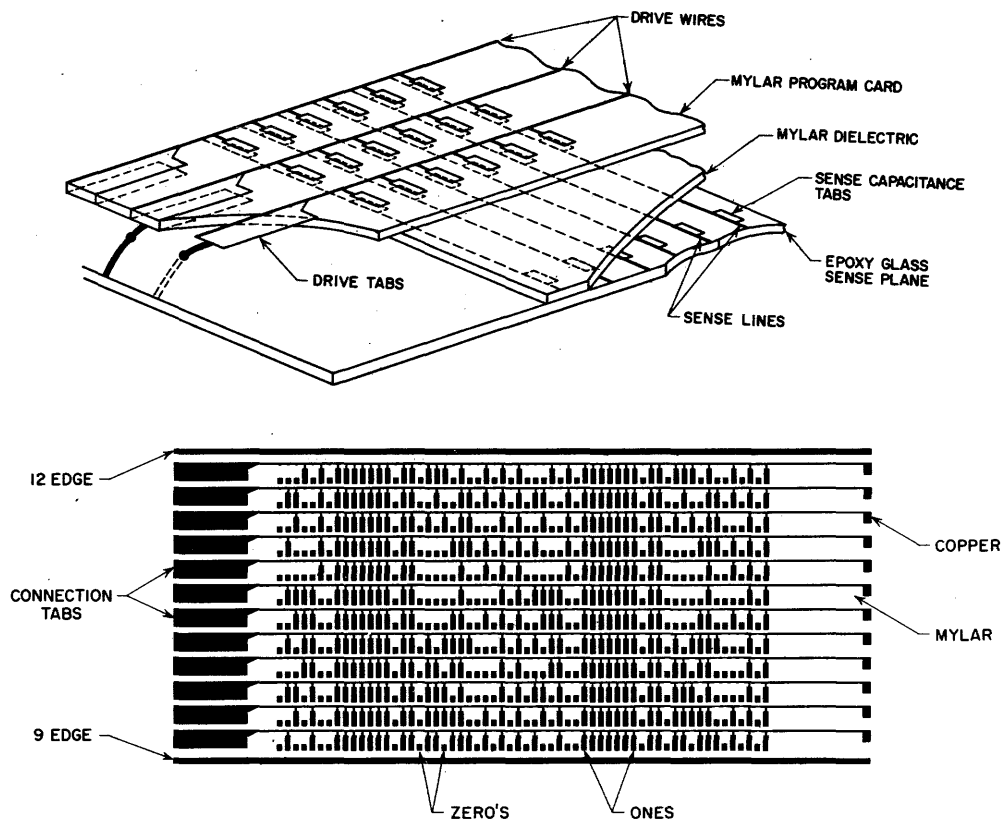


Figure 5. Model 30 Card Capacitor Read-Only Storage.

In summary, every ROS word provides a branching ability. The branch can be 4-way, 2-way, or 1-way (simple next address). A partial next address is normally used, but provision is made for obtaining a full 13-bit next address when required. The total length of the branch control group is 18 bits, plus 1 parity bit.

#### *Function Control*

The function control group is subdivided into four fields: source A, source B, operation, and destination.

1. Source A (CA)

This 4-bit field selects one of the 10 hardware registers to be gated to the A input of the ALU.

2. (CF)

The 8 data bits from a register can be presented to the A input "straight" or they can be presented "crossed." The term "crossed" means that the high-

order four bits of the source register enter the low-order four bits of the ALU, and the low-order four bits of the source register enter the high-order four bits of the ALU. The A input can further be controlled by presenting all eight bits, the low-order four only, the high-order four only, or none, to the ALU.

3. Source B (CB)

This 3-bit field selects one of three registers to be presented to the B input of the ALU. The B input is the "true/complement" input and has HI/LO controls but no straight/crossed controls.

4. (CG)

This field controls the gating of the B input to the ALU. That is, the low-order four bits only, the high-order four bits only, all eight bits, or none of the eight bits of B may be presented to the ALU.

5. Constant Generator (CK)

This field is gated to the B buss, main core STAR and ROSAR, thus providing



a source for constants, mask configurations, and address constants.

6. Carry (CC)

This 3-bit field controls carry in, AND, OR, EXCLUSIVE OR functions and permits the setting of carry out into the carry latch, if desired.

7. True / Complement & Binary / Decimal Control (CV)

This 2-bit field controls the true/complement entry of the B input to the ALU, also whether the operation is decimal or binary.

8. Destination (CD)

This 4-bit field selects one of the 10 hardware registers to receive the output of the ALU. A given register may be used both as a source and as the destination during a single ROS cycle.

In summary, the Operation group specifies one of ADD binary, ADD decimal, AND, OR, or EXCLUSIVE OR. It also specifies true or complement; 0 or 1 carry input; save or ignore resulting carry; use true/complement latch; and use carry latch.

### Storage Controls

#### (CM) (CU)

These two fields control core storage operation. Either main storage, local storage, or MPX (for I/O) storage can be addressed for storage read-write calls; five values of CM are used to specify the address register to be gated to STAR.

An example of the sequence of ROS control, Figure 6, shows an ADD cycle using a simplified data flow.

*Step 1*—As the routine is entered, the contents of UV are gated to the STAR, a read call is issued to main storage, and register V is decremented by 1.

*Step 2*—The A-field data is regenerated in storage and the A-field data byte is transferred from register R to D.

*Step 3*—The contents of IJ are gated to STAR, a read call is issued and J (lower —4 bits) are put in Z.

*Step 4*—Z is tested for 0 to set up the branch condition at the next step, the B-field data byte

is read out (R) to the adder, as are the D-register contents (A-field data) and the carry from a previous cycle. The output (Z) is gated into R.

*Step 5*—If the zero test of A in Step 4 is true, a write into the B field is performed (the address is still in STAR), J is decremented by 1 and the routine is repeated. If the zero test of Z in step 4 is false, then a branch is made to the write call and the routine is exited.

As an example of Model 30 microcoding efficiency, the execute portion of a fixed-point binary add uses approximately 20 words. However, the add can be combined with 13 additional operation-code executions, such as subtract, AND, OR, EXCLUSIVE OR, etc., using a total of 45 words. A one-half word multiply giving a full-word product requires about 95 words. The total floating-point feature, which utilizes the fixed-point microprograms, requires approximately 500 words. It should be recognized that the microprogrammer has the choice of optimizing for minimum words or maximum performance.

### IBM 1401/40/60 COMPATIBILITY FEATURES

It was a market requirement that Model 30 execute the IBM 1401-40/60 instructions directly. Further, it was desired to provide these features without disturbing the Model 30 design, which was optimized for System/360 requirements. As a result, these features are provided by an addition of only four circuit cards plus extensive microprograms.

The general approach utilizes the following:

1. System/360 input-output devices,
2. Conversion tables in local storage,
3. Microprogram decoding and execution of the instructions directly.

The internal performance is several times the 1401, based on a typical mix of instructions found in 1401 programs. For individual instructions, however, the speed ratio varies widely.

#### Method

The internal code used in Model 30 for the compatibility feature is EBCDIC and, further, Model 30 has a binary-addressed storage. Thus,

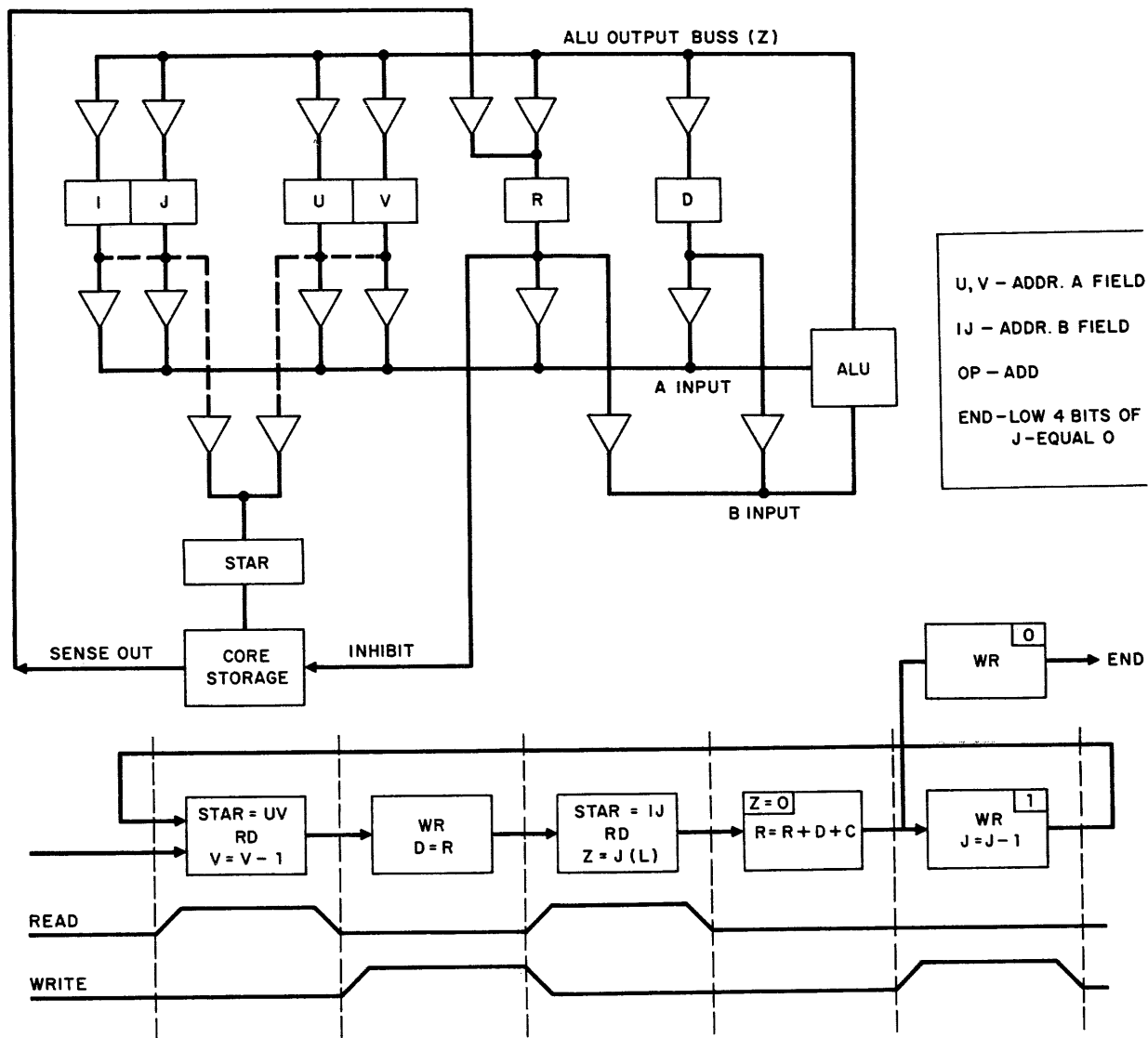


Figure 6. Model 30 Read-Only Storage ADD Cycle.

a certain amount of translation of character codes and conversion of numbers from decimal to binary radix, and back again, takes place during processing. These conversions and translations are accomplished by table look-up, using the tables in local storage. These tables are read into storage as part of the special load procedure required prior to execution of a 1401 program. The table used to translate the EBCDIC to BCD requires 128 characters. It is used during the execution of 1401 instructions such as bit test, move zone and move numeric, which depend on the actual bit coding.

A 72-byte table is required to hold the constants used to convert the 1401 decimal ad-

dresses to binary addresses and the converse. This conversion takes place at execute time, hence the programs can operate correctly no matter what methods are used in the 1401 program to generate or modify addresses. Each conversion requires from 6 to 12 microseconds depending on the value of the address.

Other functions which utilize tables are op decode and I/O device address assignment. Additional areas in local storage are used for hardware register back-up, sense switches, system specification, and working area. While most of the available 512 bytes of local storage are used, no main storage is used. Hence, an 8K 1401 program will run on an 8K Model 30.

Each input or output device type has an individual microprogram routine. Most devices are attached to the Model 30 multiplex channel. Magnetic tapes are also available on selector channels. The throughput for the compatibility feature requires an analysis of the particular I/O devices used and the particular program being run. However, in every practical case it will exceed the system being simulated.

## SUMMARY

The Model 30 effort has helped to prove two points:

1. Small systems, with the extensive function and facility of the largest systems, are practical.
2. The microprogram control is sufficiently general that a good system design can be used to simulate a wide variety of architecture.

## SYSTEM/360 MODEL 40

The performance of Model 40 is approximately three times that of Model 30. To attain this performance at minimum cost, four major design decisions were important.

1. The adoption of a 16-bit-wide storage of 2.5  $\mu$ sec together with a basic 16-bit wide data flow, implemented in 30-nsec SLT circuits, provided an optimum configuration.
2. A 0.625  $\mu$ sec read-only storage as a means of control for all CPU functions was used because this technique offered significant advantages in cost, flexibility, and design freedom, compared with more orthodox control systems.
3. The inclusion of a 1.25  $\mu$  sec local storage of one hundred forty-four 21-bit words to provide general register storage. This resulted in a considerable reduction in accesses to main storage.
4. By using the local storage to preserve the contents of the CPU during channel operations, much of the CPU data flow can be used for channel functions, thus considerably reducing the cost of channels.

## MICROPROGRAMMING

Microcoded programs, physically residing in permanent form in a transformer read-only

storage (TROS), form the heart of the control section of the Model 40. To reduce the resulting physical changes associated with changes in the microprograms, the microprogram design is automated and debugged before actual physical implementation by means of an IBM 7090 programming system, the Control Automation System (CAS). CAS is utilized not only by Model 40, but by all the System/360 models using ROS control.

The basic input to the system is a logic sketch page produced by the microprogrammer. The separate micro-instructions are written on this page in a formal language, TACT. When this initial writing phase is complete, the page is transcribed into punched cards. The control program for the 7090 is directly derived from the Model 40 control signal specification and acts as a set of inviolable rules. Within this framework, and using associated established microprograms for reference where necessary, the 7090 simulates the Model 40 and attempts to run the microprogram using submitted data. Errors or violations are detected, stop the program, and cause a diagnostic analysis routine to be entered.

Facilities are available for various printouts to provide for analysis and subsequent correction.

The debugged microprogram, in the form of magnetic tape, is submitted to assignment checking. This operational phase checks the manual assignment of the absolute address given to each ROS word, and produces listings giving the absolute address and binary bit pattern of each assigned ROS word. Two decks of cards are also produced and used in the production and testing of the read-only storage.

An output of the CAS program is a fully-checked and redrawn version of the original logical sketch page. If microprograms are subsequently updated, a revised CAS page is automatically printed on receipt of change.

## TROS

The finally-debugged microprogram is translated into a series of micro-instructions, held in read-only storage.

In Model 40, this takes the form of a transformer read-only storage—TROS. TROS is

made up of 16 modules, each containing 256 ROS words (micro-instructions) to make a total capacity of 4096 words. Each module is made up of 128 tapes, each tape containing two words. The word tape carries two ladder networks, each of which, after modification, holds the bit pattern derived from a specific micro-instruction.

The tapes are stacked in a module, as shown in Figure 7, with transformers inserted through the prepared holes in the tapes. These 54 transformers are in the form of U and I cores. The I cores carry the sense windings. Each stage of the ladder network corresponds to one bit position of the ROS word. Whether the bit is a 1 or a 0 is determined by breaking the current path on one side of the ladder with a punched hole through the printed wiring, so that the current then either passes through the core for a 1 or bypasses it for a 0.

Signals are taken from the tapes to the sense amplifiers, the outputs of which are used to set 54 latches.

The total cycle time of the TROS and the basic cycle time of the CPU are both 625 nsec.

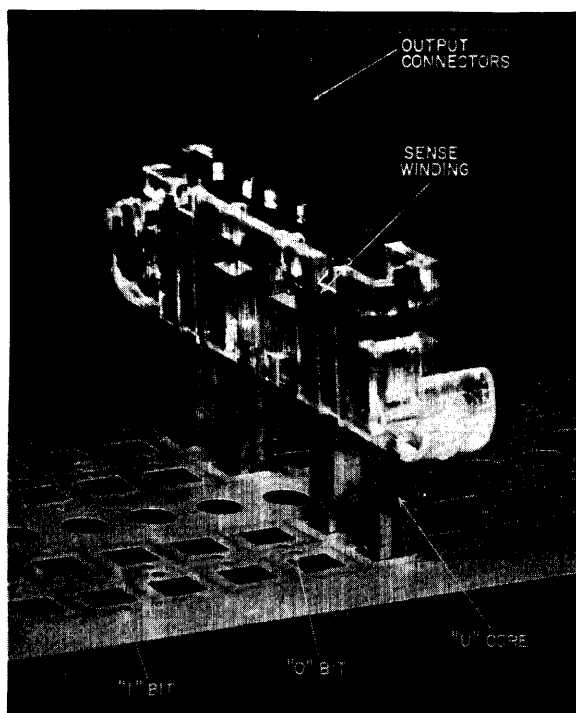


Figure 7. System/360 Model 40 Transformer Read-Only Storage.

Normally, there are four 625-nsec TROS cycles for each 2.5- $\mu$  sec main storage cycle.

It is possible, by the inclusion of a feature which adds additional ROS modules, to simulate other equipment, such as the 1401 or the 1410. This enables programs written for these machines to be run on the Model 40. Model 40 implements these features by microprogramming and conventional programming; Model 30 utilizes microprogramming exclusively. For example: the input/output and edit commands in the 1401 simulated on the Model 40 are executed with System/360 programming, while most of the remainder of the 1401 instructions are microcoded.

The primary reason for adopting this approach in the Model 40 is to reduce the added TROS requirements and the ensuing packaging problems and costs.

#### DATA FLOW

Figure 8 is a schematic representation of the Model 40 data flow.

The data flow may be divided into sections characterized by their data-handling capabilities:

1. one-byte arithmetic handling (8 bits plus parity),
2. one-byte local storage addressing,
3. two-byte storage addressing and data input/output referencing,
4. one-byte service for the channel data and one-byte service for flags related to the channel interface,
5. two-byte data flow to and from local storage.

#### *One-Byte Data Flow*

One-byte arithmetic handling is performed by the arithmetic and logical unit (ALU). The ALU is a one-byte-wide adder/subtractor which operates in either decimal or hexadecimal mode. It is capable of producing both arithmetical and logical combinations of the input data streams and is checked by means of two-wire logic, where one true and one complement signal is expected on each pair of wires.

Data bytes are fed to the P and Q busses from the associated registers or from the emit

field of the current micro-instruction. The data are then manipulated by the ALU in accordance with the content of the ROS control word. Several instructions may have common micro-program subroutines in which the difference lies only in the ALU function. One of 16 different ALU functions is preset by a 4-bit field in a ROS control word executed before branching to the common subroutines.

*Two-Byte Data Flow*

Data transfers between local storage, channel registers, CPU registers and main storage are carried out in two-byte steps.

LOCAL STORAGE

This is a small, high-speed storage which provides registers for fixed and floating-point operations, channel operations, dumping of CPU working register contents, interrupts, and for general working areas. Only the fixed and floating-point locations are addressable by the main program.

The ferrite local store contains 144 locations allocated as shown in Figure 9. Each location is 21 bits long. Addressing is completely random and the unit may be split-cycled with read-or-write operations in any sequence. A read-

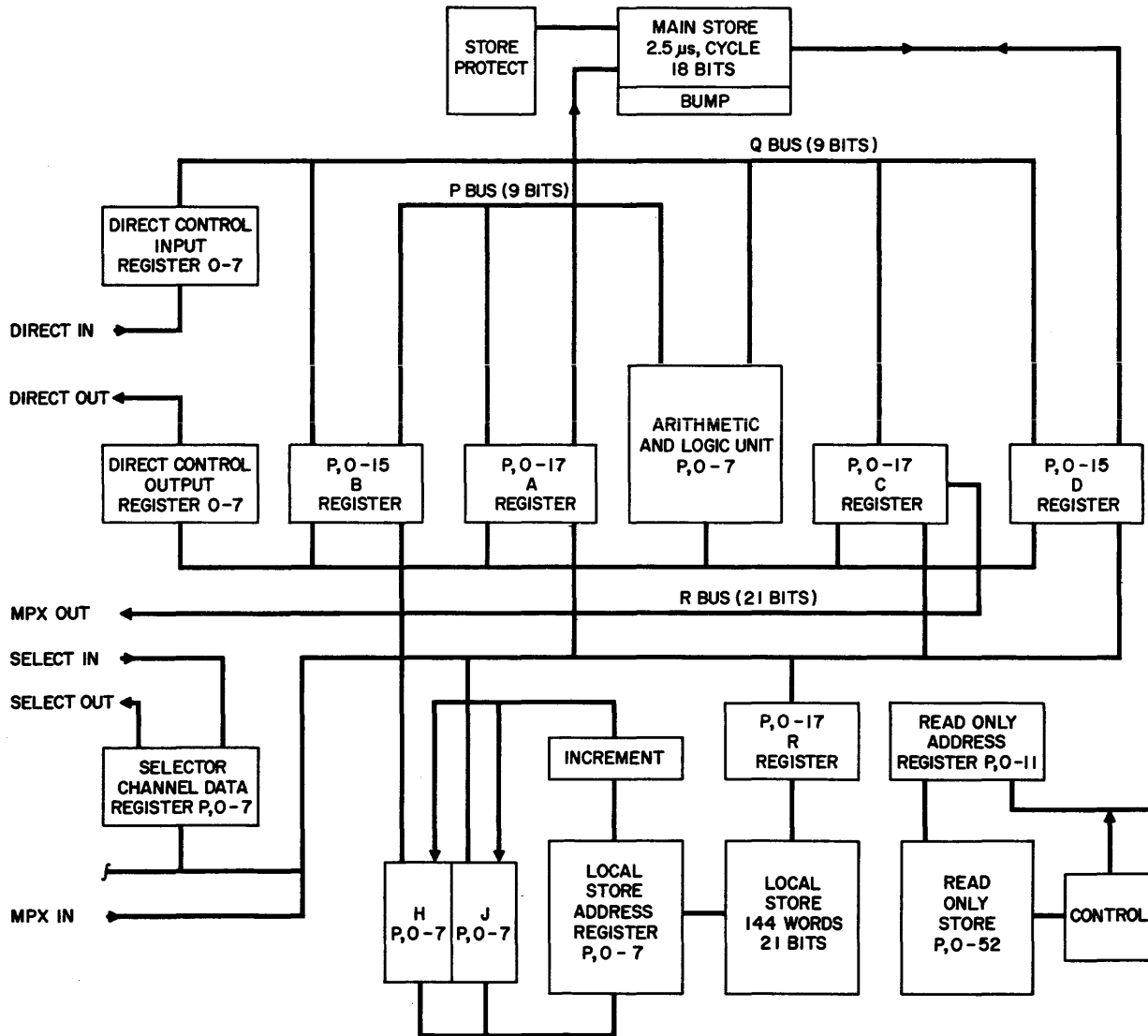


Figure 8. Model 40 Data Flow.

000	WORK AREA	064	WORK AREA IN UNDEFINED STATE	128		192	FLOATING POINT REGISTERS
		066	PROGRAM STATUS WORD				
		067					
009		WORK AREA AND LOG OUT AREA	071			START I/O SWITCH	
	072						
015	073		1ST LEVEL DUMP AREA				
016		079				207	
031		080					
032	SELECTOR CHANNEL 1 UNIT CONTROL WORD	095	SELECTOR CHANNEL 1 BUFFER			224	FIXED POINT
037	MULTIPLEX CHANNEL WORD AREA	096					
038							
041	INTERRUPT BUFFER						
042	UNASSIGNED						
043							
047		111					
048	SELECTOR CHANNEL 2 UNIT CONTROL WORD	112	SELECTOR CHANNEL 2 BUFFER				REGISTERS
053	UNASSIGNED						
054							
056	2ND LEVEL DUMP AREA						
057							
063		127		191		255	

ADDRESSES IN DECIMAL

Figure 9. Model 40 Local Storage.

or-write cycle requires 625 nsec. A complete read-and-write cycle therefore takes 1.25  $\mu$  sec.

One example of use of the local store is the double-dump routine executed under certain types of channel operation. If the machine is currently in CPU mode and a multiplex channel interrupt occurs, all data relevant to the current CPU operation are dumped in the local storage first-level dump area. If, subsequently, a selector channel interrupt occurs, all data relative to the multiplex service are dumped, and the selector channel is serviced. When all selector channel operations are complete the multiplex data are restored and multiplex servicing is continued. Similarly, when multiplex servicing is complete, CPU data are restored and CPU operation is resumed.

## MAIN STORAGE

The main storage array (2.5- $\mu$  sec cycle, 1.25- $\mu$  sec access) of the machine is divided into two logical sections. These are the true main storage, and a special area of 256-2048 bytes, called the bump storage. The bump storage is used to hold channel control words used in multiplex channel operations and is accessible by the microprogram only.

## CHANNELS

### *Multiplex Channel*

The multiplex channel is an extension of the CPU in the sense that the regular CPU data flow and microprogram are used for all data transfers. This channel, which allows a number (maximum of 128) of relatively low-speed units to be operating simultaneously, normally scans all attached control units continuously. When a device reaches the point where it needs to send or receive a byte of data, its control unit intercepts the first available scanning signal and transmits the unit address to the CPU. The CPU data flow is then cleared and retained in the local store using the dump routine. After the byte transfer has been completed, the control unit and device disconnect from the channel, permitting scanning of other devices to be resumed, and the CPU processing to continue.

### *Selector Channels*

Two types of selector channels are available on Model 40, the A channel and the B channel.

They differ in that the CPU interference caused by I/O operations on the B channel is approximately one third of that caused by the A channel.

*The A channel* time-shares the CPU data flow and microprogram to a high degree. Data bytes are never transferred directly between main storage and the interface busses, but move via a 16-byte buffer in the Model 40 local storage. The transfer between interface and buffer is conducted serially, by byte, at the rate dictated by the I/O device. Each transfer causes the microprogram to hesitate one 625-nsec cycle. When the 16-byte buffer is half full, the channel requests the use of the microprogram and CPU data flow. Up to 12 microprogram cycles are required to preserve the current contents of the data-flow registers and to load the control word. The 8-16 bytes in the buffer are transferred 2 bytes per storage cycle to main storage as a block, at a rate of 2 bytes every 2.5 microseconds.

*The B Channel* is more conventional; it uses SLT hardware and does not use the local storage as a buffer. Data is transferred to main storage as soon as two bytes are accumulated. Interference is basically constant at 1.25  $\mu$ sec per byte.

## SYSTEM MAINTAINABILITY

One of the more unique features of Model 40 maintenance hardware is the use of the read-only storage as a source of diagnostic routines. One module of the TROS contains a complete set of tests to validate the CPU, local and main storages. These tests are automatically applied each time the system reset is operated to ensure an operational machine.

## SYSTEM/360 MODEL 50

The performance range of System/360 Model 50 is approximately ten times the Model 30.

A review of the following key engineering decisions will highlight the distinguishing characteristics and engineering achievements of Model 50.

1. The 30-nsec family of SLT circuits is used.

2. A small,  $0.5 \mu\text{sec}$ , local core storage contains the general purpose and floating-point registers.
3. A relatively low-cost  $2.0 \mu\text{sec}$ . main storage is used.
4. The data paths, local storage and main storage are each 32 bits wide (plus 4 parity bits).
5. A  $0.5 \mu\text{sec}$  read-only storage provides sequence control throughout.
6. Both selector and multiplexor type channels are provided to cover a wide performance range.
7. The CPU and channels utilize common hardware, yet maintain substantially overlapped operation.
8. The CPU, channels, and storage are packaged in a unified structure.

### CENTRAL PROCESSING UNIT

The first four decisions are highly interdependent and were reached somewhat simultaneously to produce a system in the right cost and performance range, utilizing components that would be available at the right time and that would fit together in a good physical and speed relationship. These components were selected on the basis of good performance per-unit-cost and a balanced design, rather than for performance alone.

The choice of the 30-nsec family of circuits allowed an internal clock cycle from register through adder and back into register of 500 nsec. This family of circuits provides the combination of compact packaging, good fan in-fan out ratios, and low power dissipation, with a nominal delay per stage of 30 nsec.

A typical 30-nanosecond SLT module is illustrated in Figure 10, together with its circuit diagram. Such a module has a power dissipation of approximately 30 milliwatts and allows a fan-out factor of 5 and a fan-in to the OR of 5. The fan-in to the AND is limited only by packaging considerations.

Figure 11 illustrates the basic Model 50 data flow. The main adder path, the working reg-

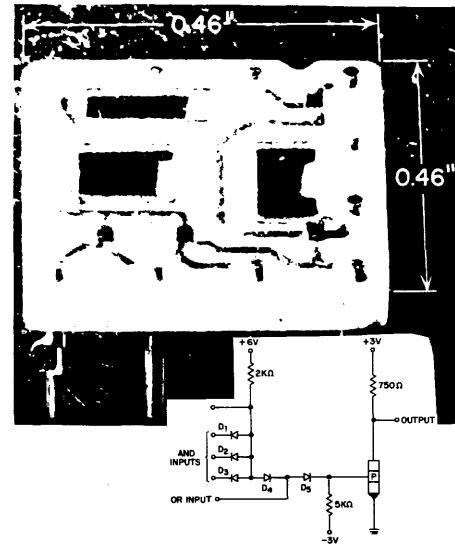


Figure 10. Solid Logic Technology 30-nsec AOI Circuit.

isters, the main storage, and the local storage are all 32-bits wide plus 4 parity bits. An auxiliary 8-bit data path through a logical processing unit, called the mover, is provided for processing variable field length information. This path allows a byte to be selected from each of two working registers under control of two byte counters. The two bytes can then be combined in a variety of logical functions and returned to one of the working registers. With the mover, decimal operands are first aligned in the working registers, then processed arithmetically 32 bits at a time using the main adder.

The main  $2\text{-}\mu\text{sec}$  storage unit for the Mod 50 is approximately  $32'' \times 14'' \times 26''$ ; a reduction by one-third the physical size of the 7090 storage (Figure 12). It is available in capacities of 64K, 128K, and 256K bytes (8 bits plus parity). Bump storage which is part of main storage is used to hold channel control words for multiplex channel operations. Bump storage area of 1024 to 4096 bytes is accessible only by microprogram control and not by the problem program. The use of a combined sense-inhibit line allowed a three-wire ferrite core plane which can be machine wired.

The local storage contains 64 thirty-six bit words (32 plus parity) and has a read-write



cycle time of 500 nsec. This ferrite core storage unit provides working locations for the CPU and channels, as well as the general and floating point registers. Regeneration from either the L or R register allows a swap of information between the CPU and local storage on a single cycle.

READ-ONLY STORAGE

The decision to use a read-only storage for sequence control produced a great unifying and

organizing influence on the design procedure. It not only forced a centralization of all controls, but also forced an early definition of all gate signals thereby allowing the design to proceed independently in several areas of the machine. Additional benefits resulted from the use of the Control Automation System (CAS). This system not only provides the necessary record-keeping and generation of manufacturing information for the read-only storage, but also provides documentation of the instruction

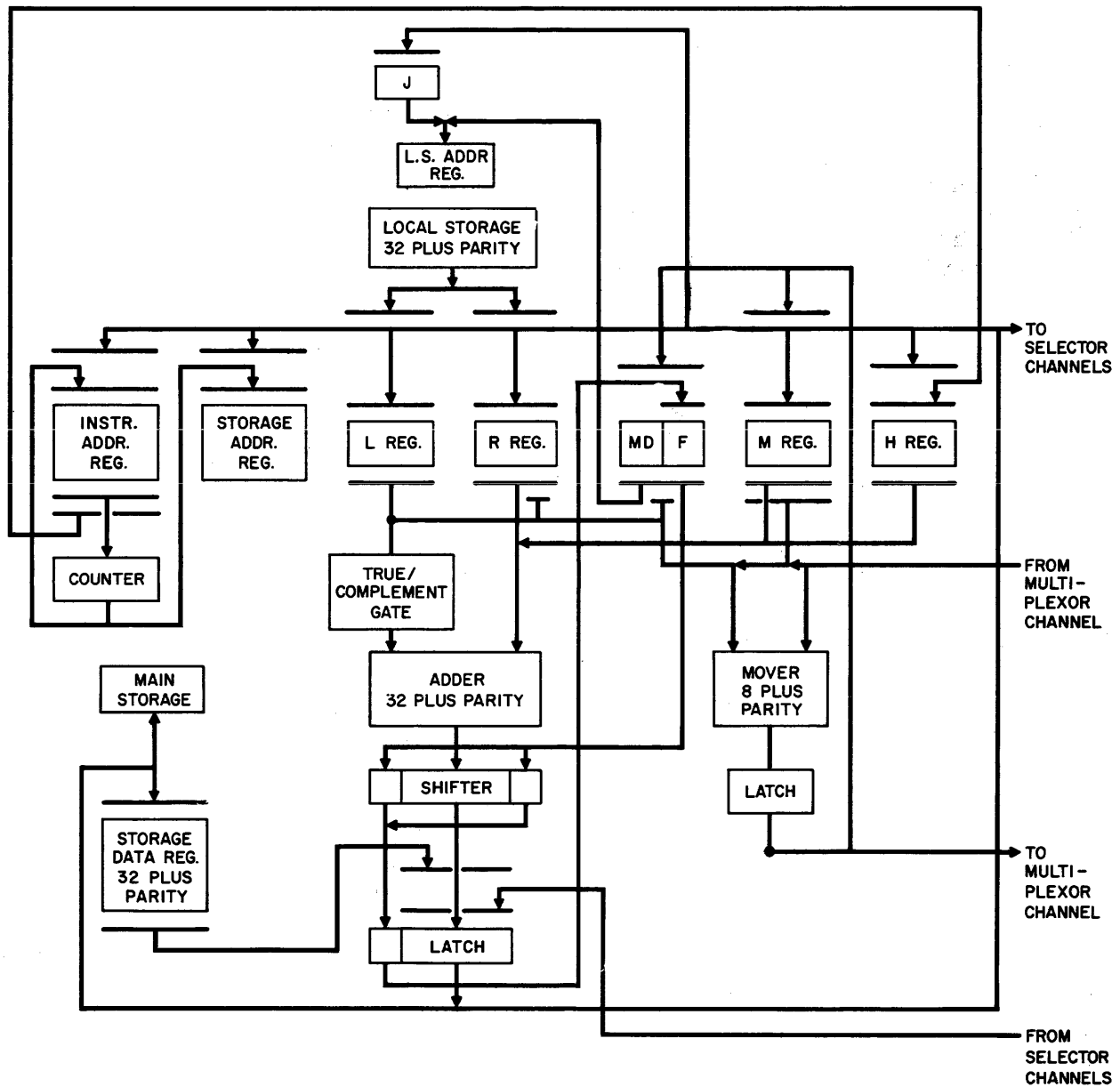


Figure 11. System/360 Model 50 Data Flow.

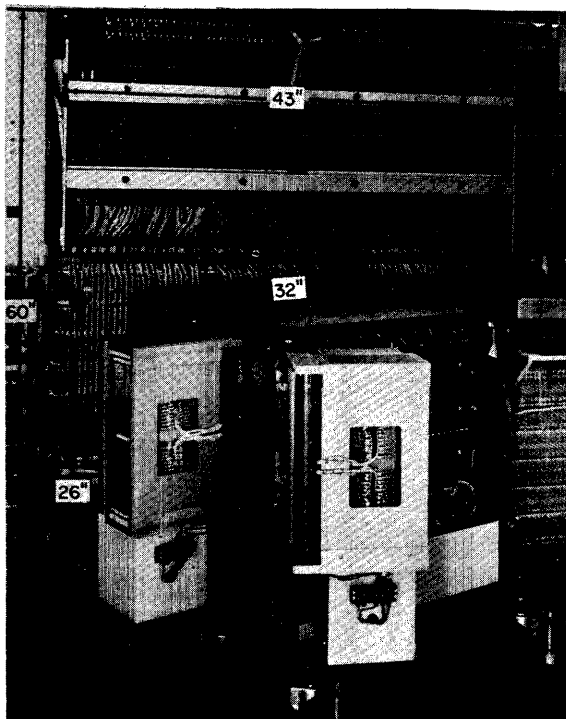


Figure 12. Physical Comparison Model 50 and IBM 7090.

sequences and allows complete simulation of these sequences before producing hardware.

The balanced capacitor technology is used in the read-only storage unit of the Model 50. A bit plate contains the information content of the storage in the form of tiny tabs attached to a long electrical drive line (see Figure 13). These are etched on a glass epoxy plate by a process similar to that from which printed circuit cards are manufactured. This bit plate is covered by a sheet of 1-mil mylar which forms the dielectric of the capacitor. The bit-plate tab forms one plate of this capacitor and a sense line running orthogonally to the drive line forms the other plate.

The sense lines are also etched lines. A pair of these lines form inputs to the base and emitter of a differential amplifier at one end and are terminated to ground at the other end. One sense plate contains 400 parallel sense lines.

The bit plates lie over the sense plates and are separated by the 1-mil mylar, with the

drive lines running vertically and the sense lines horizontally. Approximately a 5-inch pound torque is applied to a system of pressure pads to maintain constant pressure between the two plates.

Thus, switching on the array driver provides a change in voltage that is capacitively coupled from the drive line to the sense line. For impedance-matching purposes in the sensing circuits, a balancing line is used in conjunction with each drive line. The appropriate location of the bit tabs determines whether the signal received by the sense amplifier is a 1 or a 0.

The read-only storage of the Model 50 contains 2816 words of 90 bits each. It has a cycle time of 500 nsec and an access time of 200 nsec. To allow the result of one cycle to immediately influence the choice of the next, two words are read from storage simultaneously, and a choice between them is made on the basis of the result of the first cycle. The chosen word then controls the second cycle. This technique allows the same speed to be maintained as if sequential logic circuit controls were used.

## CHANNELS

A wide range of channel performance is provided in the Model 50 by the inclusion of two quite different channel designs. Both types of channels allow a substantial overlap with CPU operations.

The multiplexor channel provides concurrent operation of multiple low to medium speed devices. It makes extensive use of CPU hardware and contains relatively little hardware of its own. (Example: Local storage and some CPU registers are used as working locations.) The control words for each operating device are held in an extension to main storage called bump storage. As each byte is handled, the channel takes control of the CPU and obtains the required control word from bump storage. The CPU registers required for the operation are dumped into local store. The multiplexor channel can also operate with a single higher speed device in a "burst" mode. In this mode the control word is held in local storage to speed the operation, but bytes are still handled one at a time.

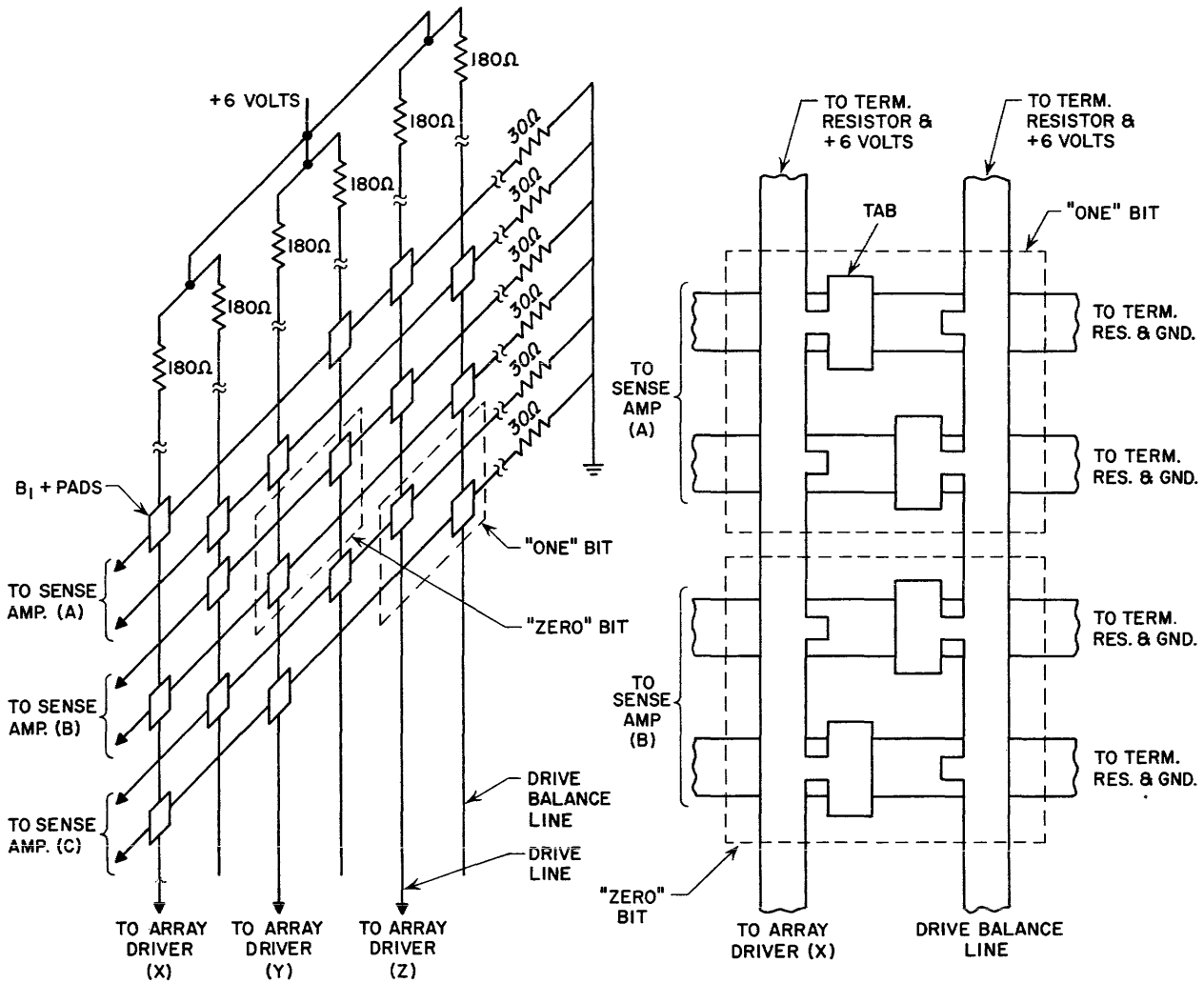
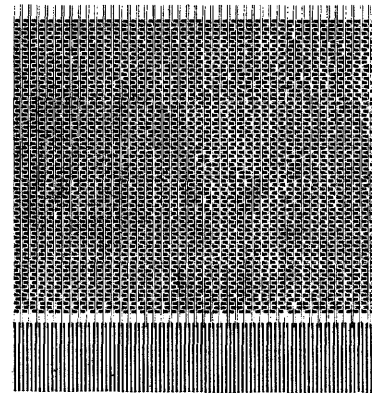


Figure 13. Model 40 Capacitor Read-Only Storage.

High speed I/O devices use the selector channels (up to *three* are attachable) which can handle 8-bit data up to an 800 KC byte rate. These channels contain sufficient hardware to assemble a full word of data before requiring the main storage or CPU facilities. Control-word information is retained in hardware instead of local storage. CPU facilities are used only when transferring a word to storage or chaining between I/O commands, resulting in a much higher maximum data rate than the multiplexor channel.

An additional selector channel is available which operates in a lockout fashion and is capable of operating at a data rate up to 1.3 mc.



### MAINTENANCE

Four decisions stand out in the maintenance area. They are:

1. The decision to include a hardware Log In—Log Out system for the execution of fault location tests (FLT's). An integrated approach (using the read-only storage to control FLT sequencing and log paths, and using existing hardware where feasible) reduced the cost from being excessive to being defensible, and in addition provided better fault resolution than conventional diagnostics. The FLT's can provide fault localization to within a few small cards, on the average, and have the additional advantage of being automatically produced and updated. The Log Out system is also used with the error checking circuits, to provide a complete snapshot of internal status at time of error.
2. Use of a Progressive Scan technique provides a reasonable means of running fault—location tests on channel hardware; with resultant improved fault resolution, thus the entire processor can be examined with high resolution, non-functional test.
3. The DIAGNOSE instruction allows the initiation of any micro instructions, in any order. This permits integration of the control of various maintenance techniques into the read-only storage. This instruction gives the diagnostic programmer a power tool.
4. The decision to allow for single-man servicing. This is made possible by bringing all controls and indicators together on a single system control panel which is usable in two positions; the normal operating position, and swung left 180° so that it faces the principle servicing area (CPU logic, main storage, and channel hardware). See Figure 14. The small logic cards, which are the principle replaceable item, are easily accessible, with the majority located on the outer sides of the gates.

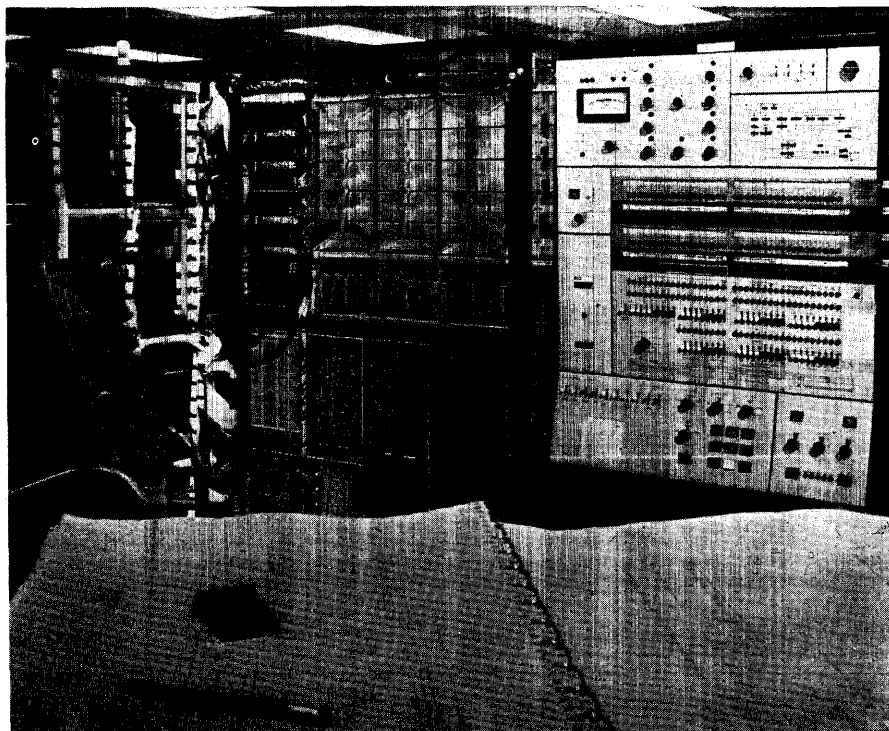


Figure 14. Model 50 System Control Panel (Open Position).

## SYSTEM/360 MODELS 60/62

Models 60/62 are large-scale processors with performances that are approximately 20 and 30 times that of Model 30.

Basic design considerations are:

1. Main storage speeds of 2  $\mu$ sec and 1  $\mu$ sec.
2. High-speed circuit family with nominal delay of 10 nsec per logical block,
3. Local storage in transistors, with 125-nsec access and non-destructive read,
4. Read-only storage control of CPU functions at a 250-nsec cycle,
5. 64-bit-wide storage, plus 8 parity bits; 64-bit data flow, plus 8 parity bits.

Models 60/62 attach stand-alone storage and channel units compared to the integrated designs of the smaller models in System/360. The entire instruction set is standard. Model 60 is equipped with a 2- $\mu$ sec main storage packaged in two separate frames. Address interleaving between the frames increases the effective speed of storage. The combined capacity of the storage frames varies and is available in 128K, 256K, or 512K byte sizes. Input-output control is provided through a channel capable of handling 8-bit data at a 1.3-mc byte rate. Up to 6 channels are attachable and capable of operating simultaneously. Information is passed between storage and channel on a 72-bit, double-word basis, minimizing interference with operating programs. High-speed I/O devices, tapes, disks, and drums are primarily intended for direct attachment to the channel. Slower speed units, terminals, peripheral readers, punches, and printers, attach to smaller System/360 Models which, in turn, may be connected to one of the Model 60 channels in a multiprocessing arrangement.

Model 62 differs from Model 60 only in the speed and configuration of the main storage. The 1- $\mu$ sec storage associated with this model is available in 256K-byte, self-contained units. A maximum of two of these units is directly attachable, providing 512K bytes of storage. The addressing is conventional and is accomplished without interleaving.

## TECHNOLOGIES UTILIZED

The foundation of the central processing unit design is a basic building block or module con-

taining four logical diodes with load resistors, one emitter follower and inverter transistor, plus three control diodes fabricated on a single substrate. This device houses the primary circuit used almost exclusively in Models 60/62.

Variations of diode and transistor arrangements exist within the same circuit family and these different module types equip the logician with a full array of AND-OR design elements. The basic block has a nominal delay of 10 nsec. Signal swings vary from +1 to +3 volts. The circuits display good noise-rejection characteristics in that worst-case simultaneous switching is tolerable under maximum loading situations of five-way AND's driving five-way OR's into a ten-load output. Circuit speed has purposely been compromised to improve driving and loading capability.

All the circuitry is contained on one module, with the exception of the two collector resistors which are located external to the module in a resistor pack. With the exception of the components outlined, the circuit configuration is essentially the same as the 30-nsec circuit block.

The local store is not required to accommodate integrated I/O channels, consequently a favorable trade-off is achieved by structuring it in transistor registers as opposed to ferrite cores. In addition to reduced cost faster speeds are obtained, since the store is not destructively read, so regeneration time is eliminated. The unit contains twenty-five 36-bit registers. Sixteen are general purpose registers, eight are floating-point registers and one is a working register used for variable field length control. Since one or more registers participate in the execution of each System/360 instruction, the accessibility and maneuverability of local storage contents is mandatory for good internal performance. Any of the registers can be easily read or modified in 125 nsec, a nice fit for the 250-nsec machine cycle. All CPU manipulations occur within the 250-nsec machine cycle.

A balanced capacitor read-only storage device, similar to that used by the Model 50, provides logical control for the processor. The cycle time of the read-only storage is 250 nsec, with the output being available 100 nsec after select. Sixteen bit planes of 176 words each provide a total of 2,816 words. Each word is 100 bits wide.

To achieve desired CPU speeds, the read-only storage is supplemented with conventional control hardware. Approximately 500 control lines are generated in all, of which 400 emanate from the read-only storage. Conventional control logic is utilized primarily where sequencing becomes data-dependent and exclusive read-only storage control would require additional cycles to be taken.

### LOGICAL ORGANIZATION

The size of the CPU-storage interface varies in width from a byte on Model 30, to half-word on Model 40, to a full word on Model 50. In the Models 60/62, it is double-word wide. A reduction in the number of storage cycles required for program execution is purchased at the price of hardware. The data path continues double word throughout. Significant performance im-

provements in the handling of long precision floating-point arithmetic and variable field length operations are achieved.

A 64-bit instruction-buffer register is directly fed from storage and initially receives all instructions (see Figure 15). Four half-words are loaded and passed, one half-word at a time, through a 16-bit extender register to a 16-bit decoding register where instruction execution commences. As the last of the four half-words enters the extendor register, storage is signaled to refill the buffer register. The flow through these three registers neatly overlaps instruction fetching with execution and adds substantially to the internal performance of the CPU.

A parallel adder, 60-bits wide, is the confluence of the data path. Address calculation, arithmetic, shifts, register-to-register trans-

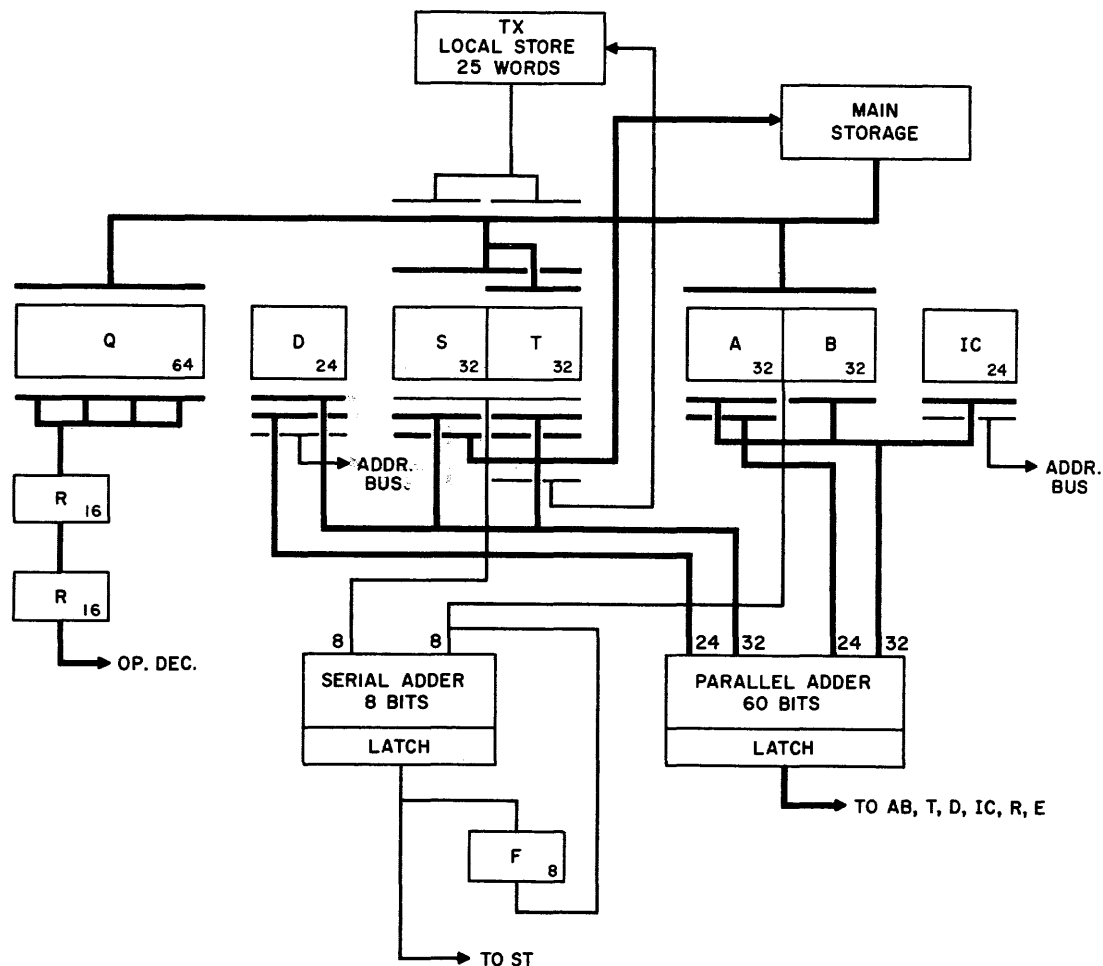


Figure 15. System/360 Models 60/62 Data Flow.

fers, and parity generation and checking are handled in the adder. It is a high-speed unit split into four sections, four groups per section, four bits per group for look-ahead propagation. Worst-case carries ripple out in 135 nsec. The entire adder output can be held in a latch register equipped with gates which can shift the full sum 4 bits right or left into the latch. Adder input gates provide for left shifts of one or two bits. The adder accommodates, in one pass, the 56-bit fraction arithmetic of long precision, floating-point instructions.

Two 64-bit registers operate in concert with the parallel adder. Both can be directly loaded from storage and one provides the store path to main core. At times they participate in an action as a 64-bit unit. Frequently they are treated as separate and independent 32-bit registers (identified logically as A-B, S-T). A finer subdivision into 8-byte units can also be effected. Both the A-B and S-T registers have three-position byte counters controlling byte movements. By utilizing the 32-bit working register in local store, in addition to parallel adder paths, any combination of register-to-register transfer between, A, B, S and T can be made. The base registers in local storage transmit between the S-T registers. A 24-bit instruction counter and a 24-bit storage address register connect to the parallel adder for loading and incrementing.

An 8-bit serial adder draped with extensive gating is woven into the data flow. It drives a latching register and is byte fed into the working registers. The many variable field operations, including arithmetic, work through the serial adder as do the logical functions AND, OR and EXCLUSIVE OR.

To provide maximum flexibility in present or future high-performance system configurations, physically-independent storage and channel units are employed. One common multiplex-type interface serves to permit the attachment of either the 2360 (2  $\mu$ sec) or the 2362 (1  $\mu$ sec) storage units to the CPU and 2860 channel. This interface also provides the mechanism for the attachment of multiple 2361 large-capacity storage units. Figure 16 shows that the key to this interface is the cable that serves as the conductor for multiple driving circuits feeding multiple receiving circuits. This permits effici-

ent time sharing of the inter-unit data and address paths under the control of just a few direct connection signal lines.

#### SYSTEM/360 MODEL 70

System/360 Model 70 is designed to fill a marketing need for a very-high-performance data processing system. It may serve as a direct functional replacement for any member of the System/360 family, with which it maintains complete program compatibility.

The sole constraints on program compatibility are storage size, input/output configuration, and the absence of time-dependent program loops.

The fundamental elements contributing to Model 70 performance are: high-speed circuitry, interleaved 1- $\mu$ sec storage units, and logical organization.

#### CIRCUITS AND PACKAGING

To achieve the desired performance, the Model 70 Engineering team specified a circuit family which would permit an effective balance between the basic machine cycle, storage-access time, and storage-cycle time.

The optimum cycle time was established at 200 nsec. This, in turn, coupled with machine-packaging constraints, dictated a circuit with a nominal switching time of 5 nsec. The resultant circuit is of the AND-OR INVERT family. The switching time varies from 4 nsec and longer as a function of fan-in and loading, i.e., line length and number of loads. The basic circuit configuration is similar to the 10-nsec AOI but differs in component characteristics.

These circuits are packaged in modular form; the modules are mounted on small cards having a capacity for either 12 or 24 modules. A substantial number of these small cards are functionally packaged to achieve high density and relatively short line lengths. With a circuit that performed well, extreme care had to be exercised, not only in defining the functional cards but also in card placement. More difficulty was experienced in controlling delays in transmission than logical delays.

#### STORAGE

Main storage for the Model 70 consists of two banks of 1- $\mu$ sec storage. Each storage bank has

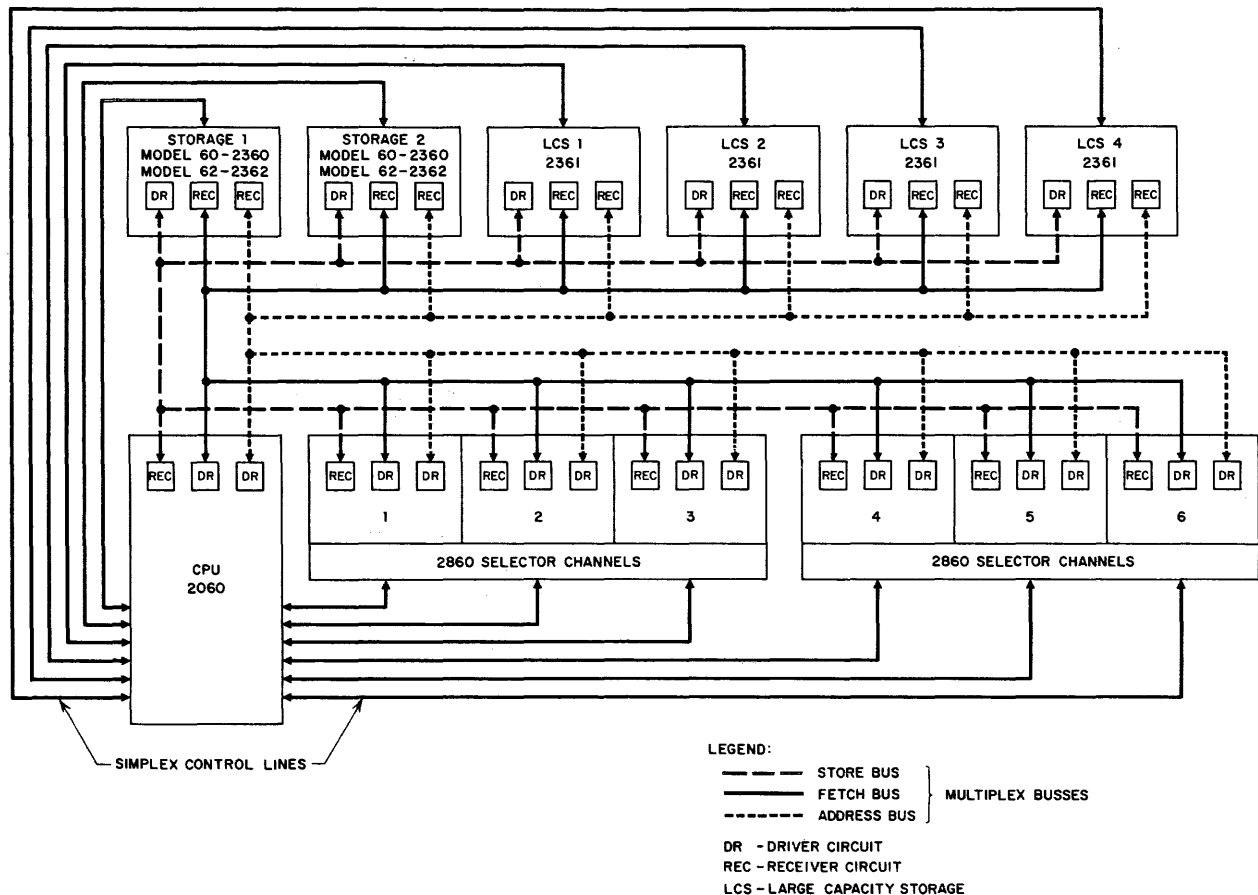


Figure 16. Model 60 Storage-Channel Interface.

a capacity of 256 K bytes arranged in 32K double words (64 information +8 parity bits). Data with even double word addresses are stored in one bank; data with odd double word addresses are stored in the other. These two banks are independent of each other, having exclusive drive schemes and data registers.

When accesses are made to sequential storage addresses, the storage units operate in an interleaved fashion.

The two units cannot be accessed concurrently but must be offset by at least 400 nsec, a restriction imposed by the maximum rate of the storage bus.

#### LOGICAL ORGANIZATION

The initial design approach was to accentuate the performance of arithmetic operations.

Emphasis, however, was brought to bear on those instructions used heavily in the compiling function, e.g., load, branch, store, and compare. A data flow schematic is shown in Figure 17.

Instruction buffering is provided via two double-word registers, each supplied by independent banks of main storage. Instructions are executed sequentially and as the contents of each register is exhausted, it is replenished from storage during which time the contents of the second register is being used.

Some degree of instruction overlap is achieved by operand buffering. A sequencing unit controls the instruction decoding, effective address generation, and operand fetching, while the preceding instruction is being executed. The sequencing unit is not allowed to operate more than one instruction ahead of the execution unit and is not allowed to change any address-



ble registers while the execution unit is still operating on the previous instruction. This eliminates recovery problems in the event that a branch or interrupt causes the abandonment of the instruction being worked on by the sequencing unit. Address generation is accomplished through the use of a three-input adder in one machine cycle (200 nsec); one input is from the instruction register, the other two from the general purpose registers specified by the instruction.

The heart of the execution unit is the main adder which is a 64-bit adder-shifter combination. It has a three-stage full carry lookahead scheme which permits an add operation in one clock cycle. The main adder is supplemented with an 8-bit exponent adder for floating point operations and an 8-bit decimal adder for decimal and VFL operations.

A read-only storage mechanism was not included in the Model 70 since it did not lend itself to the machine organization, especially in the area of required cycle time. As a result, the control functions were implemented through conventional logic design.

A further attempt to improve performance was made through utilization of transistor circuitry, rather than core storage, for the general purpose and floating-point registers. This approach permitted faster access, eliminated regeneration time between fetches, and permitted accessing more than one register at a given time.

In view of the storage interleaving and instruction overlap, a precise estimate of machine performance on a particular program, loop or sub-routine, must involve careful scrutiny of instruction and data addresses and instruction sequences.

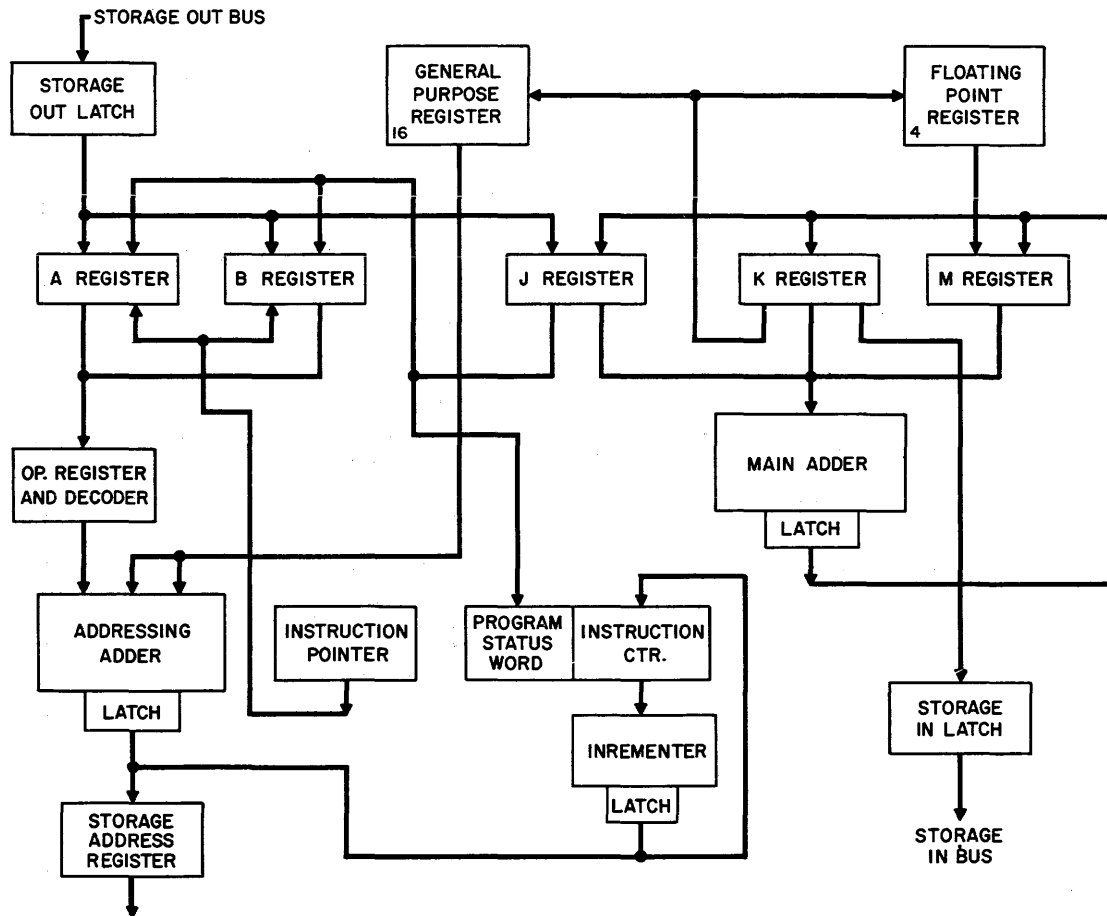


Figure 17. System/360 Model 70 Data Flow.

## LARGE CAPACITY STORAGE

A considerable asset in achieving thruput improvement is the addition of large capacity storage (LCS) to the Model 70. This storage unit, with a read-write cycle of  $8 \mu\text{sec}$ , may be attached to the system in increments of 1024K and 2048K bytes up to a maximum of 8 million bytes. These units are attached directly to the storage bus and can be considered an extension of directly addressable main storage.

The LCS can also be attached to Models 50, 60 and 62, and has the facility of being shared between any two of these systems.

## CHANNEL

The 2860 Selector Channel is a very high performance data channel designed to operate on Models 60, 62 and 70.

The performance of this channel utilizing 30-nsec circuits permits operation at data rates up to 1.3 megacycles; the rate being measured by the number of 8-bit bytes that pass, via the I/O interface, to or from an appropriate input or output control unit. Any interconnection to input or output control units is via the I/O interface.

The channel is of the selector type, permitting interconnection of multiple I/O control

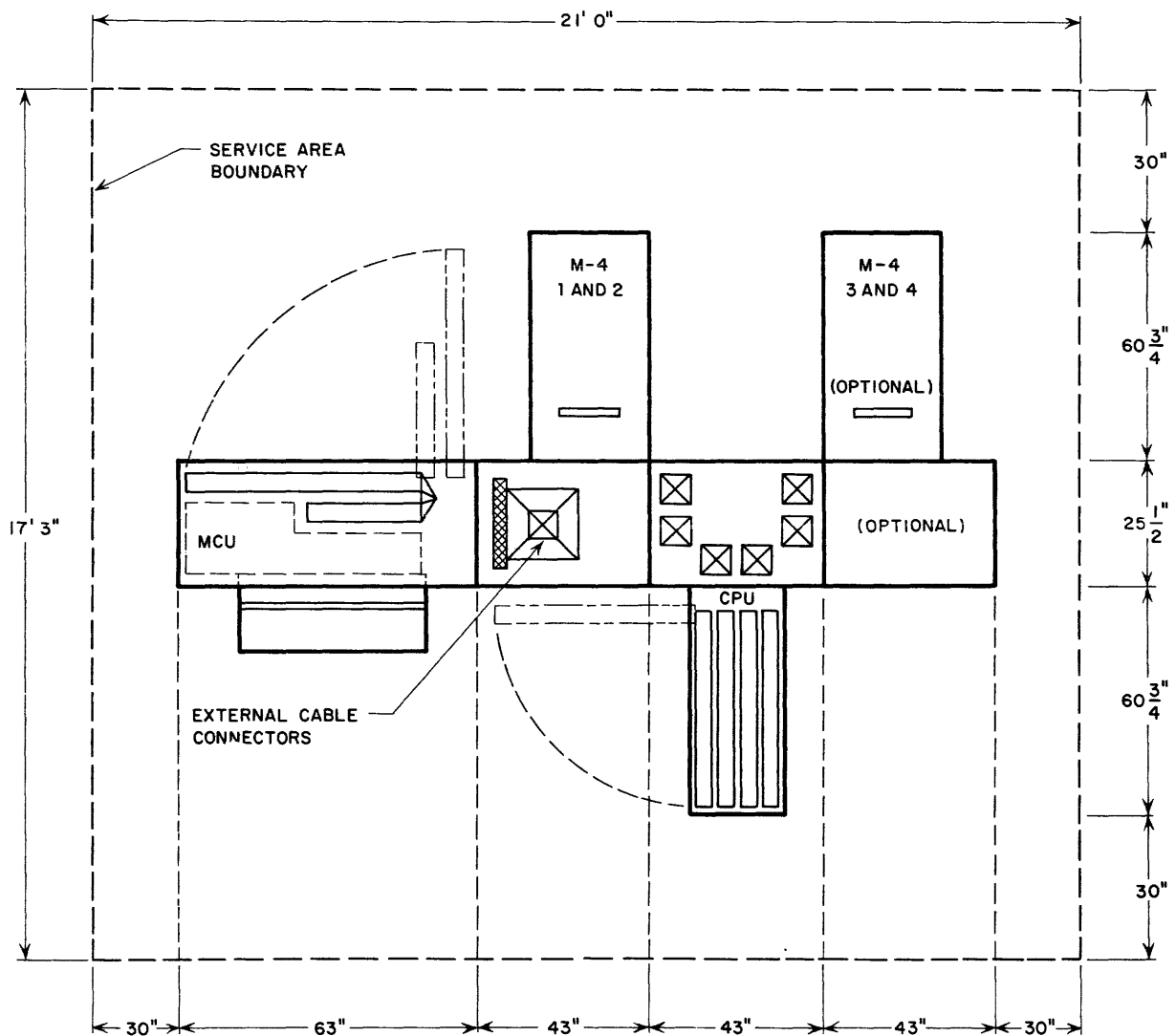


Figure 18. Model 70 Physical Arrangement.

units to a single channel. A maximum of 6 channels may be used. Up to 8 control units may be attached to one channel. However, the channel at any given time operates with one, and only one, device of the many attached. The channel is "instructed" by the processor system to commence an operation. An operation performed by an instruction may involve any sequence or list of commands to that particular device. After being instructed, the channel independently obtains "commands" and transmits data to or from processor storage until it completes its operation.

In operating with storage, the channel shares the buss control unit, used by the processor, to obtain its storage references. The Models 60, 62 and 70 use a double level of priority. In the first level, each channel vies with the other channels attached to that particular CPU for priority, and then in turn vies with the processor for specific priority.

#### PHYSICAL ORGANIZATION

Figure 18 shows the Model 70 physical arrangement. Each gate in the CPU contains 20 large cards and 4 half-size cards for termination of interframe cables. Two of the gates contain the Execution-unit, the other contain the sequencing-unit. The maintenance-control unit (MCU) frame contains maintenance circuitry,

the bulk power supply, and a small amount of the CPU circuitry.

Mounting the CPU and the storage units on a common wall helps performance in that it allows short, direct cable connections between them, rather than long under-floor cables. It also means that all installations will have the same cable lengths in these paths.

The 2860 Selector Channel frame is a three-gate stand alone frame housing three swinging gates, each capable of containing 20 large cards. Power supplies are mounted in the internal column between gates. Since each channel occupies one full gate, up to three channels may be contained in a given three-gate frame.

#### BIBLIOGRAPHY

1. AMDAHL G. M., BLAAUW G. A., and BROOKS F.P. JR., "Architecture of the IBM System/360," *IBM Journal of Research and Development*, Vol. 8, No. 2 (April 1964).
2. DAVIS E. M., HARDING W. E., SCHWARTZ R. S., and CORNING J. J., "Solid Logic Technology: Versatile, High-Performance Microelectronics," *IBM Journal of Research and Development*, Vol. 8, No. 2 (April 1964).
3. CARTER W. C., MONTGOMERY H. C., PREISS R. J., and REINHEIMER H. J. JR., "Design of Serviceability Features for the IBM System/360," *IBM Journal of Research and Development*, Vol. 8, No. 2 (April 1964).



# UNISIM—A SIMULATION PROGRAM FOR COMMUNICATIONS NETWORKS

*J. H. Weber and L. A. Gimpelson  
Bell Telephone Laboratories, Inc.  
Holmdel, New Jersey*

## I. INTRODUCTION

The design and analysis problems associated with large communications networks are frequently not solvable by analytic means and it is therefore necessary to turn to simulation techniques. Even with networks which are not particularly large the computational difficulties encountered when other than very restrictive and simple models are to be considered preclude analysis. It has become clear that the study of network characteristics and traffic handling procedures must progress beyond the half-dozen switching center problem to consider networks of dozens of nodes with hundreds or even thousands of trunks so that those features unique to these large networks can be determined and used in the design of communications systems. Here it is evident that simulation is the major study tool.

## II. SIMULATOR REQUIREMENTS

The type of problem to which this simulator is directed is quite different from many of the queuing and other problems which are the primary application for most simulation programs. Whereas in management simulation and tandem queuing processes (job shop simulations, etc.) problems are characterized by a fairly complex sequence of possible alternatives, the number of demands simultaneously in process is ordinarily not so great as to tax the capacity of the computer.

Furthermore, the measured outputs are directly influenced by most or all of the transactions sequenced through the program.

In telephone (and other) traffic simulations, however, particularly of the network type, the possible number of alternatives before any call (demand) is not very great, but the number of calls which are simultaneously in progress is quite large, and their interactions are not predictable. The measure of performance is typically grade of service (or probability of blocking), which is normally in the order of one per cent of the total offered calls. This measure, furthermore, applies to each traffic parcel in the network. For example, if there are 20 nodes in a network, there are 190 two way traffic items, each with a different demand rate. If the smallest demand contributes only 1/1000 of the total calls in the network at any time, then 1000 calls must be processed for each one from this smallest parcel. A blocking of one per cent should be measured with reasonable reliability on this smallest parcel, and even if 1,000,000 calls are processed, only (1/1000) (1/100) (1,000,000) or ten calls will be blocked, and therefore contribute directly to the measurement on the smallest parcel. Since many networks must be tested using different load levels, it is clear that a primary requirement for a simulator which is to be used in traffic network studies is that it be fast. An improvement in speed of 5

per cent or 10 per cent can be worth many thousands of dollars even for a single study.

The other important characteristic that simulators for this application must have is that they be capable of handling large networks. The toll network in the United States has in the order of 2000 switching centers. Although it is not possible to incorporate even a substantial fraction of this number into a simulation program, the number of nodes which can be accommodated must be capable of exercising all of the proposed routing and control parameters, which can ordinarily not be done with anything fewer than about 20 or 30 nodes, and preferably should be somewhat larger. In practice, the capacity of a simulator is not governed by the number of nodes in the network, but by the number of calls simultaneously in progress, and this maximum must be commensurate with the maximum number of allowable nodes. It would do no good to simulate a 2000 node network which allowed less than one simultaneous call in progress per node pair since this would not be a realistic situation. This requirement, of course, is in conflict with the speed criterion, since to take advantage of low speed bulk storage media such as disc files might cause an intolerable slowing down of the program.

### III. SIMULATOR CHARACTERISTICS

The simulator can accommodate systems with both direct (line switched) and store-and-forward traffic and with two nonpreemptive priority levels allowed for each traffic mode. It also allows trunk reservation by mode, priority, or route and can develop its own alternate routing tables according to specified rules. These can then be dynamically changed according to the state of the system.

All congestion is assumed to result from trunks only: the switching centers offer no delay or blocking to any call. This assumption, although perhaps somewhat unrealistic in certain circumstances, allows the simulation to operate quite rapidly while facilitating the evaluation of alternate routing patterns strictly on the basis of their basic structure and routing doctrine, without introducing obscuring effects of particular switching ma-

chines. It is felt that the structure information will be common to a large variety of networks, whereas the switching machines are of course unique to each particular system.

The essential characteristics of the simulator are as follows:

1. Two modes of traffic, direct and store-and-forward, are allowed. Direct traffic is served upon arrival, using either a direct or an alternate route. If a direct call is unable to be served immediately, it is considered blocked and is either lost from the system or reattempts after some fixed interval of time. The reattempt interval may be exponentially distributed or constant, and calls may either reattempt after each try or be lost with a given probability. Store-and-forward calls are served immediately, using the direct or alternate route, if possible. If the call cannot be served immediately, it is stored and queued on the most direct route.
2. Each mode of traffic is allowed two nonpreemptive priorities, and these can be distinguished by:
  - a. Different retrial times for direct traffic.
  - b. Head of the line queuing for the higher priority store-and-forward traffic.
  - c. Trunk reservation procedures in which a certain number of trunks in a group are reserved for high priority traffic only.
3. The simulator accommodates networks with the following maximum dimensions:
  - 63 nodes
  - 1,953 trunk groups

These maxima cannot be simultaneously realized, the primary limitation being approximately 6,000 calls in progress simultaneously.

The simulator will handle approximately 500,000 calls per computer hour; this is a maximum speed obtained when a moderately loaded network of about 35 nodes is being simulated (i.e., the number of calls in queues and being retried is

small compared with the number of calls in progress). The time required to process the simulator output and produce statistics on the run (included in the above estimate) varies with the network size and may take as long as the simulation itself.

4. The alternate routing procedure which is provided determines its own symmetrical routing arrangement (see Reference 1) based on the shortest paths between each two points, hierarchical routing according to preset rules or general routing with the route sections read in directly. The routing either remains fixed throughout the run or can be changed at periodic intervals according to the dynamic condition of traffic in the network.
5. The output data are arranged such that information about probability of blocking, delay distributions, trunk usage and several other aspects of the system are summarized over prearranged intervals of time. The results of these summarizations are then printed out, as well as placed on magnetic tape, and selected sections can then be combined using another program to derive means and variances of the appropriate statistics over any group of intervals which is desired.
6. The simulator has the ability to change loads at a linear rate during the course of the run; that is, any load or combination of loads can be made to vary at a fixed rate over a desired period of time, including a step function in which a change is made in zero time.
7. Trunk reservation is set up not only to distinguish between high and low priority calls but also between direct and store-and-forward calls: it is possible to reserve some trunks for direct calls only, as well as for the higher priority. Trunks may also be reserved for first routed traffic at the expense of alternate routed traffic.
8. Routing is of the "stage-by-stage" variety. As calls progress through the network their next choice of route is

determined only by the condition of the immediately succeeding links. Calls are not allowed to switch through the same node twice, and maxima, in the form of number of links and distance, can be established for each traffic item. It is also possible to allow calls to return to a previous point for rerouting if they are blocked.

#### IV. SIMULATOR ORGANIZATION

The Simulator Program (Figure 1) is made up of a number of subprograms, some of which are simultaneously in core and some of which are read in sequentially. This subprogram structure was used in order to accelerate the programming, maximize the availability of core storage for any program, simplify debugging and allow maximum flexibility for future changes. Briefly, the sequential programs are as follows:

1. The *Traffic Generator* accepts as input data the point-to-point offered loads, the holding times of the various types of

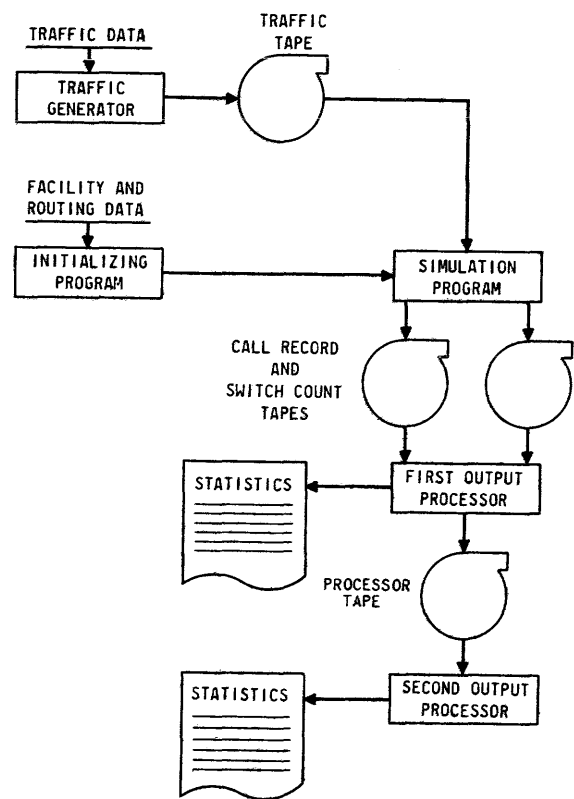


Figure 1. Simulator Organization.

traffic, and the load changes which can be expected during the course of the run. The program generates all the calls which will be used in a simulation run, placing their mode and priority, time of arrival, originating and terminating points and holding time on the Traffic Tape in chronological order. This tape is used as input to the main Simulation Program.

2. The *Simulation Program* accepts as input the structure of the network in terms of nodes and trunks; trunk reservation information, if any; routing options and limitations; retrial specifications; and the Traffic Tape. It then processes the calls through the simulated system and reports the results on two tapes. One of these, called the Call Record Tape, contains a record of all calls which have been processed; that is, arrival time, service time, holding time, mode and priority, origin, destination and number of links used. The other tape, called the Switch Count Tape, records the results of periodic measurements of all of the trunk groups in the network, reporting on occupancy and reservation levels. These two tapes contain all of the raw output information from the simulation and are used as input to the Output Processor Programs. The Dynamic Alternate Router Program, which determines the routing pattern as a function of the traffic condition of the network, is alternated in core with the main processing programs, the program not in use being temporarily stored on a scratch tape or disc file.
3. The *First Output Processor* accepts as input the Call Record Tape and the Switch Count tape, as well as specifications of the number and length of the time intervals over which the information on the two tapes are to be averaged. It then prepares mean values of blocking probabilities, delay distributions, average delays, trunk usages and links-per-call distributions over the specified intervals. This information is printed out and placed on magnetic tape. By visual inspection of

these outputs the program user can then determine the intervals over which the system was sufficiently close to steady state operation to allow longer term averaging. He then can make the appropriate specifications for the Second Output Processor.

4. The *Second Output Processor* accepts the output tape from the First Output Processor and the interval averaging specifications of the program user, and determines over-all means and variances of all system statistics originally derived by the First Output Processor.

The detailed operation of the several programs mentioned above will be given in subsequent sections.

## V. TRAFFIC GENERATOR

The Traffic Generator Program (Figure 2) is run prior to the Simulation Program and the Traffic Tape which is produced can be re-used as required. This procedure realizes a saving in computer time since traffic need not be regenerated to test several network configurations and routing schemes.

The Traffic Tape contains an entry for each offered call consisting of essential information such as arrival time, terminal nodes, holding time, etc. The calls are generated by using input quantities such as offered loads to generate interarrival times using independent random numbers, selected from a specified distribution (single parameter with unit mean). The program continually searches for the next earliest arrival. Finding that item, a holding time and direction are determined. This information is placed on the output tape. The item then receives a new arrival time, is placed back into the list, and a new search is initiated for the next arrival.

The selection of the earliest arrival is expedited by the use of a technique, suggested by W. S. Hayward, Jr., in which entries in the list are paired, and the earlier of the two arrival times of each pair is placed in a second list. This same process is continued using pairs from subsequent lists until the earliest arrival is selected. The node pair and call type associated with the arrival time can be simply



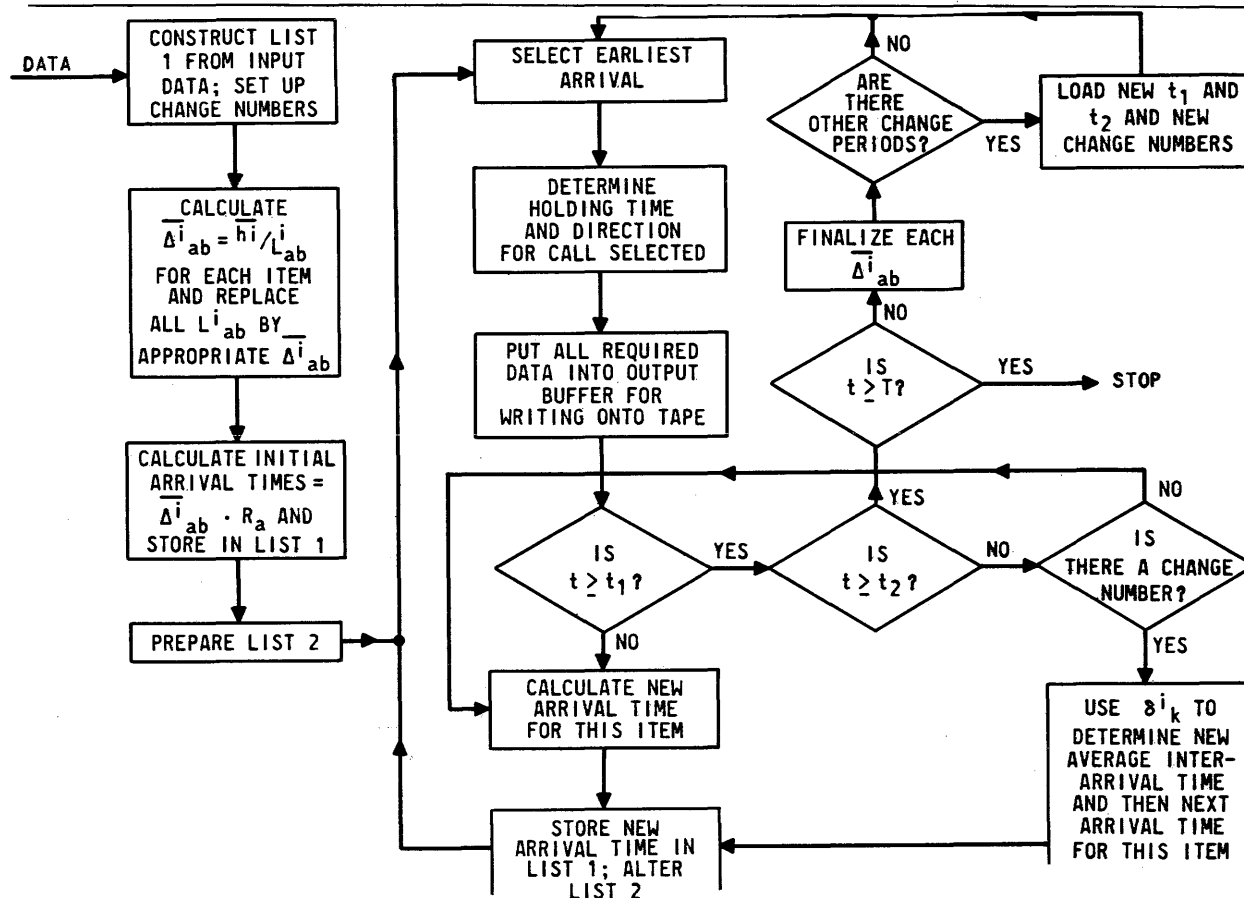


Figure 2. Traffic Generator.

determined. The economy of this technique results from its ability to change individual items very rapidly without full reconstruction of the lists; in particular, it can select the required call more quickly than would be possible with a full search of the first list.

During prespecified time intervals in which there are to be load changes, a calculation of each new interarrival time is preceded by a test for the presence of a flag in the listing of the current item of traffic. Should there be one, the interarrival time is modified to produce a linear change in the load.

## VI. SIMULATION PROGRAM

### A) General Description

The Simulation Program (Figure 3) is composed of a number of subprograms which can be grouped into three categories:

1. Operator Program
2. Routing Programs
3. Record Keeping Programs

The division of tasks between the subprograms was made to allow separate writing and debugging of the programs by a team of programmers. This technique also permits changes in operation to be effected in parts of the simulator without major rewriting of the entire program; for example, several routing schemes are available as "plug-in" units.

The Operator Program maintains a single chronological queue (or linked list) containing all events which occur in the simulator. This includes call arrivals, call departures, call retrials and various instructions to perform control actions. For example, if the event at the head of the queue is the arrival of a new call, placement is attempted; successful placement will move the call back in the queue to a point in time equal to the arrival time plus the holding time; when that time is reached (following processing of intervening events), this call is removed from the network.

In order to maximize the number of calls simultaneously in progress, advantage was taken

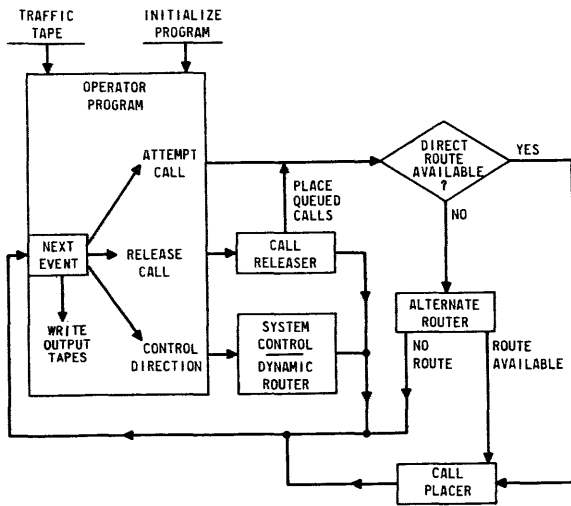


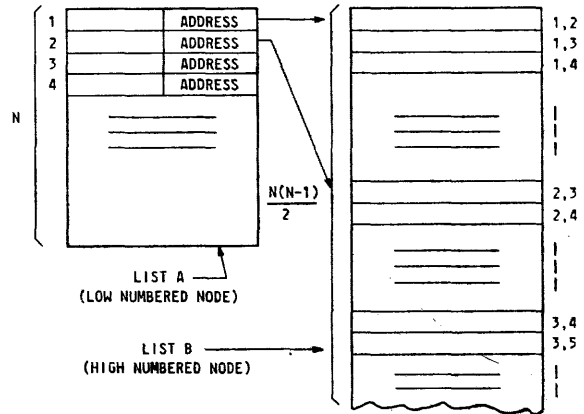
Figure 3. Simulation Program.

of the statistics of network behavior in organizing the data structure. Since the chronological queue is the basic data item using most of the memory, the cell size to be used was of critical importance. It turned out that if a call used only one, two or three links, then three words would be sufficient, but if more links were needed (and a maximum of seven was a requirement for some calls) a larger cell would be needed. In most networks at least 95 per cent of the calls use fewer than four links; so the cell size was set at three words, with the possibility of adding additional words if the particular call required them.

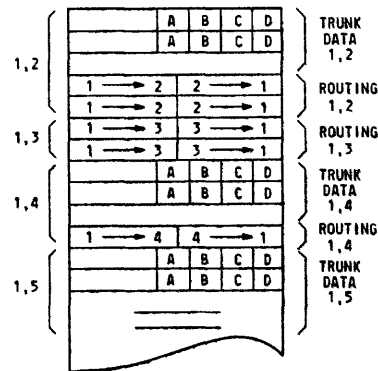
The Operator Program presents a new call to the Direct Router, which determines whether a single-link placement is possible by checking trunk availability for the mode and priority of the new call. If a direct route cannot be obtained, the call is presented to the Alternate Router, which attempts a multiple-link connection. The Alternate Router uses a Trunk and Routing Table (described below) to find an available route for the call. A valid route obtained by either routing program is transmitted to the Call Placer, which updates the Trunk and Routing Table to account for the new call. That call (with its event time changed to its departure time) is returned to the Operator Program for reinsertion into the chronological queue. Upon complete failure to route the call, the Operator will either add

the call to a nonchronological queue (for store-and-forward traffic) or, with a new event time obtained by adding a retrial interval to the arrival time, place the call at this event time in the chronological queue (for direct traffic). At the conclusion of the call the Operator presents the call to the Call Releaser, which alters the Trunk and Routing Table to account for the call's withdrawal and checks each trunk group previously employed by this call for the presence of nonchronological queues, reporting these to the Operator.

The Trunk and Routing Table (Figure 4b) contains the occupancy status of each equipped link (Trunk Data) and routing information for each pair of nodes (Routing). Thus the same table keeps a record of the traffic on individual links and provides full routing information for each pair of nodes in the simulated network. The routing information consists of lists of intermediate nodes to be used for calls which cannot be placed directly.



a. LISTS FOR ENTERING TRUNK AND ROUTING TABLE



b. TYPICAL TRUNK AND ROUTING TABLE

Figure 4. Typical Trunk and Routing Table.

The Trunk and Routing Table is constructed from the input data by an Initialization Program (Figure 5). For symmetrical routing the program is supplied with point-to-point distances (or other weightings) and the numbers of trunks installed throughout the network. It then determines a number of economical routes in each direction between every two nodes. The first intermediate node to be tried for each

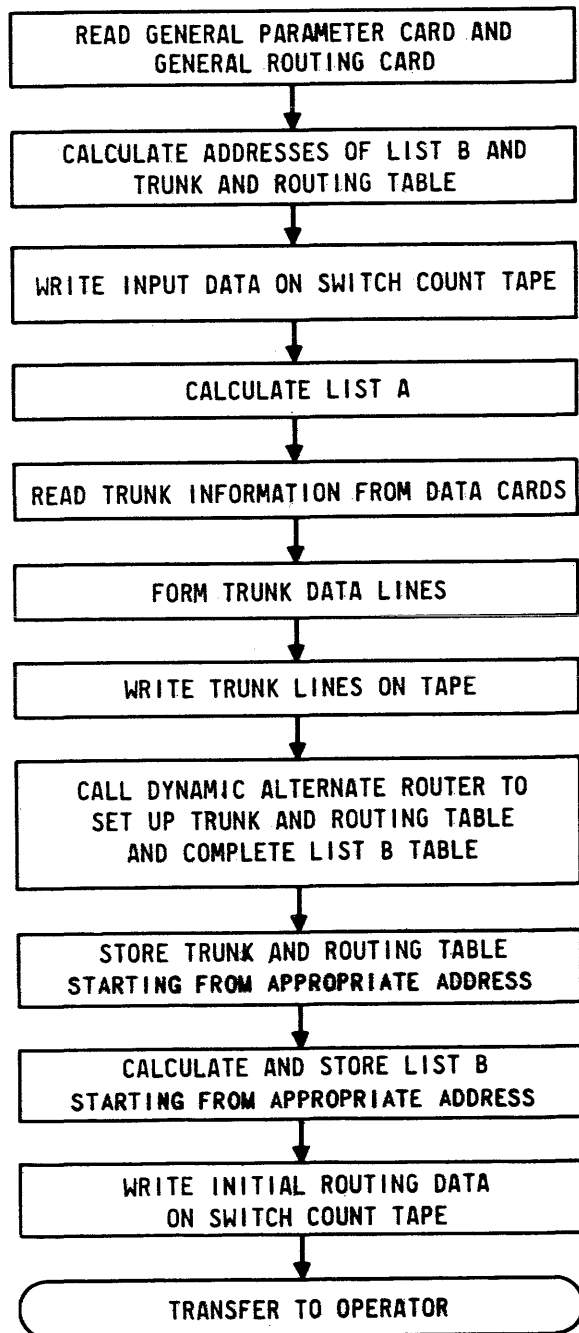


Figure 5. Initialization Program.

route selected, together with the minimum number of links required using that node, is entered into the Trunk and Routing Table. This method of preparing the routing data is especially convenient when large networks are being tested since the number of alternate routes becomes so large that manual specification is not feasible.

The Initialization Program also determines two lists which facilitate entering the Trunk and Routing Table. Shown in Figure 4a for a network of  $N$  nodes, there are  $N$  entries in List A, each consisting of the address of the first line of a series of entries in List B. When either trunk availability or routing information is required for a specific call type between nodes  $X, Y$  ( $X < Y$ , numerically; direction of call is not a consideration at this point), indirect addressing permits use of line  $X$  of List A to immediately obtain that line of List B which pertains to the node pair  $X, Y$ . List B contains information required to route a call from  $X$  to  $Y$ .

#### B) Direct Router

When calls of the following classes reach the top of a chronological queue they are delivered to the Direct Router:

1. New calls.
2. Calls which are to be retried after having previously been blocked.
3. Calls at the head of a nonchronological queue associated with a link on which a call has just been released.
4. Store-and-forward calls which have just completed transmission to an intermediate node after having previously failed to reach their destination.

The Direct Router (Figure 6) must determine the nodes between which a connection is required. If the node pair is unequipped (rapidly determined by a sign-bit test in the Trunk and Routing Table), the Call Data is sent immediately to the Alternate Router. If there is a trunk group installed, the Direct Router enters the Trunk and Routing Table and, using the call type, determines the availability (considering reservations) of the trunk group for this call. The lack of an available trunk sends the Call Data to the Alternate

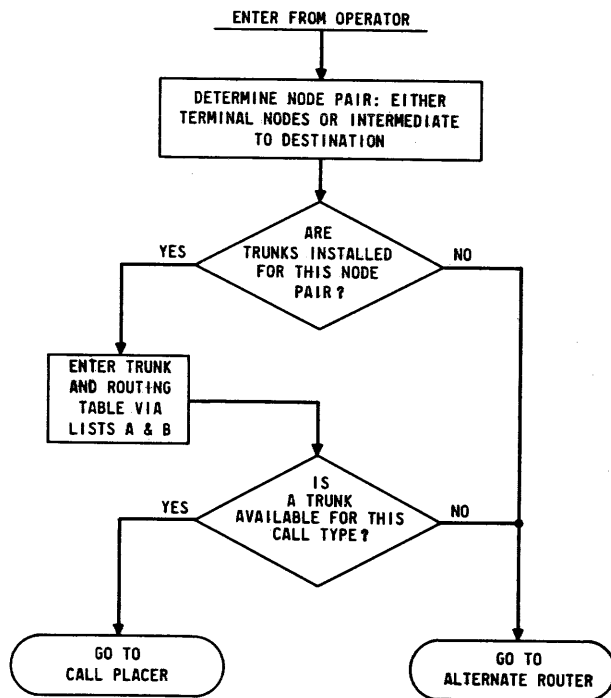


Figure 6. Direct Router.

Router, while availability indicates that the call can be placed to its destination, and the Call Placer is supplied with this information.

Since every call is offered to the Direct Router and most calls are carried over direct routes, this program was written with particular attention to operating speed. In conjunction with this the structure of the Trunk and Routing Table was largely determined to facilitate its use by the Direct Router.

C) *Alternate Router*

A call is referred to the Alternate Router (Figure 7) only if it has been determined that there is no direct route available.

The Alternate Router attempts an immediate multiple-link connection, progressing through the network on a node by node basis. Should the Router either exhaust the list of node choices or be unable to find a route with no more than the permitted maximum number of links, a direct call will be returned to the Operator Program for retrial at a later time (a specified per cent of the retrial traffic can be "lost" rather than retried), while a store-and-forward call will be queued on a trunk group.

If there is a direct link between the terminal nodes, the store-and-forward call will queue on this link immediately; if there is no such link, the call will queue along the "first choice route;" this is a route selected to have no more links than the minimum number of links specified when the first choice of an intermediate node was obtained by the Alternate Router.

"Crank-back" or call "back-up" for direct traffic is available (Figure 8). When used, a call which has been blocked at some point in its routing releases the last link accepted, returning to the previous node where it now tries to reach its destination via another link. The number of links which may be released before each forward progression is an input quantity.

D) *Call Placer and Releaser*

Upon indication from either router that there is an available route for the current call, the Call Placer (Figure 9) determines which

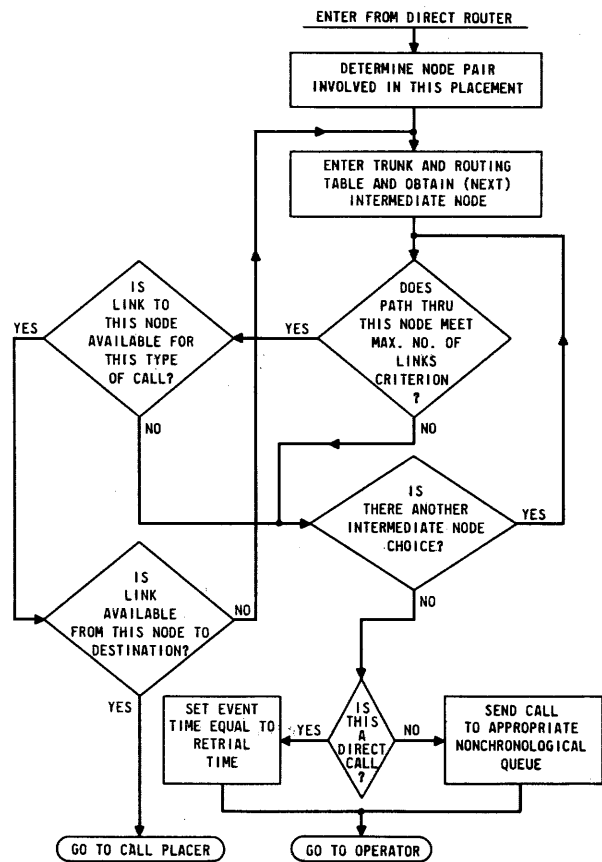


Figure 7. Alternate Router (Without Dynamic Router or Call Back-Up).

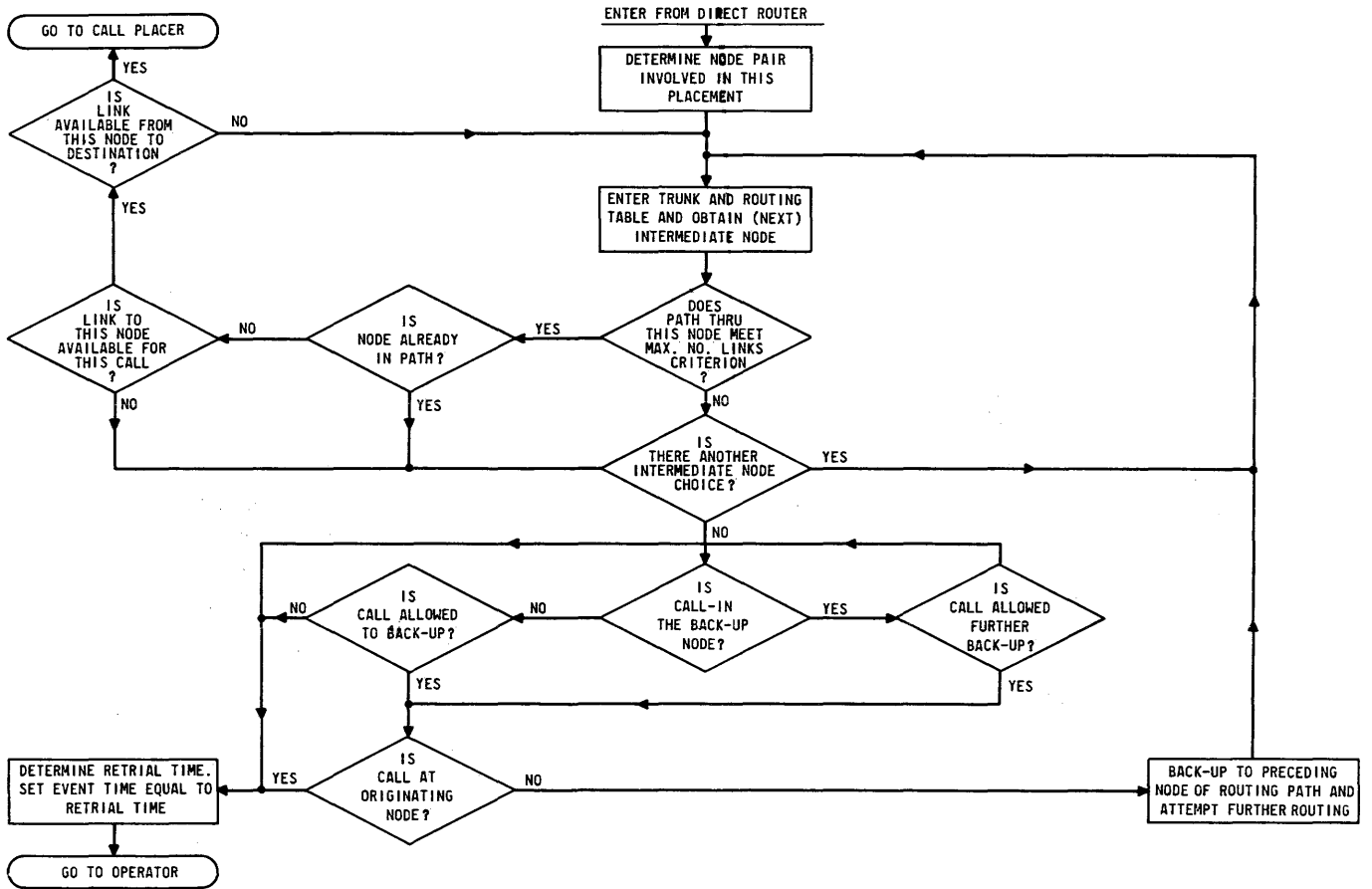


Figure 8. Option 2 Alternate Router (Without Dynamic Router, But With Call Back-Up).

links require updating, enters the Trunk and Routing Table and alters the appropriate data. After updating these records the Call Placer returns the call to the Operator for reinsertion into the chronological queue at the release time.

At the conclusion of a call, the Call Releaser (Figure 10) must update the Trunk Data. Then the call is sent to either the output queue (for transfer to the Output Tape) or to a non-chronological queue on a trunk group (for a store-and-forward call not yet at its destination and still blocked). Finally, the Call Releaser checks each of the trunk groups used by the call just released to determine if there are any nonchronological queues waiting for these groups and presents this information and program control to the Operator.

E) Operator Program

The chronological queue is administered by the Operator Program (Figure 11) using

linked lists, which facilitate the maintenance of information in the computer by allowing the ordering of data to be easily altered. In the simulator, a list number is stored with each chronological event. This number is the address of the next event in the queue. The address (or location in core) of an event remains unchanged from the time it enters the simulator until it is removed (for example, from original entry from the Traffic Tape until the end of processing and placement on the Call Record Tape). The position of a call in the queue, however, is easily changed by altering the list numbers of the preceding event and the event being moved. Vacancies in core are linked together in the same fashion using a push down vacancy list. Nonchronological queues are similarly constructed by taking blanks from the vacancy list. When calls have been completed they are linked into an output queue which is periodically read on to the Call Record Tape; after this the slots are returned

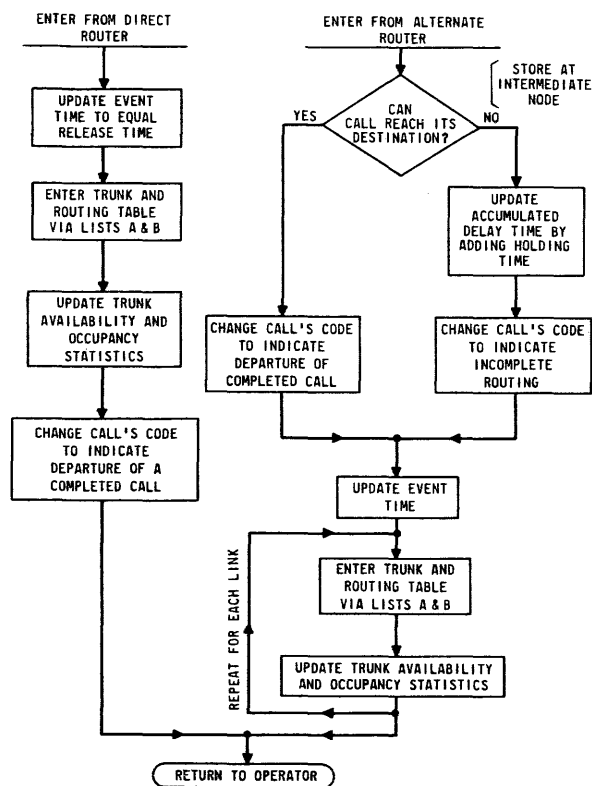


Figure 9. Call Placer.

to the vacancy string. These several cohabitant lists maintain their own links and only the first element of each need be identified.

The large size of the chronological queue makes it inadvisable to search through the list from the beginning for each insertion. Pointers, actually addresses of intermediate events, are used by the Operator Program to enter the chronological queue at spaced intervals in event time; once in the vicinity of the required event time, the Operator steps along call by call to the exact point in time for insertion of an event.

## VII. OUTPUT PROCESSOR PROGRAMS

In order to effectively display the result of simulation experiments, processing programs calculate and display the expected values of important parameters and measure their statistical variability. In order to perform these functions most effectively, these output programs have been divided into two separate sections.

The first of these, called the First Output Processor, reads raw system statistics from the Switch Count and Call Record Tapes and evaluates the mean values of appropriate system statistics over prespecified subintervals of time. These statistics are then printed out and also read onto magnetic tape. The program user can examine the results and, by ascertaining the time periods over which equilibrium was obtained, write the specifications for the second program, called the Second Output Processor.

The Second Output Processor collects the appropriate information from the Processor Tape (generated by the First Output Processor) and determines means, standard deviations, and over-all network statistics for a prespecified number of processing intervals.

The First Output Processor determines average values of certain quantities over prespecified time intervals. Each time interval contains a specified number switch counts (which are snapshots of each trunk group reporting the number of trunks busy at a given instant).

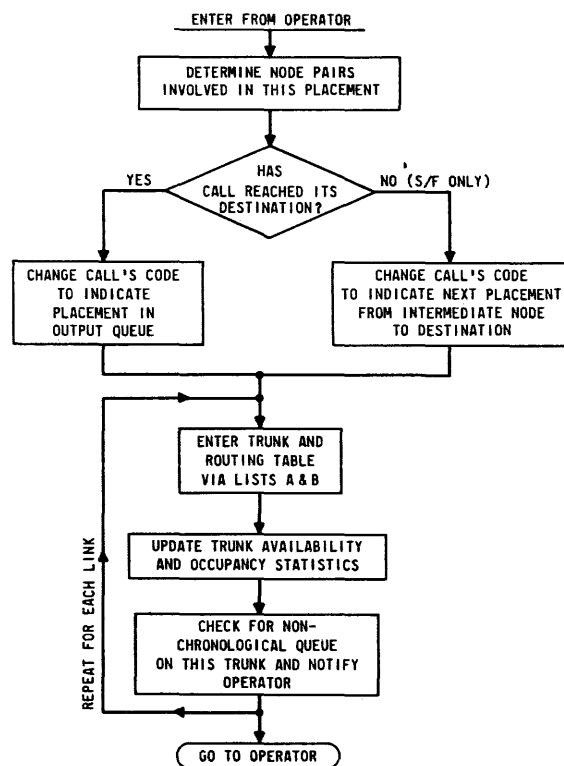


Figure 10. Call Releaser.

These intervals are ordinarily short enough so that no important trends will be obscured. All quantities are stored on the Processor Tape in a form suitable for reading back into memory. The quantities which are evaluated and printed out (both on tape and on paper) are as follows:

A. *Input and Miscellaneous*

1. Input information, including offered loads by mode, priority, origin, and destination. This information is simply transferred from the Traffic Tape via the Switch Count Tape to the Processor Tape and is printed without further processing.
2. Time of start and completion of the measurement period for each reporting interval and the reporting interval number. The reporting intervals for each run are assigned consecutive numbers to be used in specifying the analyses to be done by the second stage program.

B. *Call Tape*

1. Blocking probabilities, delay distributions and average delays of calls by mode, priority and direction for each origin-destination pair. Five points in the delay distribution are kept for each item. The magnitude of delay at each of the five points can be specified as an input to the First Output Processor.
2. Distribution and average of number of links per call, with a maximum of seven links per call recorded and with the same breakdown as in Item 1 above.

C. *Switch Count Tape*

1. Number of switch count intervals which are included in the reporting interval.
2. Number of trunks in each group.
3. Carried loads for each trunk group (either by mode and priority, or by first routed or other).
4. Trunk reservation levels for each trunk group.
5. Distribution and average of queue lengths by priority for each trunk

group. Three points in the distribution are kept. These points are specified in the input to the First Output Processor.

6. Alternate routing tables when they have changed.

The Second Output Processor prints the same statistics as the First Output Processor, averaged over a prespecified number of processing intervals. Each period of time for which a Second Output Processor calculation is made is called a Reporting Interval and the program provides the standard deviation of each measurement based on the variation among processing intervals within a reporting interval. In addition, the Call Record Tape items are averaged for the two directions of traffic in each point-to-point item; that is, the blocking probabilities and delays for traffic from node 1 to node 2 and from node 2 to node 1 are averaged to give a single value for the node pair (1,2). Means and standard deviations of over-all weighted average values of each measurement are also calculated. For example, the over-all blocking probability is the average of all of the point-to-point blocking probabilities weighted by the offered loads or:

$$\bar{B} = \frac{\sum_{ij} a_{ij} B_{ij}}{\sum_{ij} a_{ij}}$$

where  $a_{ij}$  = offered load between node  $i$  and node  $j$

$B_{ij}$  = blocking probability of calls between  $i$  and  $j$ .

In addition to these over-all system means and variances or standard deviations, a statistic called the Misery Factor has been added. This is a measure of the maldistribution or unbalance of call tape statistics throughout the network. It is defined for blocking probability as:

$$M_B = \sqrt{\frac{\sum_{ij} a_{ij} B_{ij}^2}{\sum_{ij} a_{ij}} - \bar{B}^2}$$

Sample printouts from the Second Output Processor are shown in Figures 12 through 16. These were obtained after the simulation of a 34 node network; note that the printing of trivial lines (no traffic, no alternate routes, etc.) has been suppressed to condense the data and that the figures include only the first page of the printout in each case. Figure 13 shows the input to the Traffic Tape as transferred to the output by the simulator; Figure 13 shows the initial alternate routing table determined by the simulator; Figure 14 shows the Switch Count Tape reduction over specified processing intervals; Figure 15 shows the Call Tape reduction for the same intervals used in the previous figure; and Figure 16 shows the overall means, standard deviations and Misery Factors for the time intervals listed in the columns at the left. The measure of statistical variability is the standard deviation of the measurement over the reporting interval.

VIII. APPLICATIONS AND EXTENSIONS

The most extensive application of UNISIM to date has been an investigation into possible

routing configurations and control schemes for the telephone toll network. This work has been described in Reference 2. Since the purpose of such simulators is essentially to evaluate performance in advance of detailed design and implementation, it, by its very nature, cannot be directly validated by comparison with experimental results. The results which have been obtained, however, to the extent that they can be compared with measured performance of operating systems, have been in good agreement with observed phenomena.

Subsequent applications have been concerned with the evaluation of certain military communications networks.

It was found in the military applications that although the program for the most part was directly applicable, certain changes were required as new routing philosophies were encountered. Since the program was modularly constructed, these changes were restricted to one subprogram, the Alternate Router. UNISIM, however, was written almost entirely in machine language (assembly) code (FAP),

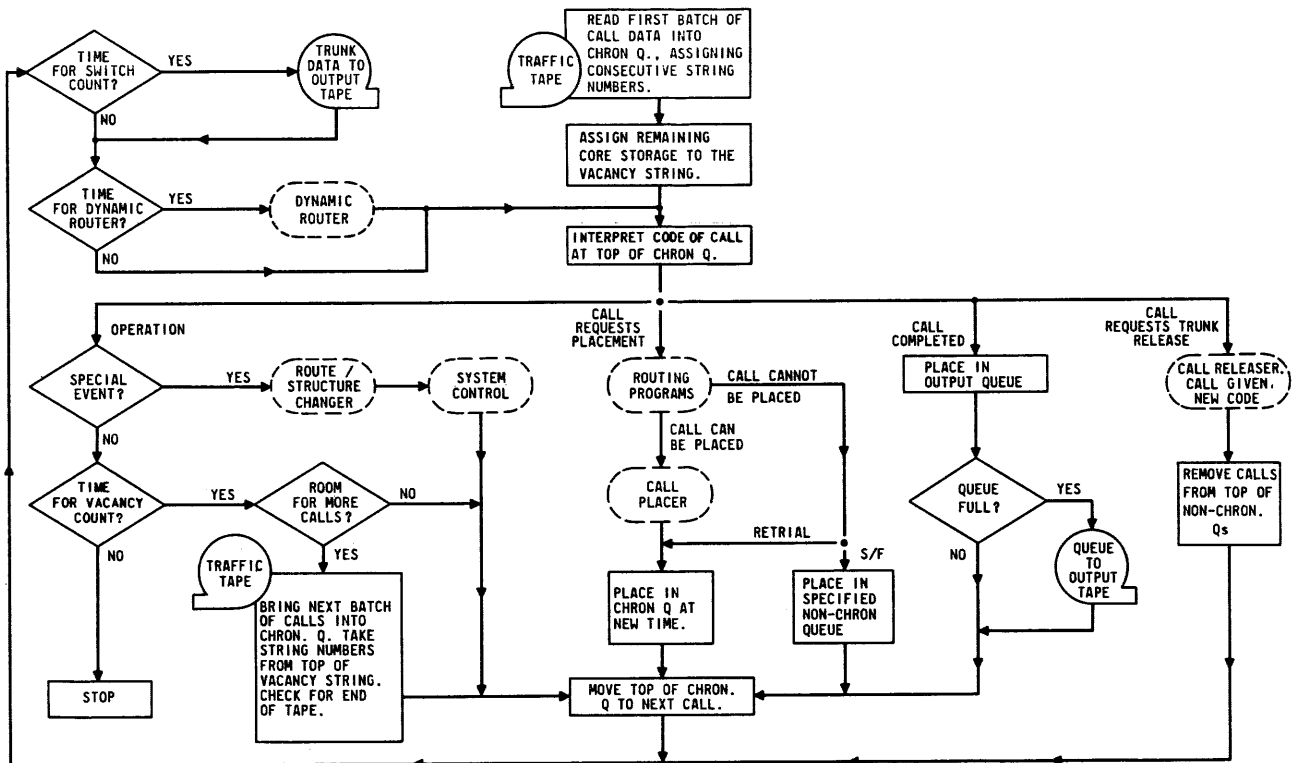


Figure 11. Operator Program.



with suitable use of macros, and the modifications were therefore not so easy to make as they might have been if a different language could have been used.

IX. LANGUAGE CONSIDERATIONS

As was mentioned earlier, UNISIM is characterized by the requirement to handle large numbers of simultaneous demands, using a logical structure which is of moderate complexity. It must also estimate quantities which occur with low probability, and therefore must operate rapidly and be of large capacity.

These requirements were met in part by sectioning the program, so that all parts of the program need not be in core simultaneously, which provided more space for the data. In addition, most of the program was written in

machine language in order to obtain maximum utilization of memory by means of tight packing and dynamic storage allocation, and high running speeds.

The simulator section was itself divided into subprograms, some of which are used very frequently (Direct Router—used for every call) and some of which are used infrequently (Alternate Router). Since it was necessary for all of these programs to access the same packed data, all were written in FAP, although this made later alterations somewhat cumbersome. Experience with the program quickly brought out the need for changing parts of the program as new situations presented themselves. The parts which needed changing were typically in the less frequently used portions of the simulator itself, such as the Alternate Router.

NODE PAIR	PER-CENT FRST DIR TRAF	PER-CENT OFFD ONLY A,B	PER-CENT OFFD ONLY A,C	OFFRD LOAD	OFFRD LOAD	OFFRD LOAD	OFFRD LOAD	OFFRD LOAD
				FRBM 1,000 T8 3000.000	FRBM 4000.000 T8 7000.000	FRBM 8000.000 T8 11000.000	FRBM 12000.000 T8 15000.000	FRBM 16000.000 T8 19000.000
1 2	50.	100.	100.	61.63	65.33	61.41	57.72	53.11
1 3	50.	100.	100.	2.91	3.08	2.90	2.73	2.51
1 4	50.	100.	100.	2.29	2.43	2.28	2.14	1.97
1 5	50.	100.	100.	0.20	0.21	0.20	0.19	0.18
1 6	50.	100.	100.	0.05	0.05	0.05	0.05	0.05
1 7	50.	100.	100.	0.07	0.07	0.07	0.07	0.06
1 8	50.	100.	100.	0.28	0.30	0.28	0.27	0.25
1 9	50.	100.	100.	0.09	0.10	0.09	0.09	0.08
1 10	50.	100.	100.	8.87	9.40	9.03	8.48	8.06
1 11	50.	100.	100.	0.08	0.08	0.08	0.08	0.07
1 12	50.	100.	100.	0.18	0.18	0.17	0.18	0.17
1 13	50.	100.	100.	0.25	0.26	0.25	0.26	0.24
1 14	50.	100.	100.	0.47	0.49	0.47	0.48	0.44
1 15	50.	100.	100.	0.04	0.04	0.04	0.04	0.04
1 16	50.	100.	100.	0.12	0.12	0.12	0.12	0.11
1 17	50.	100.	100.	0.15	0.16	0.15	0.15	0.14
1 18	50.	100.	100.	0.55	0.57	0.55	0.57	0.52
1 19	50.	100.	100.	0.82	0.85	0.82	0.84	0.78
1 20	50.	100.	100.	3.74	3.89	3.73	3.85	3.54
1 21	50.	100.	100.	0.95	0.99	0.95	0.98	0.90
1 22	50.	100.	100.	0.81	0.84	0.81	0.83	0.77
1 23	50.	100.	100.	0.93	0.97	0.93	0.96	0.88
1 24	50.	100.	100.	0.22	0.23	0.22	0.23	0.21
1 25	50.	100.	100.	0.38	0.40	0.38	0.39	0.36
1 26	50.	100.	100.	0.36	0.35	0.35	0.36	0.35
1 27	50.	100.	100.	0.08	0.08	0.08	0.08	0.08
1 28	50.	100.	100.	1.56	1.53	1.50	1.54	1.51
1 29	50.	100.	100.	1.90	1.86	1.82	1.88	1.84
1 30	50.	100.	100.	0.96	0.94	0.92	0.95	0.93
1 31	50.	100.	100.	1.58	1.55	1.52	1.56	1.53
1 32	50.	100.	100.	0.43	0.42	0.41	0.43	0.42
1 33	50.	100.	100.	0.56	0.55	0.54	0.55	0.54
1 34	50.	100.	100.	0.65	0.64	0.62	0.64	0.63
2 3	50.	100.	100.	46.89	49.70	46.72	43.92	40.40
2 4	50.	100.	100.	65.03	68.93	64.80	60.91	56.04
2 5	50.	100.	100.	4.48	4.75	4.56	4.29	4.07
2 6	50.	100.	100.	1.57	1.66	1.60	1.50	1.43
2 7	50.	100.	100.	1.74	1.84	1.77	1.66	1.58
2 8	50.	100.	100.	3.39	3.59	3.45	3.24	3.08
2 9	50.	100.	100.	3.88	4.11	3.95	3.71	3.53
2 10	50.	100.	100.	158.81	168.34	161.61	151.91	144.31
2 11	50.	100.	100.	2.11	2.24	2.15	2.02	1.92
2 12	50.	100.	100.	2.28	2.23	2.19	2.26	2.21
2 13	50.	100.	100.	3.06	3.18	3.06	3.15	2.90
2 14	50.	100.	100.	2.88	3.00	2.88	2.96	2.72
2 15	50.	100.	100.	0.82	0.85	0.82	0.84	0.78
2 16	50.	100.	100.	1.54	1.60	1.54	1.58	1.46
2 17	50.	100.	100.	1.70	1.77	1.70	1.75	1.61
2 18	50.	100.	100.	6.67	6.94	6.66	6.86	6.31
2 19	50.	100.	100.	8.38	8.72	8.37	8.62	7.93
2 20	50.	100.	100.	27.27	28.36	27.23	28.04	25.80
2 21	50.	100.	100.	10.78	11.21	10.76	11.09	10.20

Figure 12. Offered Loads as Used in Generation of Input Tape.

INITIAL ALTERNATE ROUTING TABLES											
NODE PAIR	MAXIMUM LINKS DIRECT	ROUTING LOW TO HIGH				ROUTING HIGH TO LOW				LENGTH OF LINK	
		NODE	LINKS	NODE	LINKS	NODE	LINKS	NODE	LINKS		
1 2	5	4	2	10	2	4	2	10	2	79.61	
1 3	5	0	0	0	0	5	3	0	0	243.44	
1 4	5	2	2	4	2	2	2	4	2	174.70	
1 5	5	0	0	0	0	9	3	0	0	0.	
1 6	5	2	2	10	2	2	2	10	2	0.	
1 7	5	10	2	2	3	10	2	0	0	0.	
1 8	5	4	3	0	0	0	0	0	0	0.	
1 9	5	10	2	2	2	10	2	2	2	0.	
1 10	5	4	3	0	0	11	3	7	3	0.	
1 11	5	4	2	2	2	4	2	2	2	0.	
1 12	5	3	2	10	2	3	2	10	2	225.23	
1 13	5	2	2	4	2	2	2	5	3	0.	
1 14	5	0	0	0	0	4	2	0	0	0.	
1 15	5	10	2	2	2	10	2	2	2	0.	
1 16	5	4	3	0	0	8	3	0	0	0.	
1 17	5	2	2	10	3	2	2	13	3	0.	
1 18	5	4	4	20	2	20	2	21	3	0.	
1 19	5	3	3	4	0	19	3	0	0	0.	
1 20	5	2	2	10	2	2	2	10	2	0.	
1 21	5	3	3	0	0	16	4	0	0	0.	
1 22	5	20	2	4	3	2	2	13	3	0.	
1 23	5	3	3	0	0	10	2	18	3	0.	
1 24	5	2	2	10	3	15	4	0	0	0.	
1 25	5	2	3	10	3	18	3	16	4	0.	
1 26	5	20	2	4	4	25	3	20	2	0.	
1 27	5	3	3	0	0	21	3	0	0	0.	
1 28	5	2	3	10	3	18	3	13	3	0.	
1 29	5	20	2	4	4	25	3	15	4	0.	
1 30	5	3	3	0	0	20	2	0	0	0.	
1 31	5	10	2	2	3	10	2	8	3	0.	
1 32	5	4	3	0	0	7	3	13	3	0.	
1 33	5	2	2	10	2	2	2	10	2	0.	
1 34	5	4	3	20	2	17	3	8	3	0.	
1 35	5	3	3	0	0	13	3	0	0	0.	
1 36	5	2	2	10	2	2	2	10	2	0.	
1 37	5	20	2	4	3	17	3	18	3	0.	
1 38	5	3	3	0	0	20	2	0	0	0.	
1 39	5	2	2	10	2	2	2	10	2	762.35	
1 40	5	4	2	3	2	17	3	21	3	0.	
1 41	5	0	0	0	0	24	3	0	0	0.	
1 42	5	2	2	10	2	2	2	10	2	0.	
1 43	5	20	2	4	3	17	3	20	2	0.	
1 44	5	3	3	0	0	24	3	0	0	0.	
1 45	5	10	2	4	3	21	3	10	2	0.	
1 46	5	3	3	0	0	23	3	0	0	0.	
1 47	5	2	2	10	2	2	2	10	2	0.	
1 48	5	20	2	4	3	17	3	24	3	0.	
1 49	5	3	3	0	0	25	3	0	0	0.	
1 50	5	2	2	10	3	2	2	25	3	0.	

Figure 13. Initial Alternate Routing Tables.

The frequently used sections, such as the Direct Router or the Operator, are basic to the structure either of the system or of the simulator, and changes in these would normally be of a sufficiently fundamental nature to require a substantially new approach. This type of program, then, need not be easily alterable.

This sort of experience leads rather naturally to a categorization of simulation programs according to the following classifications:

- (1) Basic subroutines which are called by virtually every transaction. These programs should be efficient but need not be easily modified.
- (2) Subroutines which specify the logic to be followed under more unusual cir-

cumstances, such as when congestion is encountered. Such routines are critical to the problem, since it is the behavior of the system under such circumstances that is normally the question under investigation. The logic followed in these cases ordinarily can be varied and is under constant assault as new operating procedures are invented and must be tested. These programs, however, must have access to the same data structure as the basic subroutines of class (1).

- (3) Programs which perform such functions as preparing the inputs or processing the data. These programs are normally reached only at infrequent times, and can access data which is buffered and altered from the basic data

in the simulation. They also may be largely arithmetic in function as opposed to the logical operations generally performed by the programs within the simulation.

It would be desirable in the development of future simulation languages if the capability for using different existing languages for certain sections of the program could be provided. For example, heavily used portions of the program may be best written in a machine language in order to obtain maximum efficiency in speed and space. Others may best be written in a familiar language which is suitable for arithmetic operations (although if a simulation language can incorporate extensive arithmetic functions, this would be useful). The third type of program, however, (type (2))

could well be written in some intermediate type of simulation language. The characteristics of the language would have to include at least the following:

- (a) It be "natural" in the sense that it be reasonably easy to understand and use, and has a documenting capability to facilitate program changes.
- (b) It be capable of interfacing with a program written in machine language and perhaps with a commonly used compiler language such as Fortran. In particular, it must be capable of operating on the same data as do machine language sub-routines, with full capability for flexible bit packing and accessing of information which is stored in dynamically changing arrays.

SWITCH COUNT TAPE REDUCTION - MEAN - FOR REPORTING INTERVAL NUMBER 1									
PROCESSING INTERVAL NUMBERS 2 THROUGH 6 NUMBER OF SWITCH COUNTS 50									
BEGINNING TIME IS 2000.000					ENDING TIME IS 12000.000				
NODE PAIR	TOTAL TRUNKS	TRUNKS RESC FULL	TOTAL CARRIED LOAD	CARRIED FULL ACCESS	CARRIED LIMITED ACCESS	PERCENT OCCUPANCY	PERCENT FULL ACCESS	PERCENT LIMITED ACCESS	
1 2	77	0	73.200	67.320	5.880	95.065	91.998	8.002	
1 3	7	0	4.320	3.180	1.140	61.714	73.741	26.259	
1 4	23	0	17.720	8.580	9.140	77.043	48.243	51.757	
1 10	18	0	14.480	11.420	3.060	80.444	78.909	21.091	
1 20	7	0	5.140	4.940	0.200	73.429	96.235	3.765	
2 3	43	0	39.740	38.940	0.800	92.419	97.977	2.023	
2 4	79	0	74.860	62.680	12.180	94.759	83.786	16.214	
2 5	31	0	22.380	13.520	8.860	72.194	60.731	39.269	
2 8	8	0	4.800	3.860	0.940	60.000	80.585	19.415	
2 9	9	0	5.880	4.400	1.480	65.333	74.993	25.007	
2 10	169	0	163.560	150.160	13.400	96.781	91.816	8.184	
2 11	5	0	1.960	1.860	0.100	39.200	95.425	4.575	
2 12	4	0	2.260	2.100	0.160	56.500	93.642	6.358	
2 13	18	0	14.760	8.620	6.140	82.000	58.581	41.419	
2 14	10	0	6.740	5.840	0.900	67.400	86.718	13.282	
2 18	11	0	9.020	8.840	0.180	82.000	98.022	1.978	
2 19	9	0	7.180	7.180	0.	79.778	100.000	0.	
2 20	26	0	16.380	14.980	1.400	63.000	91.517	8.483	
2 21	16	0	10.660	10.620	0.040	66.625	99.658	0.342	
2 22	7	0	5.160	5.160	0.	73.714	100.000	0.	
2 23	14	0	12.680	12.680	0.	90.571	100.000	0.	
2 24	5	0	3.360	2.720	0.640	67.200	81.828	18.172	
2 25	7	0	4.920	4.800	0.120	70.286	97.570	2.430	
2 29	33	0	30.320	29.980	0.340	91.879	98.892	1.108	
2 30	11	0	9.020	9.020	0.	82.000	100.000	0.	
2 31	18	0	15.080	14.260	0.820	83.778	94.433	5.567	
2 32	7	0	4.660	4.500	0.160	66.571	96.538	3.462	
2 33	7	0	4.800	4.000	0.800	68.571	83.042	16.958	
2 34	8	0	6.080	6.080	0.	76.000	100.000	0.	
3 4	57	0	52.400	43.620	8.980	91.930	82.921	17.079	
3 9	16	0	11.460	7.680	3.780	71.625	66.770	33.230	
3 10	24	0	20.920	20.460	0.460	87.167	97.865	2.135	
3 20	11	0	9.500	9.380	0.120	86.364	98.818	1.182	
3 29	10	0	7.320	7.320	0.	73.200	100.000	0.	
4 9	26	0	22.280	16.740	5.540	85.692	75.303	24.697	
4 10	48	0	47.440	34.200	8.240	88.417	80.658	19.342	
4 20	11	0	7.920	6.860	1.060	72.000	86.798	13.202	
4 29	13	0	9.080	8.580	0.500	69.846	94.457	5.543	
5 10	46	0	38.620	29.020	9.600	83.957	75.226	24.774	
6 10	37	0	25.880	25.880	0.	69.946	100.000	0.	
7 8	13	0	6.820	6.100	0.720	52.462	89.325	10.675	
7 10	21	0	15.180	14.980	0.200	72.286	98.638	1.362	
7 17	19	0	12.480	12.220	0.260	65.684	97.873	2.127	
8 10	35	0	29.140	26.080	3.060	83.257	89.488	10.512	
8 11	21	0	14.060	12.540	1.520	66.952	89.465	10.535	
8 17	21	0	14.700	13.100	1.600	70.000	89.154	10.846	
8 18	19	0	12.320	11.440	0.880	64.842	92.916	7.084	
8 25	10	0	6.440	4.780	1.660	64.400	74.326	25.674	
9 10	34	0	27.920	24.880	3.040	82.118	89.096	10.904	
10 11	30	0	23.640	22.520	1.120	78.800	95.243	4.757	
10 13	18	0	14.020	10.240	3.780	77.889	72.965	27.035	

Figure 14. Switch Count Tape Reduction—Mean—For Reporting Interval Number.

CALL TAPE REDUCTION - MEAN - FOR REPORTING INTERVAL NUMBER 1																
PROCESSING INTERVAL NUMBERS				2 THROUGH				6				NUMBER OF SWITCH COUNTS	50			
BEGINNING TIME IS		2000.000											ENDING TIME IS		12000.000	
ORIG	DEST	BLCK PRBB.	AVE. DELAY	DELAY DISTRIBUTION				AVE. NO. LINKS	LINK DISTRIBUTION							
				0.5	1.0	1.5	2.0		1	2	3	4	5	6	7	
1	2	0.051	0.	0.	0.	0.	0.	1.189	0.844	0.127	0.025	0.004	0.	0.	0.	0.
1	3	0.027	0.	0.	0.	0.	0.	1.211	0.843	0.104	0.053	0.	0.	0.	0.	0.
1	4	0.080	0.	0.	0.	0.	0.	1.168	0.865	0.109	0.019	0.007	0.	0.	0.	0.
1	5	0.	0.	0.	0.	0.	0.	1.700	0.	0.700	0.100	0.	0.	0.	0.	0.
1	6	0.	0.	0.	0.	0.	0.	0.900	0.	0.300	0.100	0.	0.	0.	0.	0.
1	7	0.	0.	0.	0.	0.	0.	1.500	0.	0.300	0.300	0.	0.	0.	0.	0.
1	8	0.050	0.	0.	0.	0.	0.	2.233	0.	0.767	0.233	0.	0.	0.	0.	0.
1	9	0.	0.	0.	0.	0.	0.	1.300	0.	0.500	0.100	0.	0.	0.	0.	0.
1	10	0.049	0.	0.	0.	0.	0.	1.127	0.888	0.100	0.010	0.002	0.	0.	0.	0.
1	11	0.	0.	0.	0.	0.	0.	1.600	0.	0.800	0.	0.	0.	0.	0.	0.
1	12	0.	0.	0.	0.	0.	0.	2.267	0.	0.733	0.267	0.	0.	0.	0.	0.
1	13	0.140	0.	0.	0.	0.	0.	2.350	0.	0.650	0.350	0.	0.	0.	0.	0.
1	14	0.072	0.	0.	0.	0.	0.	2.397	0.	0.703	0.197	0.100	0.	0.	0.	0.
1	16	0.200	0.	0.	0.	0.	0.	2.100	0.	0.	0.400	0.100	0.100	0.	0.	0.
1	17	0.	0.	0.	0.	0.	0.	2.467	0.	0.733	0.067	0.200	0.	0.	0.	0.
1	18	0.	0.	0.	0.	0.	0.	2.275	0.	0.825	0.100	0.050	0.025	0.	0.	0.
1	19	0.073	0.	0.	0.	0.	0.	2.233	0.	0.800	0.166	0.033	0.	0.	0.	0.
1	20	0.006	0.	0.	0.	0.	0.	1.293	0.741	0.231	0.022	0.006	0.	0.	0.	0.
1	21	0.025	0.	0.	0.	0.	0.	2.131	0.	0.869	0.131	0.	0.	0.	0.	0.
1	22	0.082	0.	0.	0.	0.	0.	2.210	0.	0.830	0.130	0.040	0.	0.	0.	0.
1	23	0.	0.	0.	0.	0.	0.	2.341	0.	0.783	0.155	0.	0.062	0.	0.	0.
1	24	0.	0.	0.	0.	0.	0.	1.733	0.	0.667	0.133	0.	0.	0.	0.	0.
1	25	0.101	0.	0.	0.	0.	0.	2.258	0.	0.792	0.158	0.050	0.	0.	0.	0.
1	26	0.025	0.	0.	0.	0.	0.	2.662	0.	0.	0.538	0.262	0.	0.	0.	0.
1	27	0.	0.	0.	0.	0.	0.	1.800	0.	0.	0.600	0.	0.	0.	0.	0.
1	28	0.073	0.	0.	0.	0.	0.	3.298	0.	0.012	0.689	0.286	0.012	0.	0.	0.
1	29	0.024	0.	0.	0.	0.	0.	2.316	0.	0.772	0.139	0.088	0.	0.	0.	0.
1	30	0.	0.	0.	0.	0.	0.	2.351	0.	0.705	0.240	0.056	0.	0.	0.	0.
1	31	0.010	0.	0.	0.	0.	0.	2.356	0.	0.674	0.306	0.010	0.010	0.	0.	0.
1	32	0.	0.	0.	0.	0.	0.	2.317	0.	0.750	0.183	0.067	0.	0.	0.	0.
1	33	0.033	0.	0.	0.	0.	0.	2.100	0.	0.900	0.100	0.	0.	0.	0.	0.
1	34	0.100	0.	0.	0.	0.	0.	2.369	0.	0.681	0.294	0.	0.025	0.	0.	0.
2	3	0.031	0.	0.	0.	0.	0.	1.226	0.811	0.158	0.027	0.005	0.	0.	0.	0.
2	4	0.038	0.	0.	0.	0.	0.	1.201	0.837	0.132	0.024	0.006	0.000	0.	0.	0.
2	5	0.056	0.	0.	0.	0.	0.	1.076	0.937	0.053	0.006	0.004	0.	0.	0.	0.
2	6	0.025	0.	0.	0.	0.	0.	2.171	0.	0.829	0.171	0.	0.	0.	0.	0.
2	7	0.011	0.	0.	0.	0.	0.	2.094	0.	0.906	0.094	0.	0.	0.	0.	0.
2	8	0.	0.	0.	0.	0.	0.	1.078	0.928	0.066	0.006	0.	0.	0.	0.	0.
2	9	0.100	0.	0.	0.	0.	0.	1.146	0.904	0.052	0.039	0.005	0.	0.	0.	0.
2	10	0.044	0.	0.	0.	0.	0.	1.142	0.868	0.122	0.009	0.	0.001	0.	0.	0.
2	11	0.011	0.	0.	0.	0.	0.	1.035	0.976	0.012	0.012	0.	0.	0.	0.	0.
2	12	0.	0.	0.	0.	0.	0.	1.225	0.861	0.094	0.019	0.009	0.017	0.	0.	0.
2	13	0.	0.	0.	0.	0.	0.	1.282	0.816	0.107	0.056	0.021	0.	0.	0.	0.
2	14	0.	0.	0.	0.	0.	0.	1.139	0.899	0.081	0.	0.019	0.	0.	0.	0.
2	15	0.025	0.	0.	0.	0.	0.	2.794	0.	0.600	0.432	0.139	0.029	0.	0.	0.
2	16	0.019	0.	0.	0.	0.	0.	2.420	0.	0.723	0.168	0.075	0.034	0.	0.	0.
2	17	0.011	0.	0.	0.	0.	0.	2.121	0.	0.887	0.106	0.008	0.	0.	0.	0.
2	18	0.003	0.	0.	0.	0.	0.	1.271	0.794	0.155	0.040	0.008	0.002	0.	0.	0.
2	19	0.008	0.	0.	0.	0.	0.	1.361	0.750	0.178	0.063	0.019	0.010	0.	0.	0.
2	20	0.002	0.	0.	0.	0.	0.	1.026	0.977	0.020	0.003	0.	0.	0.	0.	0.
2	21	0.	0.	0.	0.	0.	0.	1.057	0.947	0.049	0.004	0.	0.	0.	0.	0.
2	22	0.042	0.	0.	0.	0.	0.	1.209	0.829	0.139	0.024	0.007	0.	0.	0.	0.

Figure 15. Call Tape Reduction—Mean—For Reporting Interval Number.

- (c) The implementation should preferably be monitor independent, so that it can be used at installations with nonstandard systems. Failing this, the structure of the translator should be sufficiently well documented so that modifications can be made locally.
- (d) Although not essential, if the language is to be used for an entire program, as might be the case for some smaller problems, I/O and arithmetic functions should be included. Special subroutines, such as programs to assemble delay distributions, would be useful in this respect.

The above statement of certain desired characteristics of a simulation language in effect sets a rather limited goal, on the grounds that

such a program would provide most of the advantages of a full new language, while at the same time materially reducing the complexity of the program. This would improve the likelihood of its being learned and used, which is the ultimate test of value.

It also would not be excluded from application on grounds of space or speed, since critical sections can be done in other languages, and full bit manipulation capability would be provided.

These requirements are quite general in nature, but we do not believe that detailed questions of program structure can readily be deduced from a specific problem of the sort which prompted the writing of UNISIM, nor do we expect that this is an appropriate vehicle for such discussions.

The essential point, however, is that the programmer have available the full capability of the machine if necessary, and a simulation language which is to have application to problems of this sort must make this possible. To our knowledge no language which presently exists and is in general use, provides this capability.

X. ACKNOWLEDGMENT

The programming of the various sections of UNISIM was accomplished by the following team at Bell Laboratories: Miss G. T. Watling,

Mrs. E. E. Bailey, Miss S. A. Switch, and Mrs. A. Sheehan.

REFERENCES

1. "Some Traffic Characteristics of Communications Networks with Automatic Alternate Routing," Bell System Technical Journal, Vol. 41, pp. 769-796, March 1962, J. H. WEBER.
2. "A Simulation Study of Routing and Control in Communications Networks," Bell System Technical Journal, November 1964, J. H. WEBER.

OVERALL SYSTEM MEANS - SWITCH COUNT DATA - REPORTING INTERVALS												
BEGINNING TIME	ENDING TIME	TOTAL TRUNKS	TRUNKS RESD FULL	SUM CARRIED LOAD	CARRIED FULL ACCESS	CARRIED LIMITED ACCESS	PERCENT OCCUPANCY	PERCENT FULL ACCESS	PERCENT LIMITED ACCESS			
1.00	3000.00	6663	0	5098.130	4695.531	402.600	76.514	92.103	7.897			
4000.00	7000.00	6663	0	4972.997	4608.397	364.600	74.636	92.668	7.332			
8000.00	11000.00	6663	0	5178.997	4765.330	413.667	77.728	92.013	7.987			
12000.00	15000.00	6663	0	5365.530	4879.930	485.600	80.527	90.950	9.350			
16000.00	19000.00	6663	0	5478.197	4957.397	520.800	82.218	90.493	9.507			

OVERALL SYSTEM STANDARD DEVIATIONS - SWITCH COUNT DATA - REPORTING INTERVALS												
BEGINNING TIME	ENDING TIME	TOTAL TRUNKS	TRUNKS RESD FULL	SUM CARRIED LOAD	CARRIED FULL ACCESS	CARRIED LIMITED ACCESS						
1.00	3000.00	6663	0	339.522	291.658	81.048						
4000.00	7000.00	6663	0	235.773	214.849	60.796						
8000.00	11000.00	6663	0	229.917	211.104	66.274						
12000.00	15000.00	6663	0	213.305	196.565	68.775						
16000.00	19000.00	6663	0	206.315	199.628	60.410						

OVER-ALL MEANS-CALL DATA TAPE																
BEGINNING TIME	ENDING TIME	BLCK PRBB.	AVE. DELAY	DELAY DISTRIBUTION					AVE. NO. LINKS	LINK DISTRIBUTION						
				0.5	1.0	1.5	2.0	1		2	3	4	5	6	7	
1.00	3000.00	0.010	0.	0.	0.	0.	0.	0.	1.109	0.892	0.101	0.004	0.	0.	0.	0.
4000.00	7000.00	0.009	0.	0.	0.	0.	0.	0.	1.103	0.897	0.097	0.004	0.	0.	0.	0.
8000.00	11000.00	0.008	0.	0.	0.	0.	0.	0.	1.113	0.888	0.106	0.004	0.	0.	0.	0.
12000.00	15000.00	0.012	0.	0.	0.	0.	0.	0.	1.124	0.878	0.115	0.005	0.	0.	0.	0.
16000.00	19000.00	0.024	0.	0.	0.	0.	0.	0.	1.128	0.874	0.119	0.006	0.	0.	0.	0.

OVER-ALL INTERVAL STANDARD DEVIATIONS - CALL DATA																
BEGINNING TIME	ENDING TIME	BLCK PRBB.	AVE. DELAY	DELAY DISTRIBUTION					AVE. NO. LINKS	LINK DISTRIBUTION						
0.5	1.0	1.5	2.0	1	2	3	4	5		6	7					
1.00	3000.00	0.013	0.	0.	0.	0.	0.	0.	0.063	0.033	0.041	0.020	0.	0.	0.	0.
4000.00	7000.00	0.014	0.	0.	0.	0.	0.	0.	0.061	0.030	0.038	0.018	0.	0.	0.	0.
8000.00	11000.00	0.013	0.	0.	0.	0.	0.	0.	0.060	0.032	0.039	0.019	0.	0.	0.	0.
12000.00	15000.00	0.015	0.	0.	0.	0.	0.	0.	0.062	0.032	0.040	0.022	0.	0.	0.	0.
16000.00	19000.00	0.017	0.	0.	0.	0.	0.	0.	0.059	0.030	0.037	0.019	0.	0.	0.	0.

MISERY FACTORS																
BEGINNING TIME	ENDING TIME	BLCK PRBB.	AVE. DELAY	DELAY DISTRIBUTION					AVE. NO. LINKS	LINK DISTRIBUTION						
0.5	1.0	1.5	2.0	1	2	3	4	5		6	7					
1.00	3000.00	0.034	0.	0.	0.	0.	0.	0.	0.215	0.207	0.189	0.037	0.	0.	0.	0.
4000.00	7000.00	0.026	0.	0.	0.	0.	0.	0.	0.220	0.211	0.194	0.040	0.	0.	0.	0.
8000.00	11000.00	0.029	0.	0.	0.	0.	0.	0.	0.215	0.205	0.188	0.037	0.	0.	0.	0.
12000.00	15000.00	0.034	0.	0.	0.	0.	0.	0.	0.219	0.208	0.190	0.040	0.	0.	0.	0.
16000.00	19000.00	0.044	0.	0.	0.	0.	0.	0.	0.217	0.205	0.187	0.038	0.	0.	0.	0.

Figure 16. Overall System Means—Switch Count Data—Reporting Intervals.



# THE DATA PROCESSING SYSTEM SIMULATOR (DPSS)

## (SP-1299/000/01)

*Michael I. Youchah, Donald D. Rudie, and Edward J. Johnson*  
*System Development Corporation*  
*Paramus, New Jersey*

### 1.0 INTRODUCTION

The Data Processing System Simulator (DPSS) is a general purpose computer program that can be used for the evaluation of a proposed new design or a modification to an existing design of a data processing system prior to making equipment selections or performing any significant computer program design. The DPSS can also be used to provide guidance in the design and development of a data processing system during the detailed design stages.

The DPSS was initially designed to meet the needs of analyzing and developing the data processing requirements of the Project 465L Strategic Air Command Control System (SACCS). It has subsequently been generalized to permit its application to other systems in various stages of design and development. Results have shown the usefulness of the DPSS in Project 465L (SACCS) and, in a preliminary manner, its usefulness on the Space Surveillance Project and New York State Identification and Intelligence System. In all three cases, original concepts about the system's potential performance were evaluated and new and significant guidance and information obtained as a result of the use of the DPSS. In the case of the New York State System, the DPSS results showed a whole class of computers to be inadequate for the job.

The DPSS can be used to determine the sensitivity of a data processing system's performance to various system loading or design parameters. In addition, the total system design, including the software and equipment portions, can be subjected to a rigorous analysis and evaluation early in the design process so that key decisions can be made in the areas of:

1. The kind of equipment to be used.
2. The number of each type of equipment.
3. The kind of data processing discipline and strategy required.
4. The projected performance of the system under varying loads.
5. The system's maximum capacity.
6. The system's ability to respond as a function of loading, capacity, and environment.

### 2.0 DEVELOPMENT OF THE DATA PROCESSING SYSTEM SIMULATOR

In its development, the Data Processing System Simulator has used a higher order simulation language similar to those which have been developed previously for general simulation. However, unlike other techniques, a single combination of these higher order language macro instructions is used in a single logical arrangement permitting the representation of a wide variety of possible data processing system configurations and processing rules with no additional programming or design.

As a consequence, the necessity of writing special "event" programs or subroutines, as is usually required with the use of simulation languages, has been eliminated. Further, the flow diagramming, design, coding, compiling, and checkout of each simulator or subroutine created by the use of simulation languages have been totally eliminated. The result is a considerably shortened time required to produce a reliable model of the data processing system to be investigated.

### 2.1 DPSS Description

The following sections contain a general description of the DPSS and a sample problem illustrating how it can be used.

#### 2.1.1 General Characteristics

The Data Processing System Simulator requires approximately 1,500 instructions written in JOVIAL, a high-order programming language developed by the System Development Corporation.

The DPSS was initially designed to run on the AN/FSQ-32-V computer which is described in Appendix V, however, a new and expanded version of the DPSS has been written to run on the IBM 7094.

The AN/FSQ-32-V version of the DPSS requires 15,000 core locations which are used to store the basic program and the parametric inputs used for each run.

#### 2.1.2 Purposes and Limitations

The DPSS is used to represent (a) the inputs to the system, that is, those message units or informational units which are to be entered into the system, at either local or remote locations, (b) the processing performed by the computer on each input, and (c) the outputs generated by the processing portion of the system based upon the inputs. Processing includes the buffering and the retention of the messages prior to processing. It also includes the specification of the time to load the system with the necessary programs and environment to handle the message and the time to unload and the time to operate the actual program as well as the data preparation and data presentation functions.

### 3.0 DPSS OPERATION—GENERAL

To understand the functioning of the DPSS, a general idea of the key features of the way a possible Data Processing Central (DPC) cycle operates will be discussed. The system configuration to be considered in the example will employ a discontinuous cycle in which interleaving and interrupting are permitted. The material for this example is drawn from experience with several systems and does not represent the design of any single system. See the Glossary of Terms, Appendix III, for an explanation of key terms used in the following paragraphs.

#### 3.1 Data Processing Central (DPC) Operations

The input messages which arrive from various local and remote locations are batched at the DPC (Figure 1) and held until certain criteria for one or more of the batches are reached or exceeded. The specific nature of batching will be discussed subsequently. Once a batch criterion has been exceeded, the Executive program which controls all DPC functions then initiates the processing of the input messages which have been batched. In so doing, it calls in from auxiliary storage the necessary environment and programs for the operation of the processing portion of the DPC cycle. When the processing has been completed, a set of output programs extracts the appropriate data from the data files and prepares the required system outputs.

#### 3.2 Input Batching

An input message batch is characterized by three items, time, size, and interrupt, the latter having two sub-items (see Figure 2). The "time" item indicates that a particular message or group of messages will be accumulated for a given length of time before an indication (cycle initiation request) is given to the Execu-



Figure 1. DPC Cycle Operation.



tive or master program that a cycle should start. The second item is the "batch" size. The DPC will collect message(s) until a predetermined number of them have been accumulated. This given number of message(s) must be accumulated in less time than the batch time in order to cause a request for the initiation of a DPC cycle to be made because of "size."

Once either of these two values (time or size) has been exceeded, it must then be determined if the interrupt feature (the third item) associated with this particular batch is set to "yes" or "no," and if set to "yes" whether the "immediate" or "wait" option is set. There are two cases to consider here: (1) when the DPC cycle initiation request occurs during the operation of an interleaved subsystem, and (2) when the DPC cycle initiation request occurs during the operation of DPC cycle in progress.

When any cycle initiation request occurs during the operation of an interleaved subsystem, the Executive interrupts the interleaved subsystem at the earliest possible moment, regardless of the interrupt setting of the batch, and initiates a DPC cycle. If the cycle initiation request occurs during an on-going DPC cycle, then three things can happen, depending on the setting of the interrupt item and interrupt option. If the interrupt item is set to "no" for a batch which causes a cycle initiation request to be generated, then an indication will be given to the Executive program that the batch's limits of time or size have been exceeded but the Executive will not interrupt the cycle in progress. A new cycle will be initiated as soon as the present one has been completed.

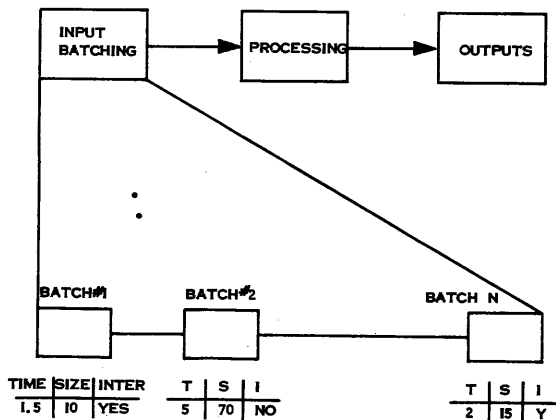


Figure 2. Batching Concept.

If the interrupt item has been set to "yes" and the interrupt option to "immediate," then the Executive program will initiate a new cycle immediately (possibly with certain programming constraints). If the interrupt item had been set to "yes" and the interrupt option to "wait," then the current cycle will be interrupted when the priority of the message causing the interrupt request is equal to or higher than the priority of the message being processed in the current cycle.

The DPSS permits the establishment of as many batches as are required for efficient system operation, assignment and modification of batch characteristics, and the assignment of inputs to each batch.

### 3.3 DPC Task Processing

Messages are processed by tasks within the DPC Program System. The tasks are sequenced according to the priority of the message being processed (see Figure 3). One of the tasks provides outputs from the system upon requests (display requests). This task is shown as the last task in Figure 3, however, there are many logical places where the task could be processed.

When a DPC cycle begins, all of the messages that have been collected in all of the batches up to the time that the cycle begins are transferred from the batches to the task processing area. In the task processing area, the messages lose their batch identity and are processed according to the task sequence.

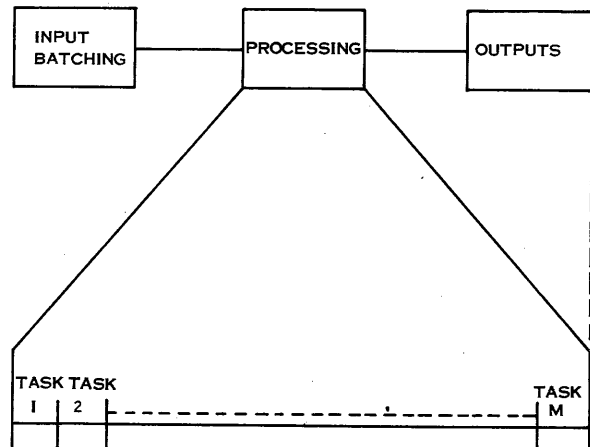


Figure 3. DPC Task Processing.

### 3.4 Input Data for Each Simulation Run

The use of the DPSS requires the definition of the program system configuration and the assignment of values for the system parameters as input data. The following set of input data are typical for each run of the DPSS. The sample values (placed within parentheses or in tables) are used to describe a system that will be simulated as an example. Note that if a system does not have some of the characteristics described in the inputs, e.g., batching, then that information can be left out of the input data.

1. The length of the period to be simulated—(3600 seconds).
2. The number of times that this test is to be repeated under the same operating conditions, referred to as the number of cycles in the test—(1).
3. The messages to be used in the test—(A, B, C, D, E, F).
4. The number of each type of message that will arrive at the DPC. This number is not an absolute number; rather, it determines the relative frequency of each particular message arriving at the DPC. This number is used in conjunction with the traffic rate:
 

(A - 10	D - 31)
(B - 22	E - 5)
(C - 19	F - 18)
5. The batch criteria for each batch and the messages that are collected in each batch.
  - a. Time
  - b. Size
  - c. Interrupt
    - 1) Immediate
    - 2) Wait

Batch	Message	Time Criteria	Size Criteria	Interrupt Option
1	A	0 Sec.	1	Immediate
2	B, D	5 Sec.	3	Wait
3	C, E, F	10 Sec.	4	No

6. System Tasks—This is a list of the tasks or jobs that the DPC Program System is required to do. The word “job” or “task”

means a collection of programs that are used to process a message or set of messages. The performance of a task is measured as follows:

- a. “Load Time”—The time it takes the environmental data tables and the operating programs to be transferred from the auxiliary storage to core memory.
  - b. “Operate Time”—The time it takes after the environmental data programs are in core for the programs themselves to process the messages to assess the intelligence they contain. In both of these cases, the distributions of the times to be used for this task and the proper parameter(s) for the distribution must be specified. (See Table I.)
7. Message-Task Relationship—This indicates the messages that are processed by each task. The DPSS will accept any message-task relationship.
 

Message	Task
A	1
C	2
B, D	3
E, F	4
  8. Message-Display Relationship and the Probability of the Forced Display—This is the relationship between messages and forced displays. It indicates which display(s) may be forced as the result of the message's being processed. Any message-display relationship may be established and tested. For each message-display relationship, the probability of a display's being forced is the conditional probability that the display will be forced given that the message has been processed. If a display is forced by more than one message, the probability of a display's being forced may be different for each message. (See Table II.)
  9. Task Sequence—This indicates the order in which the tasks operate. There is no restriction on the order in which the

TABLE I. SYSTEM TASK SUMMARY

TASK	LOAD		OPERATE	
	Distribution	Parameters	Distribution	Parameters
1	Uniform	Min 1 Sec	Exponential	Mean .001 Sec
		Max 6 Sec		
2	Exponential	Mean 4 Sec	Exponential	Mean .01 Sec
3	Uniform	Min 4 Sec	Uniform	Min. .02 Sec
		Max 5 Sec		Max .02 Sec
4	Triangular	Min 1 Sec	Exponential	Mean 1 Sec
		Peak 3 Sec		
		Max 8 Sec		
Additional Task which is per- formed whenever a system output is generated.	Uniform	Min 2 Sec	Exponential	Mean 5 Sec
		Max 4 Sec		

tasks operate. In addition, a task may operate more than once during a cycle.

Task	Sequence No.
1	1
3	2
2	3
4	4

- 10. Traffic Rate—This indicates the total volume of traffic per hour that is arriving at the DPC. By knowing the traffic rate and the relative frequency of each message, one is able to determine the expected number of each type of message that will arrive at the DPC within any time interval. Preplanned changes in traffic rates are permitted during the test—(750 messages/hour).
- 11. Maximum Interleave Time—This is the maximum time available for the interleave subsystem. This time could be zero, in which case the prime DPC cycle would operate in a cyclic fashion—(1800 Sec.).
- 12. The Bookkeeping Time for the Interleave Subsystem—This is the time that is given the interleave subsystem to store and save

pertinent data after a DPC cycle has been requested—(0 Sec.).

3.5 Results for Each DPSS Run

Each simulation run records and prints out the following items of information:

1. The arrival time of every message that is received by the computer.
2. The time that each cycle is requested.
3. The time that each cycle begins and the reason for initiation.

TABLE II. MESSAGE-DISPLAY RELATIONSHIPS

Message	Display	Probability of Forced
A	D1	.8
	D2	.3
B	None	
	D1	.6
	D3	.5
C	D4	.01
	None	
D	None	
E	D5	1
F	None	

4. The number of messages that have been collected in each batch at the beginning of each cycle.
5. The time that the processing of each individual message begins.
6. The time that the processing of each individual message is completed (processing complete means that the data files have been updated).
7. For every display that is forced during a cycle the message that forced the display, the time the message arrived and the time the display is forced are indicated.
8. For every cycle the minimum and maximum time required to process each type of message during the cycle.
9. A cumulative average processing time by cycle for each type of message in the system.
10. The time that each cycle ends.
11. A list of messages remaining to be processed at the end of each cycle.
12. The total number of messages that were received and processed by the computer during the simulation period.
13. For every type of message:
  - a. The number of messages that were expected to arrive during the simulation period.
  - b. The number of messages that actually arrived.
  - c. The number of messages processed.
  - d. The average waiting time for this type of message.
  - e. The average processing time for this type of message.
  - f. A histogram which indicates the percentage of messages whose processing time was in each of several one minute intervals; e.g., the percentage of messages whose processing time was between 0 and 1 minute, between 1 and 2 minutes, . . . , 30 to 31 minutes and 31 plus. The DPSS does not plot the histograms; rather, it supplies the data from which a plot can be drawn.
14. The number of times that each display type was forced.

#### 4.0 DETAILED EXAMPLE OF THE DPSS

The DPSS is described in detail with the aid of an example. This example will consist of a description of the system to be simulated and a detailed account of the arrival and processing of the first few messages. It should be noted that *all* of the features of the DPSS are *not* identified in this example. However, enough of them have been identified and described so that the reader can get a good understanding of the model by following the example.

##### 4.1 *Description of System Being Simulated*

The input parameters identified in Section 3.4 which describe the system being simulated are summarized in four tables: MESSAGE, BATCH, TASK, and SYSTEM tables. The MESSAGE table identifies the messages, their frequencies, the batches in which the messages are collected and the displays that are associated with the messages. The BATCH table identifies the batches, the batch criteria for each batch, and the message types that are associated with each batch. The TASK table is divided into two parts: LOAD and OPERATE. The LOAD part is used to specify the probability distribution and parameters that are used to determine the length of time it takes to transfer the processing programs and environment from auxiliary storage to core. The OPERATE part is used to specify the probability distribution and parameters that are used to determine the length of time that it takes to process each message after the processing programs and environmental data are in core. The SYSTEM table indicates the length of the test, the rate of the incoming messages and other pertinent information for the test.

See tables III, IV, V and VI for this example.

The ADD, CHANGE and DELETE columns are for the convenience of the user when he is making a series of runs and may wish to add or delete displays that are associated with a message, or to change the probability of a display being forced.

##### 4.2 *The DPSS in Operation*

The simulator generates the messages and "sends" them to the computer, in the same

TABLE III. MESSAGE TABLE

Message	Number Per Test	Batch	Forced Displays			Probability
			Display	Add	Change	
A	10	1	D1	x		.8
			D2	x		.3
B	22	2				
C	14	3	D1	x		.6
			D3	x		.5
			D4	x		.01
D	31	2				
E	5	3	D5	x		1.0
F	18	3				

fashion as it would receive them in an operating system. This system has six message types. During the simulation period they are "sent" to the computer at a rate of 750 messages/hour.

The incoming system messages arrive according to a Poisson distribution, i.e., the interarrival times of the system inputs are exponentially distributed.

If  $x$  is the interarrival time then an exponentially distributed interarrival time has the following form:

$$x = \frac{1}{a} \ln \left( \frac{1}{1 - y} \right) \quad (1)$$

Where  $1/a$  is the mean of the distribution and  $y$  is a uniformly distributed random number between 0 and 1 as determined by the use of a pseudo random number generator.

In our case, if we want  $x$  in seconds, then  $1/a = 3600/750$ . Suppose that the uniformly

distributed pseudo random number  $y = .362$  is picked, then

$$x = 3600/750 \ln \left( \frac{1}{1 - .362} \right) = 2.16 \text{ seconds}$$

This means that 2.16 seconds after the arrival of the last message (or from the beginning of the test if this is the first message to arrive) another message arrives at the central processor. The type of message that was received has not yet been determined. Each type of message is assigned a range on the interval (0, 1). The range is dependent on the relative frequency of the particular message type. To determine the message type, another random number is generated and checked to see which range it falls in.

It should be noted that the order sequence in which the ranges for the various message types are laid out on the unit interval does not in any way influence the simulation process, the reason being that the picked random number

TABLE IV. BATCH TABLE

Batch	Size	Time	Interrupt Option			Message Associated with Batch
			No	Immediate	Wait	
1	1	0 Sec		x		A
2	3	5 Sec			x	B, D
3	4	10 Sec	x			C, E, F

TABLE V

TASK TABLE - LOAD															
TASK NAME	TASK NO.	TASK SEQUENCE	MESSAGES ASSOC. WITH	DISTRIBUTION											
				EXPONENTIAL	ARBITRARY	CONTINUOUS	NORMAL		TRIANGULAR			UNIFORM			
				MEAN	CUM. PROB.	VALUE	MEAN	STANDARD DEVIATION	MIN.	MAX.	PEAK	MIN.	MAX.		
DSP. TSK	1	1	A	4 SEC									1 SEC	6 SEC	
	2	3	C											4 SEC	5 SEC
	3	2	B, D							1 SEC	3 SEC	8 SEC			
	4	4	E, F											2 SEC	4 SEC

TASK TABLE - OPERATE														
TASK NAME	TASK NO.	TASK SEQUENCE	MESSAGES ASSOC. WITH TASK	DISTRIBUTION										
				EXPONENTIAL	ARBITRARY	CONTINUOUS	NORMAL		TRIANGULAR			UNIFORM		
				MEAN	CUM. PROB.	VALUE	MEAN	STANDARD DEVIATION	MIN.	MAX.	PEAK	MIN.	MAX.	
DSP TSK	1	1	A	.001 SEC										
	2	3	C	.01 SEC										
	3	2	B, D										.02 SEC	.02 SEC
	4	4	E, F	1 SEC										
				5 SEC										

TABLE VI. SYSTEM TABLE

Message	Rate	Length of Test
		3600 Sec.
Time (in seconds)	Rate of Messages	
0	750	<p>MAXIMUM TIME INTERVAL BETWEEN COMPLETION OF A CONTROL CYCLE AND REQUEST FOR A NEW CONTROL CYCLE 1800 Sec.</p> <p>LENGTH OF TIME ALLOTTED THE INTERLEAVE SUBSYSTEM TO STORE DATA 0 Sec.</p> <p>NUMBER OF TEST RUNS 1.</p>

is always uniformly distributed over the unit interval.

The ranges for the messages in this system appear in Table VII.

This means that, if after a message has arrived and the random number is picked to determine what type of message is in the interval .47 to .77, then the message will be tagged as message type D.

Using the procedures just outlined, the inter-arrival time, the time of arrival and type message that arrived of the first 6 messages in the system. (See Table VIIIa.)

In addition to these inputs, suppose that the messages arrive at the times indicated. (See Table VIIIb.)

The DPSS does not actually generate messages this far in advance; rather, it always generates enough messages to keep the arrival of

TABLE VII. RANGE OF NUMBERS FOR EACH MESSAGE TYPE

Message Type	Range (Approximate)	From-To
A	.10	0 to .10
B	.22	.11 to .32
C	.14	.33 to .46
D	.31	.47 to .77
E	.05	.78 to .82
F	.18	.83 to 1.0

the last generated message ahead of the time in the processor. The simulator could have been designed to generate all of the messages that will be used in the simulation prior to making the actual run. This is an acceptable method provided that feedback from the central processing unit will not influence the arrival of messages.

4.2.1 DPC Cycle Simulation—Cycle I (Figures 4 and 5)

The first message (message D) arrives at the computer at 2.16 seconds after the start of the test and is collected in batch 2. It is the first message in the batch. The time criterion of batch 2 is 5 seconds, hence batch 2 will request a data processing cycle at 7.16 seconds. This batch has a “wait-interrupt” option. Message E arrives at 2.27 (2.16 + .11) seconds, and is collected in batch 3 which will request a cycle at 10 + 2.27 seconds or 12.27 seconds. Similarly, Message A arrives at 2.29 seconds and is collected in batch 1, which has the “im-

TABLE VIIIa. MESSAGE ARRIVAL TIMES AND TYPES

Random Number	Inter-arrival Time	Arrival Time from Start of Test	Random Number	Message Type
.362	2.16	2.16	.72	D
.023	.11	2.27	.82	E
.004	.02	2.29	.02	A
.770	7.06	9.35	.35	C
.478	3.12	12.47	.11	B

TABLE VIIIb. MESSAGE ARRIVAL TIMES AND TYPES

Message	Arrival Time from Start of Test
C	18.4 Sec.
E	19.6 Sec.
F	20.7 Sec.
B	21.3 Sec.
A	21.8 Sec.
D	22.8 Sec.
A	23.7 Sec.
B	25.4 Sec.
C	27.5 Sec.
E	36.5 Sec.

mediate-interrupt” option set. The interrupt occurs immediately and a cycle is initiated. Note that the computer was available for other tasks (other than Executive processing) during the first 2.29 seconds.

As soon as a cycle starts, all of the batched messages are transferred to the task processing area and the time and size criteria associated with each batch is reset. There are three messages to process in this cycle:

Message	Time of Arrival
D	2.16
E	2.27
A	2.29

The cycle is set so that the tasks 1, 2, 3, 4 operate in the order 1, 3, 2, 4. Task 1 will

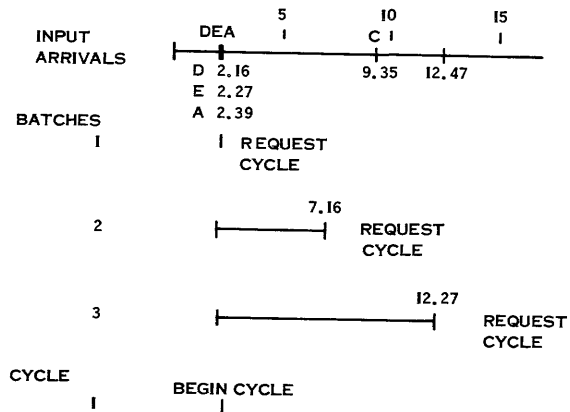


Figure 4. DPC Cycle Simulation Message Batching—Cycle I.

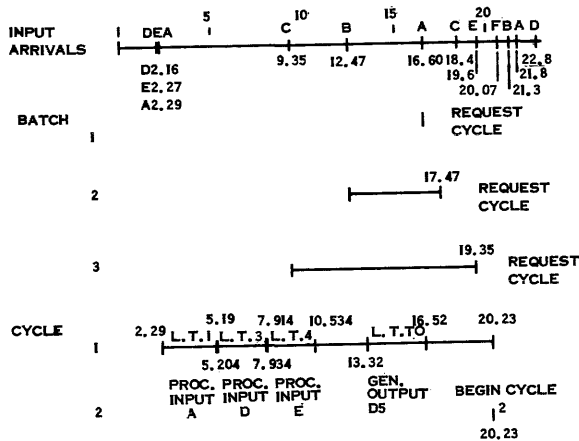


Figure 5. DPC Cycle Simulation.

Cycle I—Operation

Cycle II—Batching

operate during this cycle, since task 1 processes message type A and at least one message A arrived before the cycle began. The processing time for the task is divided into a load and operate time.

The load time for task 1 is uniformly distributed between 1 and 6 seconds. The length of time that is required to load task 1, this time, is determined by picking a uniformly distributed random number and applying it to a formula.

To obtain a uniformly distributed random number lying between a minimum “m” and a maximum “M,” one picks a uniformly distributed random number on the interval (0, 1) and uses the formula

$$y = m + x (M - m) \tag{2}$$

where y is the desired value uniformly distributed between m and M and x is a uniformly distributed random number on the interval (0, 1).

In this case, m = 1, M = 6 and if the random number x is .38 then

$$y = 1 + .38 (6 - 1) = 2.9 \text{ seconds}$$

Hence, it takes 2.9 seconds to load task 1, this time.

Since the cycle started at 2.29 and it takes 2.9 seconds to load task 1, the time is advanced to 5.19 seconds. After the task is loaded, the

operate part of the task must be performed. The operate time for task 1 is exponentially distributed with mean .001 seconds. A uniformly distributed random number (0, 1) is again chosen. The processing time for message A is determined in the same fashion as the interarrival time was chosen for the incoming messages. Suppose that the processing time is determined to be .014 seconds, the simulated time is now advanced to 5.19 + .014 or 5.204 seconds.

The DPSS performs the “load” part of the task operation once per task and the “operate” part once for each message that is processed by the task.

The DPSS is an event oriented simulator hence the simulated time “steps” from event to event. However, whenever the time is advanced, the simulator always checks to see if any other event occurred or was to occur in the interval between events. Whenever this happens, the simulator “backs-up” to take appropriate action. Message processing for task 1 is now complete. Since message A has two displays associated with it, a check must be made to determine if these displays should be forced this time.

The probabilities of forcing the two displays D1 and D2 (Table II) associated with message A are .8 and .3 respectively. Two random numbers are picked, say .94 and .47. Since .94 is not between 0 and .8 and .47 is not between 0 and .3 neither of the displays is forced. If, for example, the random numbers chosen were .63 and .47 then only D1 would be forced.

The next task in the sequence, task 3, operates since it processes message B and D and one message D arrived before the beginning of the cycle. Task 3 is “loaded” into the computer with the load time determined in the way as was used to determine that for task 1. Suppose the load time turns out to be 2.71 seconds; the system time is advanced to 7.914 seconds.

The “operate” part of task 3 is performed on message D. Task 3 has a uniformly distributed “operate” time with minimum equal to the maximum hence the “operate” time, .02, is constant. The time is advanced from 7.914 to



7.934 seconds. Task 3 is now ready to generate displays but since message D does not generate any displays, the cycle moves to the next task in the sequence.

Task 2 which processes message type C is the next task scheduled to be processed, however, no C messages arrived before the cycle began, hence task 2 is skipped during this cycle. Task 4 processes messages E and F and since an E message arrived before the beginning of the cycle task 4 will operate. Suppose its load time is 2.6, advancing the time from 7.934 to 10.534 seconds. During the load time for task 4, message C arrived at 9.35. Batch 3 has no interrupt option, however, message C will cause a cycle initiation request to be issued at 19.35 seconds (10 seconds from the arrival of message C). Since no interrupt occurred, the cycle continues, the operate time for message C taking 2.79 seconds. This brings the time from 10.534 to 13.324. During this time message B arrives at 12.47. Message B is collected in batch 2 which has a "wait-interrupt" feature causing a cycle to be requested 5 seconds after its arrival (being the first message in batch 2) at 17.47, or whenever 2 messages arrive in that batch. The task in progress is continued and since message E forces display D5 with probability 1.0 the task that forces the display must be "loaded" taking, say, 3.2 seconds, bringing the time from 13.324 to 16.524. Display D5 is now forced, which takes 3.71 seconds, bringing the time to 20.234 seconds. Message A arrives at 16.60 seconds during the "operate" part of the task which forces displays. Since message A is collected in batch 1, a new cycle is requested at 16.60. The simulator has the capability of handling a request for interruption of a logical operation in two ways. In the first way, the DPSS can honor the interrupt as soon as it occurs; in this case the "operate" part of the task which forces display D5 would have been interrupted. The second way is to recognize the interrupt request and honor it as soon as the current logical operation has been completed. For purposes of this example, the interruption will take place as soon as the current logical operation is complete. Thus the interrupt request will be held until the output processing is complete, 20.234 seconds, hence a new cycle begins at 20.234 seconds.

4.2.2 DPC Simulator—Cycle II (See Figures 5 and 6)

The following messages arrived during the first cycle and are waiting to be processed in the second cycle:

Message	Time of Arrival
C	9.35
B	12.47
A	16.60
C	18.4
E	19.6

The DPSS has the option of processing the messages in the order that they are received or according to some sequence which is independent of arrival order. In this system the messages are processed according to a preset sequence, i.e., message A is processed before message B even though it arrived later in time. The two C messages will be processed together with the one arriving at 9.35 seconds being processed before the one arriving at 18.4 seconds.

Again, in this cycle, task 1 is the first one to be operated. Suppose the load time is determined to be 3.9 seconds; this advances the

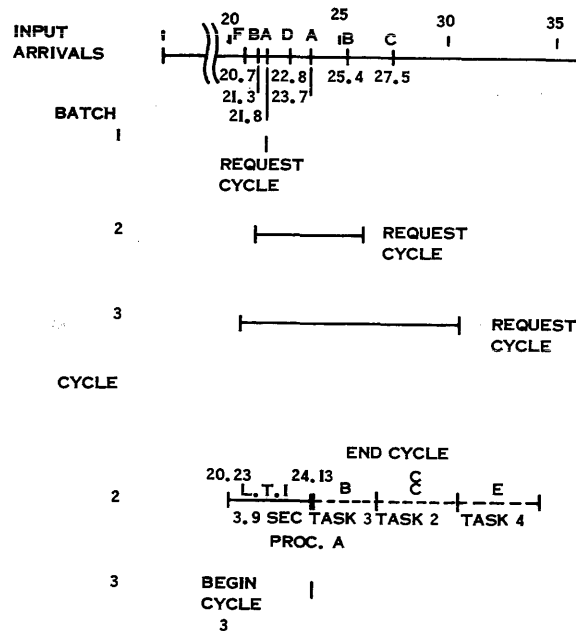


Figure 6. DPC Cycle Simulation.  
 Cycle II—Operation  
 Cycle III—Batching

time to 24.13 seconds. The following events occur during the load of task 1:

Message	Arrives at	Collected in Batch	Request Cycle at
F	20.7	3	30.7
B	21.3	2	26.3
A	21.8	1	21.8
D	22.8	2	26.3
A	23.7	1	

Message A collected in batch 1 causes a cycle to be requested at 21.8 seconds. This request is the "interrupt-immediate" type. However, under the operating rules of this system, the cycle interruption will not take place until task 1 processing has been completed at 24.132 seconds. Note that the second message A arriving at 23.7 seconds does not generate another cycle request. The messages that were to be processed by tasks 2, 3, and 4 are queued when the cycle interrupt occurs.

4.2.3 DPC Simulation—Cycle III (See Figures 6 and 7)

The third cycle starts at 24.132 seconds. The batches are again cleared and the time and size criteria counters are reset to zero. Task 1 operates first (there are two A messages to be processed) and is completed at 27.9 seconds.

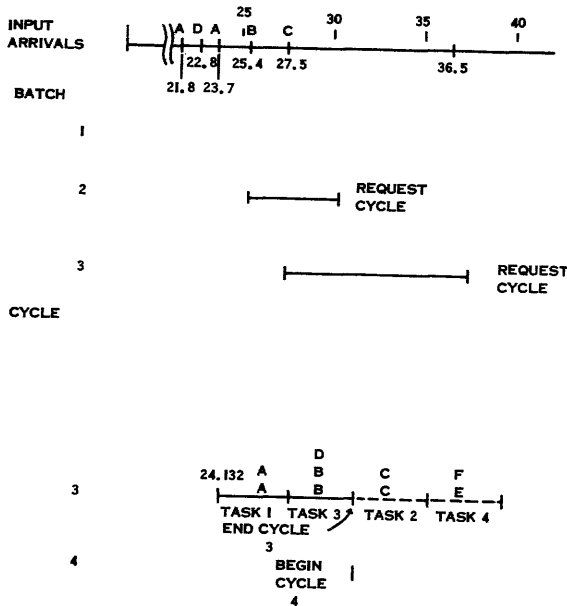


Figure 7. DPC Cycle Simulation. Cycle III—Operation. Cycle IV—Batching

During the operation of task 1, message B arrives at 25.4 seconds and message C arrives at 27.5 seconds. Message B will cause a cycle to be requested at 30.4 seconds and message C will cause a cycle to be requested at 37.5 seconds. The request from batch 2 (containing message B) will be an "interrupt-wait," that is, the interrupt will occur when all of the messages which are processed before B and D have been processed, but the interrupt will not interrupt the currently operating task. In this cycle, task 3 operates after task 1 because two B's and 1 D are waiting to be processed. Processing of task 3 is complete at 31.5 seconds. Now since B is a higher priority input than those processed by tasks 2 and 4 the cycle will be interrupted after task 3 has been completed. The inputs that would normally be processed by tasks 2 and 4 are queued for the next cycle.

4.2.4 DPC Simulation—Cycle IV (See Figures 7, 8, and 9)

Cycle number 4 begins at 31.5 seconds. The following messages have been collected in the batches during the third cycle or have been queued from previous cycles:

Message	Number
B	1
C	3
E	1
F	1

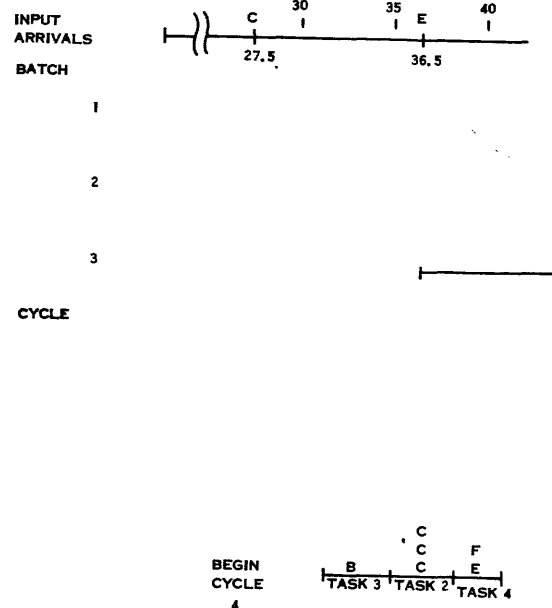


Figure 8. DPC Cycle Simulation. Cycle IV—Operation

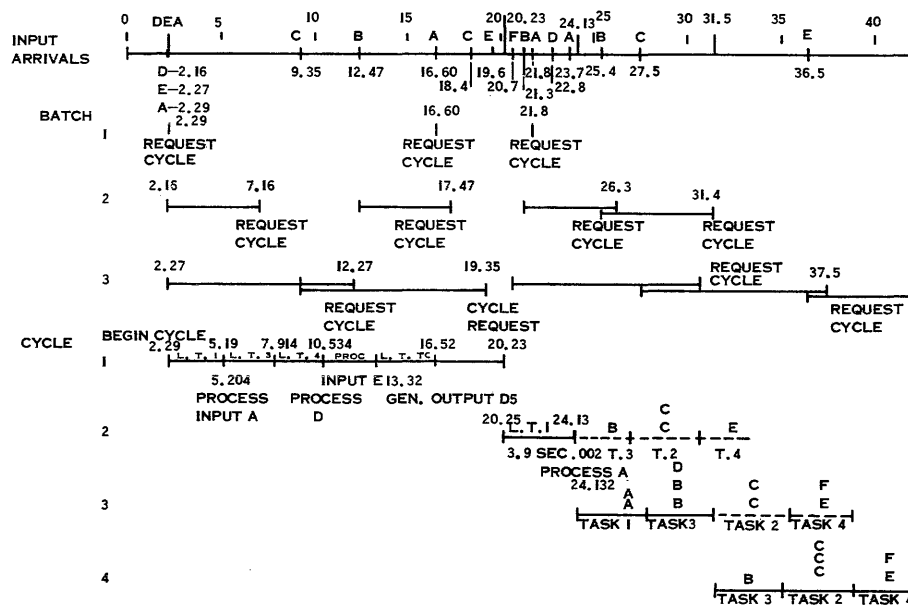


Figure 9. DPC Cycle Simulation Summary. Cycles I, II, III, IV.

Therefore, tasks 3, 2, and 4 will operate in cycle number IV if no interruptions take place. This procedure is continued until the end of the test.

5.0 DPSS APPLICATIONS

The DPSS has been applied primarily to real time management information and command/control type systems. The initial major emphasis was in the design and development of the 465L SACCS. Approximately 300 production runs were made simulating 2000 hours of actual operation.

While the checkout and installation of the entire operational SACCS program system is not yet complete, significant portions of it have been successfully demonstrated. The results of these demonstrations are classified, but it can be said that the DPSS predicted results were close to actual performance figures.

The initial application of the DPSS to the New York State Identification and Intelligence System during the feasibility study phase showed that a class of computers could not handle the job unless the user drastically changed his requirements. The choice was offered to the user early in the system acquisition process to make the trade-off of dollars versus

capability on a more informal basis. As the work on this system progresses, the DPSS will continue to be used to evaluate the various possible system configurations and aid in the selection of the appropriate hardware.

5.1 Application of the DPSS on 465L SACCS

At the outset of the investigations performed with the DPSS, many combinations of SACCS system characteristics were checked because of the complexity of the problem. A list of the major system characteristics checked are shown in Table IX.

One of the system characteristics initially subjected to detailed investigations was the length of the control cycle. This was done because system response time (the time from the initiation of a request for data until the data was presented) was found to be a function of normal uninterrupted DPC cycle time.\*

Cycle time was, in turn, found to be a function of many items, such as total message rate, tasks, sequencing, and batches. It was also

\* This evaluation was made prior to the introduction of the interrupt feature, which permits short cycles and fast response, but which causes the average age of data presented and message queue lengths to increase.

TABLE IX. SYSTEM CHARACTERISTICS INVESTIGATED IN INITIAL SIMULATION RUNS

1. Length of Control Cycle
2. Response Time
3. Maximum Message Capacity
4. Age of Data
5. Storage Requirements
6. Average Processing Time Per Message Type
7. Average Waiting Time Per Message Type
8. Queue Lengths Per Message Type
9. Message Priority System
10. Relationship and Sensitivity of the System to Combination of 1 Through 9 Above

found to be extremely difficult to predict system response time with a reasonable degree of certainty under a wide variety of operating conditions.

To overcome this problem, an "interrupt feature" was introduced into the design, which permits extremely rapid data presentation on demand. However, prior to incorporating this system change the effect of the interrupt feature on the system was checked and the "cost" of extremely short response time was dramatically demonstrated (Section 5.1.1 and Figure 11). The costs and other side considerations are discussed in the next section, Results of DPSS Runs and Their Interpretation.

#### 5.1.1 Results of DPSS Runs and Their Interpretation

The types of outputs from the simulator runs were as shown in Section 3.5 with some typical results shown in Figures 10 and 11.

In testing various possible program system structures, one of the objectives was to attempt to have the average times to process each message type become relatively constant (Figure 10). The average age is checked for a wide range of message loads and the value at which this leveling out occurs would then represent the expected average age of data in the system by message type.

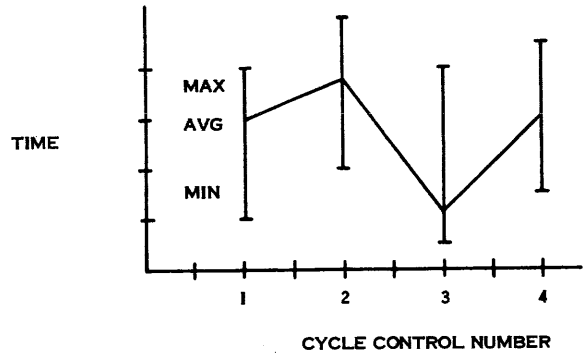


Figure 10. Min, Max and Average Time to Process Each Type of Message by Cycle.

Message-batch-task arrangements are investigated to find those arrangements which tend to stabilize the average processing time values (with minimum spread) in order to select an optimum program structure. The histogram of message processing time for each message type (Figure 11) for the message-batch-task arrangement is then evaluated.

In the case shown in Figure 11, the effect of the interrupt feature (for display requesting) on system performance was evaluated. The high priority display request technique was introduced to permit the interruption of the system in order to respond to data presentation requests in minimum time.

It can be seen from Figure 11 that for a given set of system loading conditions, 100% of the high priority display requests were honored in one minute or less. However, the effect on a low priority data message was dramatic and

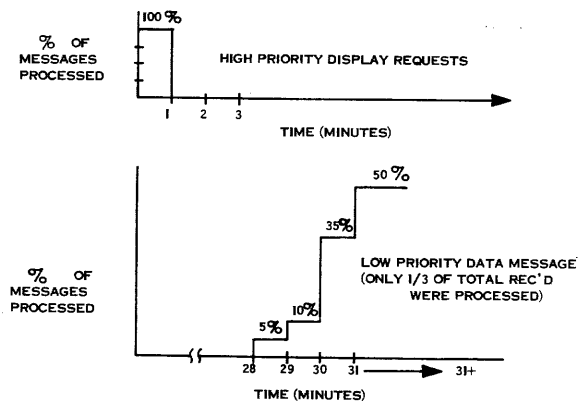


Figure 11. Histogram of Message Processing Time.

required further assessment. None of the lower priority data messages was processed in under 28 minutes. Only 5% were processed between 28 and 29 minutes, 10% processed between 29 and 30 minutes, 35% processed between 30 and 31 minutes, and 50% of all messages processed took 31 minutes or longer.

These results can be interpreted to mean that the high priority data presentation requests which were honored in one minute or less did not make use of the data contained in the lower priority data messages. For the same case, the queue length was determined from the fact that only  $\frac{1}{3}$  of the lower priority data messages that were received were processed.

It was further necessary to determine exactly what type of data was contained in each type of message and what its variability might be. If the lower priority message contained data which varied relatively little over a period of several hours, then the fact that 50% of the data might be as much as 30 minutes old is of considerably less consequence than if the data were extremely variable in less than 30-minute increments.

Therefore, by investigating the histograms for each message type, establishing limits on the number of high priority display requests which might be made periodically, and determining the proper message priority and message-batch-task relationship, it was possible to arrive at a program system structure which satisfied Operational Requirements.

### 5.2 General Results

The results of the simulations showed that the system is most sensitive to the use of the interrupt feature. This "system interrupt" was found to cause short cycle times which could create the impression of highly efficient operation. However, the queue sizes and waiting times were increasing as a result of the increased use of the interrupt features.

The total message rate imposed upon the system with all or most message types being present was found to be second in importance to the system's operational performance. The presence of all (or most) message types causes a maximum number of input-output (I/O) transfers to occur. I/O transfers are one of the most

time consuming aspects of data processing tasks and caused total cycle time to increase as the number of transfers increased.†

It was also determined that the system was not particularly sensitive to the relative frequency of each message type for any given input rate as long as all or most types of messages were present. This particular aspect of the environment assumed secondary importance based on these results.

The investigation of the "time to load" and "time to process" portions of the system's task operating times showed a high proportion of time to load vs. time to process. This suggested that the system's apparent I/O limitations would bear further investigation for possible improvement of overall system operation.

### 5.3 System Improvements

A disc file is used as the principal auxiliary storage device for the SACCS. A study of its characteristics suggested that a change from the current serial read procedures to parallel read procedures could produce a considerable reduction of I/O time. A conservative estimate of the I/O time reduction factor attainable was set at 10 because of anticipated engineering problems, and also because there are existing hardware items other than disc files which have the desired timing characteristics and do not involve modification of the disc.

When only the times to load and unload (I/O) of all system times were reduced by a factor of 10 the results of the DPSS runs showed a very high payoff available for such a modification.

The probable results from the reduction of the I/O time are shown in Table X. In these results, the "rates" refer to the message input rate per hour; the "load" refers to either a "normal" (N) loading and unloading time or a "1/10 normal" loading and unloading (I/O) time. "Interleave" means the availability of DPC time for other operations which might normally occur as part of a time sharing func-

† This is not necessarily a linear function. Also note that the effect of I/O transfer time on total cycle and response times is a complex relationship, i.e., many I/O's can occur with no responses required and response time therefore becomes meaningless in this case.

tion of the system. The load rates shown are hypothetical and do not in any way reflect actual rates. However, the effect of reducing the I/O time factor in the total system operating time is clearly demonstrated as a function of relative loads.

It is apparent from the results shown in Table X that auxiliary storage devices with the performance specifications inferred from the reduced I/O time factor would be desirable to permit maximum time sharing and maximum expansion potential in the system.

#### 6.0 DPSS CAPABILITIES SUMMARY

A summary of the capabilities of the Data Processing System Simulator are:

1. System Feasibility Studies
2. Simulates Computer Based Data Processing Systems
3. Evaluate Equipment and Processing Discipline Combinations vs. System Operational Requirements
4. Establish Equipment Configuration for the System
5. Establish Program Configuration for the System
6. Development of Detailed Design Requirements (Operational Program Requirements, Subsystem Design Specifications, Program Design Specifications)
7. Set Initial Parameters for the Operational System
8. Determine System Performance Requirements (for Acceptance Testing)

TABLE X. PROBABLE RESULTS FROM MODIFICATION OF I/O DEVICE

Conservative Assumed Improvement of I/O Time = Factor of 10			
Rate	Load	Interleave	Comment
10	N	0-30%	Acceptable
30	N	None	Poor
10	1/10	80%	All Requirements Met
30	1/10	60%	All Requirements Met
60	1/10	30%	All Requirements Met

#### 9. Evaluate Proposed System Modification and Retrofits before Implementation

The flexibility of the DPSS in terms of the variability of both the inputs to be tested and simulation program logic makes this tool useful in the early stages of establishing data processing system requirements. It is a powerful tool in performing system feasibility studies, simulating the operation and performance of computer based data processing systems, and in evaluating equipment and data processing discipline combinations as a function of system operational requirements.

Once past the initial phases of system development process, the DPSS can continue to be useful in helping to evaluate the equipment configurations being considered for the system, and in establishing the framework for the computer program configuration for the system. This latter framework includes items such as the need for Executive or master control programs, the structuring and organization of system inputs, processing tasks, and outputs.

During the system implementation and acquisition phase, the DPSS is of continuing usefulness in the development of detailed design requirements. The key design features for Operational Program Requirements (OPR), Subsystem Design Specifications (SSDS), and Program Design Specifications (PDS) can be determined and established as design goals. This work, in addition, is valuable in setting the initial parameters for the operational system.

The DPSS can be used to develop System Performance Requirements (SPR) which can be used at the conclusion of the system acquisition phase during which acceptance testing is performed. The SPR's can be established early in the design process and used by both the contractors and the procuring agencies as performance criteria for determining the successful completion of the design and implementation phase.

Proposed system modifications and retrofits can be evaluated before commitments are made for additional equipment, computer programming, or human action requirements. This evaluation is essentially the performance of the system feasibility studies discussed earlier in this section. Thus, the design, development, in-

stallation, and acceptance testing procedure can be completed by providing the analytic capability needed to continually refine and improve any system in existence or proposed for future development without becoming so deeply committed in time and dollars that prohibitive rework costs are incurred as is currently the case in the field of command/control and management information systems development.

## 7.0 EXPANDED DPSS CAPABILITIES— MODEL C

The experience gained by applying the DPSS to three major management control systems in different phases of development (see Section 1.0) identified areas in the original version of the DPSS which could be further generalized and expanded so that it can be used to simulate a greater variety of data processing systems. The expanded DPSS—Model C is now operational on the IBM 7090/7094 computer. DPSS—Model C is a self contained program package; i.e., it does not require the use of a control program. In this way, the operation of the program is “streamlined” so as to reduce the computer time required to make each run.

In addition to the capabilities of the original DPSS described in Sections 1.0 through 6.0 inclusive, DPSS—Model C has the following additional features:

- a. The DPSS—Model C program is structured so that one programming task has the ability to create inputs for other tasks.
- b. Associated with each message-task relationship is the probability of message being processed by each of its associated tasks. A probability of 1.0 ensures that a message will always be processed by a particular task.
- c. In order to accommodate transient states in the input message frequencies, DPSS—Model C permits changes in the relative frequency of input messages during the simulation run. This means, for instance, that the DPSS can simulate changes in the mode of operation of the system.
- d. DPSS—Model C has the capability of simulating several levels of system operation, where one level has priority over another. Each level can be considered as

a system in itself—messages are collected in batches, processed in tasks according to some processing discipline (e.g., first come, first serve or according to a task sequence), generate displays, etc. Every level may, but need not, have identical characteristics except that one has priority over the other one. When going from one level to another the simulator has the capability of aborting the lower priority level immediately or at the end of a logical operation (e.g., completion of a task) going to the higher level, doing the required processing and then returning to the point of interruption. Up to 100 levels can be requesting service at the same time, with the highest priority service first, the next highest priority second, and so on.

- e. The simulator has the capability of selectively emptying the individual batches for each data processing cycle. This will operate so that only the messages collected in the batches whose batch criteria are exceeded before the beginning of the cycle will be processed in that cycle. The simulator has the option of transferring to the processing area for processing in the next cycle only those messages which were collected in batches whose batch criteria was exceeded before the beginning of the cycle. The rest of the messages remain in their respective batches until their batch criteria have been exceeded. This capability enables the DPSS to simulate more than one independent system with a priority arrangement such as might be found in a time sharing system.<sup>4</sup>
- f. It is possible to delete messages in the system as a result of other messages being processed. This capability covers conditions when “partial updating” of files is specified in early messages in the system and where a “complete update” message is received which includes the conditions cited in the “partials.” In this way, the simulator will not duplicate the processing of any message.
- g. The capability is provided to regulate task operation so that a task will operate

- only once in every “n” cycles; “n” must be specified in the input data.
- h. The DPSS has the capability of resetting the computation counters (e.g., those counters used to compute average message processing time, maximum processing time, etc.) at a prespecified time after the beginning of the test. With this feature, it is possible to study systems in transition, during steady state conditions, or a combination of transient and steady state.
  - i. The DPSS has the capability of providing three sets of periodic summaries throughout the test. The summaries can be presented for every message-task relationship in the system using three time periods. For example, the average processing time for a given message being processed in a given task may be presented every 30 seconds, every 25 minutes and every hour.
  - j. The concept of generating or forcing displays as a result of messages being processed has been generalized as follows:
    - (1) There can be any number (limited only by the capabilities of the simulator) of tasks that produce forced displays. Each display producing task has its own processing characteristics.
    - (2) The tasks that produce forced displays need not follow the task which processed the data messages which caused the display to be forced, indeed, the tasks which force displays can be located anywhere in the task sequence.
  - k. Additional capabilities have been added to the DPSS to permit detailed investigation of the effects of limited buffer size on the functioning of a data processing system. The DPSS also has the option of “losing-from-the-system” those messages which attempt to enter a full buffer. The DPSS identifies those messages which have been “lost,” or it can exercise the option of queuing these messages and not losing them.
  - l. The manner in which outputs from the simulator can be presented has been made more flexible so that only the necessary or desired information will be produced.
  - m. The processing of messages by tasks has been modified to permit more flexible and detailed simulation of the operation of a task. Task processing is divided into three sub-operations: (1) performing the input operation, (2) processing each message associated with the task and (3) performing the output operation. Each of the sub-operations has its own distribution function and parameters. In addition, the DPSS has the option of performing a “save data” operation in the event of an interruption during the operation of a task. When this option is exercised, the “save data” part of the task operation will be executed before the request for interruption is honored. The “save data” operation has its own distribution function and parameters. There may be a “save data” operation associated with each task in the system.

## 8.0 FUTURE DEVELOPMENTS

Future developments of data processing simulators will most likely be along the lines of multi- and parallel-processing and will include prediction techniques so that the time required to develop a data processing system under various configurations can be studied.

## APPENDIX I

### SAMPLE PRINTOUT

This appendix contains a sample printout from a simulation run. The output data contains the following items of information:

1. The cycle number, its begin and end time.
2. For each message arrival the message identity, time of arrival, the batch in which the message is batched, the task that processes the message (mod category), the priority of the message, the time when the processing of the message was completed, the displays that are forced by this message and the time that each display is forced.



- 3. A list of messages that have not been completely processed at the end of each cycle.
- 4. Averages by batch, priority, message of
  - a. Processing time
  - b. Total time
  - c. Waiting time
- 5. Number of messages queued at beginning and end of cycle.
- 6. A list of messages remaining to be processed at the end of the test.
- 7. A total time distribution by minute, per message at end of run. It tells, e.g., what percent of messages were processed between, say, 10 minutes and 11 minutes.

DPC SIMULATION—RUN NUMBER 1  
 CYCLE REQUEST AT 7.65 REASON SIZE  
 BEGIN CONTROL CYCLE NUMBER 1 AT 22.65

BEGIN CYCLE NUMBER 5 AT 638.48  
 MESSAGE QUEUE LENGTH—565  
 BATCH SIZES

BATCH NR.	SIZE
0	0
1	0
2	1
3	373
4	60

TYPE	MOD	ARRIVAL	OUT	BATCH	PRIORITY	NR.	TIME	TOTAL	INDENT	TIME	WAIT
5	2	643.19	664.29	3	0			21.11		5.40	
13C	0	382.50	693.63	3	1			311.13		311.04	
21	2	568.09				1	809.52	A02			
21	2	568.09				2	809.57	A02P			

CYCLE INTERRUPT AT 813.67 REASON WAIT  
 END CYCLE NUMBER 5 AT 834.86

.....  
 690 MESSAGES REMAINING

TYPE	MOD	ARRIVAL	BATCH	PRIORITY
24	4	1.52	3	1
40	4	4.41	3	1

AVERAGES BY BATCH

BATCH NR	WAIT	TOTAL TIME
0	NO MESSAGES	
1	NO MESSAGES	
2	56.50	56.56
3	227.53	227.65
4	253.24	253.30

AVERAGES BY PRIORITY

PRIORITY NR	WAIT	TOTAL TIME
0	8.57	17.82
1	233.19	233.25

AVERAGES BY MESSAGE TYPE

TYPE	MOD	NR PROCESSED	WAIT	TOTAL TIME	MAX TOTAL TIME	MIN TOTAL TIME
0		NO MESSAGES				
1	2	2	275.82	275.88	346.82	346.82
2	2	1	233.24	233.29		
3	2	18	245.29	245.35	390.05	169.80

TOTAL TIME DISTRIBUTION

RAR

FROM 0 MIN. TO 1 MIN.	.33	FROM 1 MIN. TO 2 MIN.	.33
FROM 2 MIN. TO 3 MIN.	.33	FROM 3 MIN. TO 4 MIN.	
FROM 4 MIN. TO 5 MIN.		FROM 5 MIN. TO 6 MIN.	
FROM 6 MIN. TO 7 MIN.		FROM 7 MIN. TO 8 MIN.	
FROM 8 MIN. TO 9 MIN.		FROM 9 MIN. TO 10 MIN.	
FROM 10 MIN. TO 11 MIN.		FROM 11 MIN. TO 12 MIN.	
FROM 12 MIN. TO 13 MIN.		FROM 13 MIN. TO 14 MIN.	
FROM 14 MIN. TO 15 MIN.		FROM 15 MIN. TO 16 MIN.	
FROM 16 MIN. TO 17 MIN.		FROM 17 MIN. TO 18 MIN.	
FROM 18 MIN. TO 19 MIN.		FROM 19 MIN. TO 20 MIN.	
FROM 20 MIN. TO 21 MIN.		FROM 21 MIN. TO 22 MIN.	
FROM 22 MIN. TO 23 MIN.		FROM 23 MIN. TO 24 MIN.	
FROM 24 MIN. TO 25 MIN.		FROM 25 MIN. TO 26 MIN.	
FROM 26 MIN. TO 27 MIN.		FROM 27 MIN. TO 28 MIN.	
FROM 28 MIN. TO 29 MIN.		FROM 29 MIN. TO 30 MIN.	
FROM 30 MIN. TO 31 MIN.		FROM 31 MIN. TO	

APPENDIX II

THE USE OF RANDOM NUMBER GENERATOR

1. *The Random Number Generator*<sup>5</sup>

$X_{n+1} = 129 X_n + 227216619 \pmod{268435456}$  is used to generate numbers between 0 and 1 which are statistically tested to be uniformly distributed. That is

$$\frac{X_{n+1}}{268435456}$$

are numbers lying between 0 and 1 and satisfy various statistical tests for being uniformly distributed (0, 1). The first number of the sequence is gotten by letting  $X_0 = 0$  and then  $X_1 = 227216619$ .

2. *Uniformly Distributed Random Numbers*

Suppose it required to obtain a uniformly distributed random number lying between m and M. If one obtains a number x uniformly distributed (0, 1) then the converted number

$$y = m + x (M - m) \tag{3}$$

is uniformly distributed (m,M).

In practice this is used if the only information available is the minimum and maximum of a random variable.

3. *Normally Distributed Random Numbers*

A table is assumed stored in core of normal distribution with mean 0 and standard deviation 1. Suppose a random number y given with normal distribution of mean  $\mu$  and standard deviation  $\sigma$ . Then

$$\frac{y - \mu}{\sigma}$$

is a normally distributed random variable with mean 0 and standard deviation 1.

To choose a random number y, normally distributed with mean 0 and standard deviation 1 a number x, uniformly distributed (0,1) is chosen. Using x find the corresponding Z from the table of the normal distribution (mean 0, standard deviation 1).

Now,

$$Z = \frac{y - \mu}{\sigma} \tag{4}$$

Hence,

$$y = \mu + \sigma Z \tag{5}$$

and y is the desired number.

4. *Exponentially Distributed Random Numbers*

An exponentially distributed random number x has the distribution

$$P [x \leq t] = \begin{cases} 0, & t < 0 \\ 1 - e^{-at}, & t > 0 \end{cases} \tag{6}$$

where  $\frac{1}{a}$  is the mean of the distribution. Since negative values of t have no application for the simulator it will be assumed here that  $t \geq 0$ . To obtain an exponentially distributed random number, a random number y, uniformly distributed (0,1) is given and is set equal to

$$1 - e^{-ax}$$

Hence,

$$y = 1 - e^{-ax} \tag{7}$$

and solving for x,

$$x = \frac{1}{a} \ln \frac{1}{1 - y} \tag{8}$$

5. *The Triangular Density Function*

Sometimes, besides the minimum and maximum of a random variable, some "favorite" value is known. This suggests the use of the triangular density function. Let m, M be the minimum and maximum and K the "favorite" value. The density function assumes the triangular shape shown in Figure 12.

The area of the large triangular, mPN, is equal to 1. To obtain a number with such a density function, a random number X uniformly distributed (0,1) is chosen and a number Y is computed so that the region designated by A has area X.

6. *Arbitrary Continuous Distribution*

The random number generator enables the user of the simulator to utilize almost any dis-

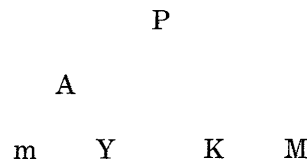


Figure 12. Triangular Distribution

tribution he desires. The basic requirements for such a distribution are:

- a. that it be continuous
- b. that the random variable be bounded.

Condition 1 is usually the case. A distribution function that has a discontinuity does not usually occur in a simulation, and can, if necessary, be approximated by a continuous distribution.

Condition 2 requires that the user specify that the value of the random variable not exceed (so far as the approximation is concerned) some specified value.

Suppose Figure 13 represents some distribution. Various values (in this case  $x_1, x_2, x_3, x_4$ ) are chosen from the distribution to be approximated.

Let  $F(x)$  be the original distribution. Then

$$P_1 = \text{Prob} [x \leq x_1] \quad (9)$$

$$P_2 = \text{Prob} [x \leq x_2]$$

and so on until

$$\text{Prob} [x \leq x_4] = 1 \quad (10)$$

Suppose a random number  $x$ , uniformly distributed (0, 1) is chosen. Then if  $0 \leq x \leq P_1$ ,

$$X = \frac{P_1}{x_1} x \quad (11)$$

where  $X$  is the desired value.

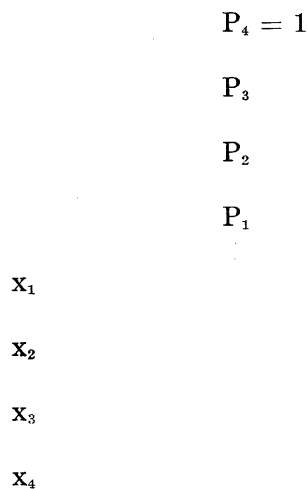


Figure 13. Arbitrary Continuous Distribution

If  $P_1 \leq x \leq P_2$ , then

$$X = x_1 + (X - P_1) \left( \frac{x_1 - x_2}{P_1 - P_2} \right) \quad (12)$$

and so on, testing whether  $R$  lies between two successive  $P$ 's and then using linear interpolation to determine the value between the two points.

### APPENDIX III

#### GLOSSARY OF TERMS

The following definitions are included for the reader who may not be familiar with the terminology of the document.

#### BATCH

A device which is used to request a DPC cycle.

#### BATCH CRITERIA

Parameters associated with each batch which determines when a batch will request a DPC cycle to be initiated.

#### DATA PREPARATION

A task which processes every incoming message to make sure that they are valid. The Data Preparation Task also determines which tasks must operate in the present DPC cycle.

#### DISPLAY

A presentation of information contained in the DPC program system files.

#### DPC CYCLE INTERRUPT

A DPC cycle interrupt occurs when all of the messages of the present control cycle are not processed before a new cycle begins.

#### DPC CYCLE REQUEST

A request for a DPC cycle when the batch criteria of a batch is exceeded.

#### DPC PROGRAM SYSTEM CYCLE

A sequence of tasks that are performed to process the system message.

#### DPC PROGRAM SYSTEM (OR CYCLE)

That part of the system within the Data Processing Central (DPC) which deals with the primary processing functions.

#### FLASH MESSAGE

A message that is processed immediately,

upon receipt by the computer. This message does not cause a new cycle to be initiated.

#### FORCED DISPLAY

A display which is generated as the result of a message (data) being processed.

#### INTERARRIVAL TIME

The time between the arrival of two successive messages.

#### INTERLEAVED SUBSYSTEM

Any subsystem other than the primary real time program system.

#### LOAD TIME

The time required to transfer the operating programs and data environment associated with a task from tape, drum or disc to core.

#### MESSAGE

An input into the DPC program system.

#### MOD

The position of a task in a sequence of tasks (the task sequence number).

#### OPERATE TIME

The time required by the operating programs to process a message once the operating programs and environmental data are in core.

#### PRIORITY

A measure of importance in a message.

#### PROCESSING TIME

The time from when processing in a message is begun to when it is completed.

#### RANDOM NUMBER

A number from a random number generator tested for certain statistical properties.

#### REQUESTED DISPLAY

A display which is generated as the result of a special message being processed (a display request message).

#### TASK

A related collection of programs (considered in the DPSS as a single operation) which operate on a message or a set of messages.

#### TOTAL TIME

The time from the arrival of a message at the DPC to completion of message processing.

#### WAIT TIME

The time which begins when a message arrives in the DPC and ends when processing of this message begins.

### APPENDIX IV

#### MACRO INSTRUCTIONS

The logical operation of the model is governed by the interpretive program. The operation of the simulator can be changed by changing the flow of the interpretive program. The flow diagram that is currently being used in the simulator is shown in Figure 14.

Figure 14 illustrates the interpretive program as it is read into the computer. The first few instructions of the interpretive program are explained below. The last line of the flow reads

STOP IA

This indicates that the flow diagram data is complete and the program is to execute the instruction labeled IA first. This instruction is found on line 1:

IA    QMSGAR    IB    ID

IA is the instruction label. QMSGAR is the instruction. The Q usually designates that the instruction asks a question. In the case QMSGAR asks whether any messages have arrived. If yes go to IB, if no go to ID. Suppose a message has arrived then one goes to IB:

IB    AGNMSG    IA    FIN

AGNMSG begins with an A which usually designates an action, and the instruction states that a message is to be generated and then one goes to IA or FIN depending upon whether there is more to do in the simulation or not.

If no message has arrived, one goes to ID:

ID    QFLASH    IE    IC

QFLASH asks whether any flash messages are to be processed. If there are go to IE, if not to IC.

#### *Macro Programming Instructions*

The instructions QBEGCC, QMSG, QSTART are macro programming instructions. The list

IA	QMSGAR	IB	ID
CA	QBEGCC	CB	CF
CB	ABEGPC	CC	FIN
CC	QTASKS	IA	CE
CE	AENDCC	IA	FIN
CF	QSTART	CG	CO
CG	QMSG	CH	CC
CH	LDTASK	IA	FIN
CK	APCMSG	IA	FIN
CC	QMSG	CK	CP
CP	QBEGFD	CQ	CR
CQ	QFORCE	CQO	CC
CQO	ALDDP	IA	FIN
CR	APCFRC	CS	FIN
CS	QFORCE	IA	CC
IB	AGNMSG	IA	FIN
IC	QFLDSP	IH	IF
ID	QFLASH	IE	IC
IE	APCFLS	IA	FIN
IF	QCC	IG	IM
IG	QINT	III	CA
IH	QFLLD	IA	FIN
II	ABEGCC	CA	FIN
IM	QREQ	IN	IR
IN	AGNMSG	ID	FIN
IR	AABORT	IS	FIN
FIN	FINISH		
IS	ABEGCC	IA	FIN
III	AENDCC	II	FIN
	STOP	IA	

Figure 14. Interpretive Program Flow Diagram

of macro instructions and their meanings are as follows:

1. *DUMP*

This instruction gives an octal dump.

2. *QBEGCC*

This instruction asks whether at the time the instruction is to be operated upon the system is at the beginning of a cycle. The beginning of a cycle consists of the tasks of loading and operating the data preparation tasks.

3. *ABEGPC*

This instruction performs the loading and operation of the data preparation task only at the beginning of the cycle.

4. *QTASKS*

The instruction inquires whether any more tasks are to be done in the cycle and sets the indicators to the next task done by the cycle.

5. *AENDCC*

This instruction resets various indicators so as to signal the end of a cycle. It also gives a summary of the output data for that cycle.

6. *QSTART*

This instruction inquires whether the system is at that moment at the start of a task, i.e., whether the program for that task was loaded.

7. *QMSG*

This instruction inquires whether there are messages in the table for the particular task to operate upon.

8. *LDTASK*

This instruction "loads" the task in question, i.e., determines how long it takes to load the task and increment the simulated clock.

9. *APCMSG*

This instruction processes the appropriate incoming message according to processing priority.

10. *QBEGFD*

The instruction inquires whether the forced display task should be "loaded."

11. *ALDDP*

This instruction "loads" the display task.

12. *QFORCE*

This instruction inquires whether there are forced displays to process.

13. *APCFRC*

This instruction processes the appropriate forced display.

14. *QMSGAR*

This instruction inquires whether any messages have arrived.

15. *AGNMSG*

This instruction generates messages if any are due to arrive.

- 16. *QFLDSP*  
This instruction inquires whether any of the flash messages generate display.
- 17. *QFLASH*  
This instruction inquires whether there are any flash messages to process.
- 18. *APCFLS*  
This instruction processes flash messages.
- 19. *QCC*  
This instruction inquires whether the cycle is operating.
- 20. *QINT*  
This instruction inquires whether the cycle is to be interrupted.
- 21. *QFLLD*  
This instruction either "loads" or processes the flash display task, whichever is appropriate.
- 22. *ABEGCC*  
This instruction begins a cycle.
- 23. *QREQ*  
This instruction inquires whether a message will arrive before the request for the instruction of a cycle.

- 24. *AABORT*  
This instruction aborts the interleaved subsystem.
- 25. *FINISH*  
This instruction finishes the run, summarizes the data and goes on to the next run, if any.

APPENDIX V

SUMMARY DESCRIPTION OF THE IBM AN/FSQ-32 COMPUTER<sup>+</sup>

The AN/FSQ-32 computer is a 1's-complement, 48-bit word computer, with 65,536 words of high-speed (2.5 m sec. cycle time minus overlap) memory available for programs, and an additional 16,384 words of high speed memory available for data and input/output buffering; the latter memory is called input memory.

There are four core memory banks (of 16K words each) which are individually and independently accessible by three control units: the central processor unit, the high speed control unit, and the low speed control unit. High speed I/O, low speed I/O, and central processing can take place simultaneously out of different memory banks, or with certain restrictions, out of the same memory bank.

*Characteristics of the AN/FSQ-32 Storage Devices*

Device	Size	Word Rate	Average Access Time
Core Memory	65K	2.5 $\mu$ sec/wd	
Input/Output Core Memory	16K	2.5 $\mu$ sec/wd	
Magnetic Drum	400K	2.75 $\mu$ sec/wd	10 msec
Disk File	4000K	11.75 $\mu$ sec/wd	225 msec
Magnetic Tapes	16 Drives	128 $\mu$ sec/wd (high density)	5 to 30 msec (no functioning) *

\* Depending on whether the tape is at load point, and whether it is being read or written.

## REFERENCES

1. H. M. MARKOWITZ, B. HAUSNER, and H. W. KARR, *Simsript: A Simulation Programming Language*, RAND Corporation, Santa Monica, California, 1962.
2. *General Purpose Systems Simulator II—Reference Manual*, IBM, 1963.
3. C. A. KRIBS, *Building a Model Using Simpac*, System Development Corporation, Santa Monica, California, TM 602/300/00, November 15, 1962.
4. J. D. SCHWARTZ, E. G. COFFMAN, and C. WEISSMAN, *A General-Purpose Time-Sharing System*, System Development Corporation, Santa Monica, California, SP-1499, April 29, 1964.
5. P. PEACH, "Bias in Pseudo Random Numbers," *Journal of American Statistical Association*, Vol. 56, No. 295, September, 1961.
6. M. I. YOUCHAH and D. D. RUDIE, *A Universal DPC Simulator Applied to SACCS Program System Design*, System Development Corporation, Santa Monica, California, SP-924, July 8, 1963.
7. D. D. RUDIE and M. I. YOUCHAH, *The Data Processing System Simulator (DPSS)*, System Development Corporation, Santa Monica, California, SP-1299, March 23, 1964.



# THE USE OF A JOB SHOP SIMULATOR IN THE GENERATION OF PRODUCTION SCHEDULES

*Donald R. Trilling*

*Westinghouse Electric Corporation, Pittsburgh, Pennsylvania*

The following describes some techniques under development at the Steam Division of the Westinghouse Electric Corporation. This plant, located at Lester, Pennsylvania, manufactures large Steam Turbines, and its main facility is an exceptionally large job shop. It is the locale where many of the concepts discussed below underwent development. However, it should be made clear that this paper is not in any way intended to be a progress report on their use there. The nature of these techniques remains highly experimental, and they are described here as a matter of interest to those who are concerned with the potential of computers in management applications.

## I. BACKGROUND

The technique of job shop simulation is becoming increasingly well-established in industry, and the advent of the macro simulator languages, such as SIMSCRIPT<sup>7</sup>, assures continued movement in this direction. To date, job shop simulators have been used principally as a tool for facilities planning<sup>1, 2</sup>, and as a testing mechanism for the merit of various decision rules<sup>3, 10</sup>. Typically, a model of the shop is set up in the core of the computer, having a similar configuration of men and machines as exist in the real shop or could exist in a contemplated shop. It is supplied with data on manufacturing orders representing the coming load in the shop. The simulator processes the orders much as the real shop would. By seeing how the model shop fares in processing its load,

we gain some insight into how the real shop will perform in processing an equivalent load. The model shows specifically how machines will be loaded, the extent of the queues that form, and when orders may be expected to be completed compared to the schedules set for them. If management is considering a change in facilities, or a change in procedures, much of the implications of these changes may be learned in advance by trying them out in the model.

The comparison of different simulations is made on the basis of certain shop statistics, reported out periodically as the simulation proceeds. These statistics are well known, and follow the pattern set by the original GE-IBM Job Shop Simulator<sup>4</sup>. Such measures as machine utilization, average waiting time in queue, average queue lengths, and order lateness are given. A sampling of these reports is included in Appendix I. To the original set we have added such measures as shop hours, overtime hours, machine substitution counts and some others.

The knowledge gained by simulation, in experiments such as outlined above, is quite useful for many scheduling decisions. However, in general, we may state that this use extends only to what might be called the guiding of scheduling decisions. It does not directly assist in the preparation of actual schedules. The idea that the technique of simulation may be turned to such a use is shown below.

The job shop simulator devised at Westinghouse Steam Division has been named SHOP-

SHOPSHAPE. SIMULATOR TO HUNT OPTIMAL PRODUCTION SCHEDULES  
 HIGHLIGHTING AGGREGATE AND PARTICULAR EVENTS. WESTINGHOUSE,  
 D.R. TRELLING, PROJ. LDR. - HILBURN, CARROLL, SHANE, WICKENHEY

DATE	JAN 26 1962
SIMULATION IDENTIFICATION NUMBER	827.0
NUMBER OF REPORTING PERIODS	8
NUMBER OF WORK DAYS PER PERIOD	20
SIMULATION BEGIN DATE, DEC HRS	526.0
ENDING DATE FOR 1ST PERIOD	523.0

Shop Statistics—Title Page, showing name, manning, dispatch rule, and re-routing table for each machine group

PERIODIC OUTPUT												
SHOP PERFORMANCE												
PERIOD 8		THIS PERIOD					YEAR TO DATE					
BEGIN DATE IN DECIMAL WEEKS		535.0					526.0					
STANDARD SHOP TIME		827.5 HOURS, OR 19.0 DAYS					1040.0 HOURS, OR 64.0 DAYS					
PLANNED OVERTIME PERIODS		135.0 HOURS, OR 6.0 DAYS					472.5 HOURS, OR 21.0 DAYS					
ALLOWED OVERTIME PERIODS		0 HOURS, OR 0 DAYS					0 HOURS, OR 0 DAYS					
HOLIDAYS AND SUNDAYS		72.0 HOURS, OR 3.0 DAYS					144.0 HOURS, OR 6.0 DAYS					
LABOR CLASS	MACH GROUP	MACHINE DESCRIPTION	AVAIL CAPAC	UTIL CAPAC	IDLE TOTAL	UTIL CAPAC	WORK TO OTHERS	AVAIL CAPAC	UTIL CAPAC	IDLE TOTAL	UTIL CAPAC	WORK TO OTHERS
	100/236		9501	9501	1,000	5765	5765	1,000				
	BPF 111		428	367	61	.858		1440	953	487	.662	
	ELN2 111		143	143	1,000	7		480	445	35	.927	32
	ELN6 111		285	285	1,000			760	852	98	.898	
	FITF 111		1140	559	581	.490	26	3040	2314	1526	.602	133
	HMS 111		143	110	33	.771	27	400	358	142	.705	72
	LOC 111					.763	87	400	386	94	.804	
						.623	2	1440				

Shop Statistics—Shop Performance Report, showing utilization and re-routes for each machine group

PERIOD 8		YEAR TO DATE																				
MACH GROUP	ARRIV H	ARRIV L	ARRIV T	ARRIV H	ARRIV L	ARRIV T	DEPART H	DEPART L	DEPART T	DEPART H	DEPART L	DEPART T	O-TIME H	O-TIME L	O-TIME T	AVER O H	AVER O L	AVER O T	AVER O H	AVER O L	AVER O T	
100/236	2075			2075			2053						0	0	0	0	0	0	0	0	0	0
BPF111	9			9			8						1.51			1.514			.0	.3		.3
ELN2111	36			36			33						1.43			1.428			1.8			1.8
ELN6111	9			9			6						1.20			1.202			0			0

Shop Statistics—Analysis of Queues, showing arrivals, departures, average queue time and average queue length for each machine group.

PERIOD 8		THIS PERIOD										YEAR TO DATE												
GROUP		Q(H)	Q(L)	Q(T)	Q(H)	Q(L)	Q(T)	Q(H)	Q(L)	Q(T)	Q(H)	Q(L)	Q(T)	Q(H)	Q(L)	Q(T)	Q(H)	Q(L)	Q(T)	Q(H)	Q(L)	Q(T)		
100/236																								
BFA 111					2339			2339			5479			.701			1985			1985			3402	.445
ELN2 111					5094			5094			.000						3100			3100			10	.006
ELN6 111					2824			2824			1609			.374			1775			1775			1372	.436
FITF 111					3787			3787			3585			.486			3047			3047			3797	.497
HMS 111					9797			9797			479			.047			4504			4504			344	.071

Shop Statistics—Inventory Carrying Costs, showing carrying costs and at-machine to on-machine time ratio

TABULATION OF COMPLETIONS																							
PERIOD 8		THIS PERIOD										YEAR TO DATE											
EARLY OR LATE	MACH GROUP	COMPLETED ORDERS				IN PROCESS ORDERS				ALL ORDERS				COMPLETED ORDERS									
		H	L	T	N	H	L	T	N	H	L	T	N	H	L	T	N						
	E15-HP					1			1			1											
	E10-15																						
	E 0-10																						
	E4-8MK																						
	E3-4MK					1			1			1			1				1			1	
	E2-3MK					44			44			52			52			113			113		
	E1-2MK					15			15			23			23			38			38		39
	E4-7DY					3			3			10			10			13			13		0
	E2-8DY					2			2			9			9			11			11		9
	E0-1DY											2			2			2			2		1
						9			9			3			3			12			12		9
	L0-1DY					1			1			1			1			2			2		6
	L2-3DY					3			3			7			7			21			21		8

Shop Statistics—Tabulation of Order Completions, showing count of orders which are completed (or in process) in each early-late category

SHAPE \*. It is completely capable of lending itself to all of the planning and experimentation described above. Its antecedent was the Job Shop Simulator \*\* jointly developed by General Electric and the IBM Corporation, and at this point the substantial contribution of their work should be acknowledged. The fact that the remainder of this article dwells on the extensive changes and departures from that program should not minimize this in any way.

The simulator developed by Westinghouse has three essential features which facilitate schedule generation. They are:

- 1) All shop orders are treated not as linear paths but as networks, which acknowledge the constant change in the combinations of parts being operated upon, and the need for material and component staging at any point in the manufacturing line-up.
- 2) The entire sequence of events which takes place in a simulation is captured and made available later for analysis.
- 3) The program is capable of handling extensive changes in capacity throughout the term of simulation. This includes selective overtime and changes in manned-machine configurations.

In addition to these three major extensions to the program, there is also provided extensive flexibility in the assignment of dispatch rules, and the capability to recognize that empty machines often perform work routed to other machines of a like type.

## II. USE OF NETWORKS IN DEPICTING SHOP ORDERS

In the manufacture of steam turbines, parts are often worked on jointly as a sub-assembly for one operation, and then disassembled and worked on individually in parallel in subsequent operations. The logic by which this process may be handled in a simulator is covered thoroughly in <sup>13</sup>, and will not be dwelt upon at length here. However, Figure 1 shows the fundamentals of the network idea. The entire figure represents one shop order, made up of two sub-orders or feeders, AA and BB. Each part in the manufacturing line-up for the

\* Simulator to Hunt Optimal Production Schedules Highlighting Aggregates and Particular Events.

\*\* *ibid.*

sub-order is given an identifying number which is an integer power of two.\* Each operation in the line-up is coded with a number which is the sum of the identifying numbers for each part being worked upon at that operation. These codes uniquely identify the parts involved. In the diagram, the operation numbers are on the left and the binary codes are on the right. Operation 6 is one which works on the parts labeled 1, 2, 4 and 8 as a sub-assembly, and therefore is coded with a 15. After that operation the parts are disassembled and the parts labeled 1, 2 and 8 are worked on jointly at Operation 7. The part labeled 4 is worked on individually at Operation 8. After Operation 8, this part is assembled with the parts labeled 16 and 32 for joint processing at Operation 12. All of the parts are worked on jointly at Operation 13.

The disassemblies and assemblies represent two types of nodes in the shop order network. There is still a third, which is seen at operation 21. This type of node is a "call-out", and it means that in order to perform Operation 21 the completed part made on the other sub-order BB is required, along with some inventory item H. It is thus a staging operation. One or many call-outs may take place at any operation, and call-outs may be for products made on other sub-orders, or for raw materials. The logic of the simulator prevents call-out operations from starting until all elements called out have arrived.

While the order depicted in Figure 1 may appear somewhat complicated, it is not nearly as complex as many used in steam turbine manufacturing which are presently being simulated on a routine basis.

The upper part of Figure 2 shows a hypothetical form by which an industrial engineer can quickly indicate what combinations of parts are required and where. The lower part shows the implied network. Cards punched from such a form would enable a computer program to assign the necessary binary codes.

## III. SIMULATION AS AN EVENTS GENERATOR

In job shop simulators the modeled shop has productive resources in the form of men, ma-

\* After Kerpelman. See <sup>5</sup>.

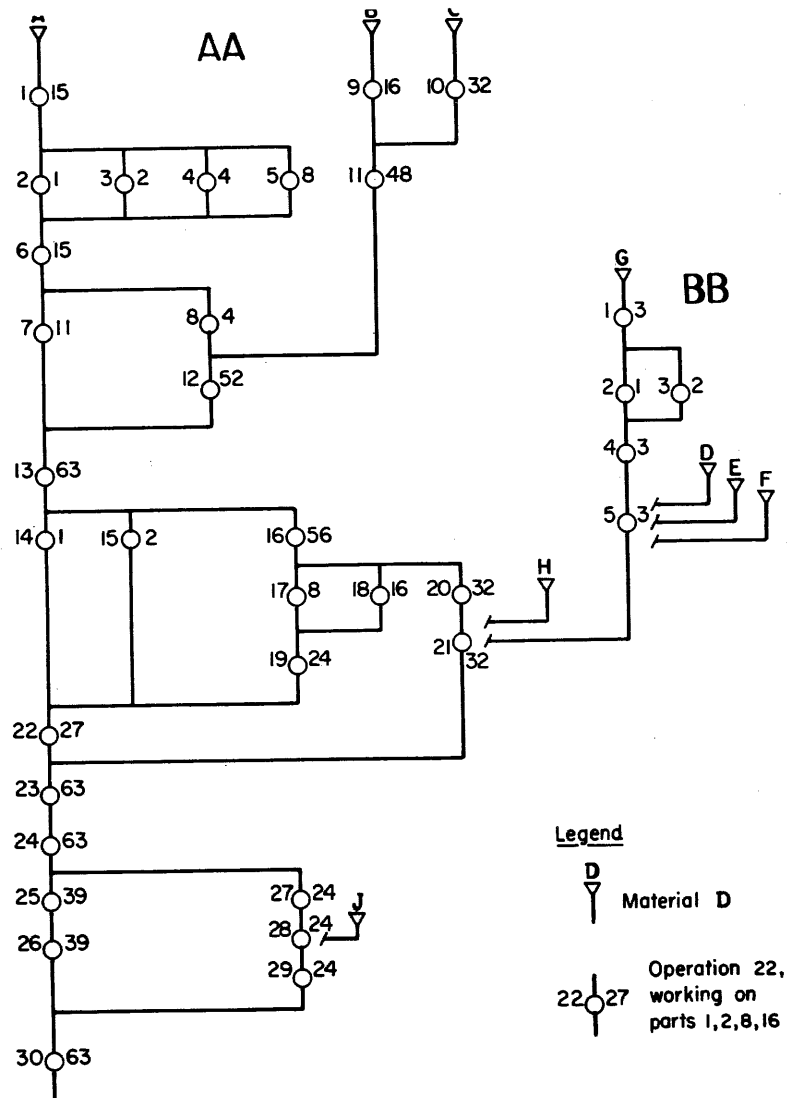


Figure 1. Use of Networks in Depicting Shop Orders

materials and machines; and a load of work to be performed represented by manufacturing data on a set of shop orders. The dynamics of how the model shop processes its load are governed by a set of decision rules. The decision rule which has received the most extensive treatment in the literature is the dispatch rule\*, which reflects the priority system by which jobs are assigned out of queue. The addition of assembly, disassembly, and call-out features adds to the number of decision rules, since the simulator program must rigorously prevent operations from beginning (or even being con-

sidered as in queue) until all necessary parts or components have arrived at the work station where the operation is to be performed.

Simulator decision rules are generally well known, and will not be dwelt upon here. However, it would pay to review several pertinent points about the logical basis of simulators. First, they are queueing devices. When machines are not available, jobs are put in queue behind the machine, just as in a real shop. Thus, *capacity* restraints are specifically recognized. Second, they are sequential devices. Events take place in the simulator sequentially, just as they would in the shop, only in com-

\* 3, 4, 6, 9, 10, or 11.

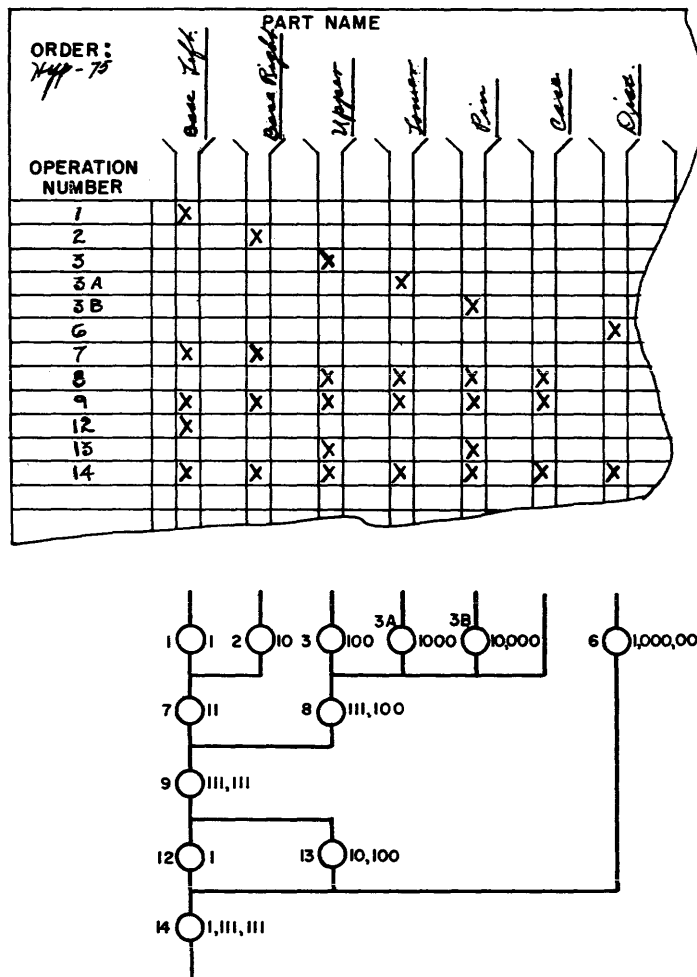


Figure 2. Coding Form and Resultant Binary Coded Network

pacted time. When a situation arises in the model requiring a decision (usually a dispatch) it is made, and all subsequent events are based on its outcome.\*\* The arrival of each job at a machine, its dispatch, and its completion, are events which take place within the model, at certain points in modeled time. At the occurrence of these events, SHOPSHAPE records them and the time they took place. Thus, the model is generating points in time when specific events may be expected to take place, and doing so with full recognition of the decision rules. Wherever possible, the decision rules are

\*\* The importance of this sequential decision process is fully developed by Rowe in <sup>9</sup>. Many of the forthcoming ideas are found there, and in <sup>10</sup>.

designed so that the resulting event sequence reflects desired management policies. By so doing, we establish the satisfactory nature of the way in which the simulator generates events.

#### IV. PARA-SCHEDULES

In looking at the way in which events took place in the model, we may examine them from two different directions. If we look at them on a "by machine" basis, we are looking at a dispatch sequence. If we look at them on a "by order" basis, we are looking at a schedule. The sequencing experienced by the individual operations of an order, leading to the points in time when they were done, is what eventually

led to the order completion date, or the point in time when the last operation was done. This sequencing, or succession in which things were done in the model, may or may not lead to order completion dates which correspond to the master schedule. Therefore, we call the schedules generated by the model "Para-schedules", leaving the term "schedule" to still denote the desired schedule or the master schedule. There are several basic points about para-schedules:

First, these schedules are *resultant schedules*. They are created as a result of what resources were available in the modeled shop to process the load, and what decision rules were used in the model.

Second, para-schedules are *feasible schedules*. There is nothing impossible about them. They could be put into effect immediately, and worked to, if in reality one can provide in the shop the men, machines and work modeled by the simulator. They recognize every form of capacity restriction, and most of the technological restrictions. Specifically, they recognize the entire process of interaction between orders competing for facilities.

Third, para-schedules are highly detailed *predictions* of when work may be expected to be finished, recognizing all corrective measures that are planned to be taken. The matter of their accuracy is discussed below.

Fourth, the para-schedule resulting from a simulation may not necessarily correspond to the desired schedule but it *can be manipulated*. The usual approach in high volume schedule generation is to consider an overload as representing what is desired to be done, but cannot be done. In simulation, overload is impossible. That which would have been an overload is moved down the time dimension. The work goes late. Too late to live with? Quite possibly so, but the schedules can be manipulated and brought to the state most highly advantageous for all factors considered.

Finally, para-schedules are detailed sequences. They represent a *dispatch list*. In the short term they should be quite capable of being worked to. In the long term they obviously will not be met in every detail. But the dispatch list implements the para-schedule that

has been determined to yield the best possible shop results.

#### *Para-schedule Manipulation*

A para-schedule, while being fully feasible, may not be desirable. It may be undesirable for many reasons, but the two principle ones would be the inability of certain critical orders to meet their due dates, and the problem that too much work is in queue behind certain machines. These two examples embody most scheduling problems, and it would pay at this point to hypothecate a schedule analyst and follow what he would do in each case as he works toward an improved para-schedule.

The queuing problem is the simpler of the two. The "Analysis of Queues" in Appendix I shows how long the queues were at each machine group, and how many days each job lost there because of them. Using this report, the analyst identifies coming problem areas. Once found, the question arises, what can he do about them, in order to improve the para-schedule? He does the same thing that would be done in the real shop—he finds some way to increase capacity. Put more men on the glutted machines. Put them on second or third shifts for a few weeks. Put on overtime people—Saturday, or even Saturday and Sunday. How much will this improve the situation? He will simulate again and see. Now that some machine groups are putting more work through, quite possibly new bottlenecks develop at other machine groups. If so, he will take corrective measures on them, resimulate, and have another look. Not many simulations will be required before he has attained the best performance possible, and all the various corrective measures deemed advisable have been invoked.

This procedure does two things. One, it balances manned machine capacity against the load as the mix changes through time; and two, it produces precise *feasible* para-schedules and dispatch sequences. The sequences produced are feasible because no more capacity is put into the model in the form of men and machines than can actually be reproduced on the shop floor.

In the design of SHOPSHAPE, great emphasis was given to recognizing all kinds of ca-

capacity changes, and permitting them to take place at any time for any duration, and for any machines, just as managers are free to do in the real shop. Changes in number of machines, number of shifts, manning and overtime, are all included. Scheduled down time may be incorporated and then shifted if desired. All holidays, weekends, and shutdowns are specifically recognized. The complete implications of every capacity change to be considered on the shop floor is recognized specifically in the model.

Now consider the problems with certain critical orders that failed to get through on time. First, how does the analyst spot them? In Appendix II will be seen a report entitled "Feeder Completion Report." This is an extract of the para-schedule which devotes one line to each sub-order (or feeder) and groups them by shop order. Each line shows how one sub-order fared compared to its scheduled due date, in days late or early. It also shows at which two operations it slipped the most and why; i.e., bottled up in queue, or failure of mating pieces to arrive. Incompleted sub-orders show late or early status at the last completed operation.

The schedule analyst will call for the Order History Diagnostic for all orders which appear to be cause for concern. This is the complete para-schedule for the order, and gives the history and the slippage of every operation. His analysis will begin by looking for any cause where the order is being held up by mating parts or materials, and it is here that the power of the network representation becomes apparent. An example is seen at the first operation of the last sub-order on the report, the Inner Cylinder Assembly. Operation 1 calls out the Inner Cylinder (made on the order just above) and a Nozzle Chamber. Assume the Nozzle Chamber is a part coming from an external source, such as an outside supplier or a feeder shop. (A comparable call-out configuration is seen at Operation 21 of AA in Figure 1.) Even if the Inner Cylinder was available on time, Operation 1 would still be delayed almost four weeks, because the Nozzle Chamber did not arrive until 628.2. Here is a case where the simulator pinpoints a delay that will be caused by an externally supplied part. Such a delay, once

recognized, can usually be avoided by expediting.

What the analyst may do about accelerating the other delaying component, the Inner Cylinder, is somewhat more complicated. Examining the operation-by-operation experience of the order, he will note that 19X is an operation which requires the part being worked on at Operation 17 and the part being worked on at Operation 18. But Operation 18 has encountered a severe queue delay of 426.4 hours, and does not arrive until 628.4. The part from Operation 17 must stay and wait for it for 437 hours\*. In order to decrease the lateness of the sub-order fabricating the Inner Cylinder Assembly, the analyst will have to find some way to accelerate Operation 18 on the sub-order fabricating the Inner Cylinder.

As a first step he could consider releasing it to the shop earlier. This may or may not be sound. After that, he might take whatever measures possible to enhance the priority of the order within the priority system. For instance, if "Earliest Scheduled Start Date Next" is the dispatch rule used on most machines, he might try having the scheduled start dates of operations on the order made earlier.

Wherever the priority for one order is increased, some other orders which compete for the same facilities will in turn suffer, and this may then lead to other points of concern. If there are but few orders of critical concern to local management, the analyst should be capable of improving their schedule performance, but perhaps at some sacrifice elsewhere. As this number of critical orders gets higher, the analyst becomes less and less of a manipulator, and more and more of a juggler. There are more implications to every change he makes, and he becomes increasingly burdened with the same problem facing all schedulers; judgements are made difficult because volume is high and all the work is interdependent. Because of this, with some exceptions, it is more desirable to return to the aggregate measures of shop performance and manipulate machine group capacities, rather than manipulate individual orders.

\* The time a part spends waiting for mating pieces to arrive is called "stay time".

One cannot discuss manipulation without entertaining the problem of what constitutes better schedules. The answer to this lies with the viewpoint of local management, and the policies which rule in their industries. As many managers can be found who consider on-time de-

livery the dominant consideration as consider high machine utilization the most important, yet in a job shop a special emphasis on one must result in a demeaning of the other. The points which are made in the study of dispatch rules hold here as well. The use of certain

E	SHOP ORDER	FEEDER	PART DESCRIPTION	DUE	RESULT	E/L	DELAYS-	OPN	MACH	GP	TYPE	SCHED	START
L	013A0090803	10220002	CYL BASE & CVR EXH STBD	604.0	619.2	11.2	22	L0MC111	5	20	H0M8116	Q	600.8 611.4
M	013A0090803	10910000	STR INLET PIPE	604.0	611.9	7.7						603.2 611.0	
M	013A0090803	77111223	VALVE BODY STBD	604.0	611.0	7.0						602.8 611.0	
M	013A0090803	82041924	STR DUMP TRANS EXH	604.0	611.6	7.5						601.0 611.0	
M	013A0090803	82041934	STR DUMP TRANS EXH	604.0	611.6	7.5						603.0 611.0	
M	013A0090803	82041936	STR DUMP TRANS INL	604.0	611.6	7.5						601.0 611.0	
M	013A0090803	82041936A	STR DUMP TRANS INL	604.0	611.6	7.5						603.0 611.0	
M	013A0090803	82042278	STR DUMP TRANS EXH	604.0	611.1	7.1						603.8 611.0	
M	013A0090804	05001136	ROTOR PORT SER	612.0	621.2	9.2	33	EL40126	5	31	EL40126	S	601.8 611.0
M	013A0090804	09231144	ROTOR	610.4	614.5	4.1	8	L0MC121	8	8	H0P4126	S	604.8 611.0
M	013A0090804	10210007	CYL BASE	611.0	619.8	8.8	8	R0P7111	8	23	F1TF116	S	611.0 611.0
L	013A0090804	10220003	CYL BASE & CVR EXH PORT	611.0	618.4	7.3	13	R0P7111	5	17	L0MC116	S	607.8 612.4
L	013A0090804	10910000	STR INLET PIPE	611.0	612.3	1.2	3	100/236	5	2	M0M8116	S	610.2 611.0
M	013A0090804	77111224	VALVE BODY PORT	611.0	611.8	.8						609.8 611.0	
M	013A0090805	05001137	ROTOR STBD	620.0	622.5	2.4	18	L5S0121	Q	16	H0P4126	Q	609.8 611.0
M	013A0090805	05231144	ROTOR	616.6	627.1	10.7	4	F1TF126	5	9	L5S0121	B	609.8 611.0
M	013A0090805	10210008	SWRN	617.0	625.3	8.3	10	F1TF116	5	2	PLM8116	Q	604.8 611.3
L	013A0090805	10220002	CYL BASE & CVR EXH STBD	617.0	622.6	5.5	2	PLM8116	8	24	X1P1F111	S	611.0 612.4
M	013A0090805	10910000	STR INLET PIPE	617.0	617.1	.1						616.2 616.2	
M	013A0090805	77111223	VALVE BODY STBD	617.0	616.6	.5						615.8 615.8	
M	013A0090805	80101055	NOZ2 CHAMBER	617.0	615.8	1.2						611.0 615.6	
M	013A0090805	82041924	STR DUMP TRANS EXH	617.0	616.6	2.5						614.0 616.0	
M	013A0090805	82041924A	STR DUMP TRANS EXH	617.0	616.6	.5						616.0 616.0	
M	013A0090805	82041936	STR DUMP TRANS INL	617.0	616.6	2.5						614.0 616.0	
M	013A0090805	82041936A	STR DUMP TRANS INL	617.0	616.6	.5						616.0 616.0	
M	013A0090805	82042278	STR DUMP TRANS EXH	617.0	616.7	.5						614.0 616.4	
M	013A0090806	05001136	ROTOR PORT SER	618.0	622.1	4.1	16	H0P4121	Q	41	100/236	S	607.8 611.0
M	013A0090806	09231144	ROTOR	618.4	621.2	2.9	9	L5S0121	Q	4	F1TF126	S	611.0 611.4
M	013A0090806	10210007	CYL BASE	619.0	623.7	4.5	10	F1TF111	5	2	PLM8116	B	608.4 611.4
L	013A0090806	10220003	CYL BASE & CVR EXH PORT	619.0	626.7	7.5	2	PLM8116	8	4	PLM8116	B	611.0 614.4
M	013A0090806	10910000	STR INLET PIPE	619.0	618.4	.2						618.2 618.2	
M	013A0090806	77111224	VALVE BODY PORT	619.0	618.6	.5						617.8 617.8	
M	013A0090806	80101055	NOZ2 CHAMBER	619.0	617.8	1.2						611.0 617.6	
M	013A0090807	05001137	ROTOR STBD	627.0	631.5	4.4	18	L5S0121	B	17	F1TF121	S	616.8 616.8
M	013A0090807	09231144	ROTOR	629.6	629.1	3.7	9	L5S0121	Q	4	F1TF126	S	611.0 618.4
M	013A0090807	10210009	CYL INLET END	626.0	630.9	4.7	2	PLM8116	8	4	PLM8116	B	614.2 614.2
L	013A0090807	10220002	CYL BASE & CVR EXH STBD	626.0	636.1	10.1	2	PLM8116	8	4	PLM8116	S	611.0 621.4
M	013A0090807	10910000	STR INLET PIPE	626.0	625.4	.2						625.2 625.2	
M	013A0090807	77111223	VALVE BODY STBD	626.0	625.7	.5						624.8 624.8	
M	013A0090807	80101055	NOZ2 CHAMBER	626.0	625.8	.4						611.0 624.6	
M	013A0090807	82041924	STR DUMP TRANS EXH	626.0	623.6	2.5						623.0 623.0	
M	013A0090807	82041924A	STR DUMP TRANS EXH	626.0	625.7	.5						625.0 625.0	
M	013A0090807	82041936	STR DUMP TRANS INL	626.0	623.6	2.5						623.0 623.0	
M	013A0090807	82041936A	STR DUMP TRANS INL	626.0	625.7	.5						625.0 625.0	
M	013A0090807	82042278	STR DUMP TRANS EXH	626.0	625.9	.3						625.4 625.6	

Feeder Completion Report

OPN	SCHED	START	E/L	MACH	GP	ORIGINAL	ARRIVE	Q-HRS	S-HRS	ASSEMBLY	CO	CP	COST	JOB	VALUE	STONS	OT	NA	BI
013A0025801 10912444 SLEEVE & PIPE ASSY																			
1	629.6	629.6	.0			F1TF141	629.6						17 8		18	1.5			DP 1
2	629.6	629.6	.0			M0AR141	629.6						104 6		123	9.1			DP 1
3	629.8	629.7	.0			FURN144	629.7						10 3		133	1.0			DP 1
4	630.8	630.8	.0			VACB146	630.8						7 5		113	.7			DP 1
5X	630.8	630.8	.0			F1TF116	630.8						38 8		177	3.3			DP 1
6	631.0	630.9	.3			100/236	630.9						38 8		177	22.5			DP 1
7	631.2	631.1	.0			100/236	631.1						8		177	48.0			LP 1
013A0025801 11000000 INNER CYL																			
1	618.4	619.4	1.1			PLM8116	618.4	157.4					104 6		105	8.4			DP 1
2	618.4	619.6	1.2			PLM8116	618.4	170.2					104 6		105	8.4			DP 2
3	618.4	620.2	1.7			L0MC111	L0MC116	96.1					27 8		133	2.5			DP 2
4	618.4	620.2	1.7			L0MC111	L0MC116	103.0					7 8		113	.7			DP 1
5	618.4	620.4	1.9			R0P0126	620.4						496 6		629	40.8			DP 2
6	619.2	621.0	1.9			R0P0116	620.2	39.5	88	OP NO 4			426 6		1,349	51.4			DP 3
7	619.8	622.1	2.5			L0MC116	622.1			OP NO 5			9 5		1,379	.8			DP 3
8	619.8	622.2	2.6			PLM8116	622.2						261 5		1,012	16.8			DP 1
9	620.2	622.4	2.3			PLM8116	622.2	19.8					204 6		834	16.8			DP 2
10	620.2	622.6	2.3			PLM8116	622.4	2.5	23				92 8		1,938	7.6			DP 3
11	620.4	622.7	2.2			F1TF116	622.6			OP NO 8									DP 3
12	620.6	622.9	2.3			100/236	622.9						203 6		2,142	18.9			DP 3
13	621.6	629.7	2.1			H0P5132	623.7						127 6		2,142	84.6			DP 3
14	621.6	629.9	2.2			F1TF132	623.9						171 8		2,441	13.1			DP 3
15	621.8	624.1	2.5			L0MC132	624.1						35 8		869	2.7			DP 2
16	621.8	624.1	2.6			R0P0132	624.1						35 8		869	2.7			DP 2
17	622.0	625.0	3.1			H0P6111	624.1	144.0					91 8		911	3.2			DP 2
18	622.8	627.5	4.8			H0P6111	624.2	428.4					577 9		2,185	53.7			DP 1
19X	624.0	628.4	4.3			F1TF116	625.9			OP NO 17			742 6		1,653	88.6			DP 2
							625.9			OP NO 18			34 6		3,873	3.2			LP 3
							626.4												
013A0025801 11000001 INNER CYL ASSY																			
1	624.6	628.6	4.0			F1TF141	624.6						65 8		7,348	5.7			DP 1
2	624.6	628.7	3.9			F1TF141	628.6						92 8		7,441	8.0			DP 1
3	624.8	628.8	4.0			F1TF141	628.7						61 8		7,492	3.6			DP 1
4	625.0	628.9	3.7			F1TF141	628.9						92 8		7,574	8.0			DP 1
5	625.0	629.0	4.1			F1TF141	629.0						37 8		7,612	3.3			DP 1
6	625.0	629.1	4.1			M0AR141	629.1						794 8		8,406	69.3			DP 1
7	625.8	629.9	4.1			F1TF141</													



START	SCHED	MACH	SHOP ORDER	FEEDER	OPN	EA	DRAWING ITEM	PART DESCRIPTION	STD HRS	QT	FROM MG	TO MG
616.4	611.6	RDPO116	013A00271101	33000000	3	IP	6340J0489001	BLADE RING 3 GOV	28.6		LMC116	FITF116
616.6	612.0	RDPO116	013A00271301	32000828	3	LA	671 J0058001	BLO RG GOV 2	31.7		LMC116	LMC111
616.8	598.8	RDPO116	013A00906406	10211743	37	MA	6670J0671000	HP CYL FWD DE	3.5		LMC116	FITF116
616.8	612.0	RDPO116	013A00271401	22001426	3	LA	671 J0058002	BLO RG GEN 2	31.7		LMC116	LMC111
616.8	614.2	RDPO116	013A00258801	10000000	11	HP	663J001000	OUTER CYL GOV	86.4		LMC116	LMC116
616.9	616.0	RDPO116	013A00285001	10220000	9	HP	6320J0017000	LP CYL EXHAUST	10.8		LMC116	LMC116
617.0	612.0	RDPO116	013A00271401	32001389	3	LA	671 J0058001	BLO RG GOV 2	31.7		LMC116	LMC111
617.1	616.4	RDPO116	037R00044300	10000000	7	HP	6590J0874001	OUTER CYL	48.2		LMC116	PLM6116*
617.2	616.4	RDPO116	037R00044400	10000000	7	HP	6590J0874001	OUTER CYL	48.2		LMC116	PLM6116*
617.8	597.6	RDPO116	013A00906406	10211743	30	MA	6670J0671000	HP CYL FWD DE	11.1		HM5116	HM5116
617.8	613.0	RDPO116	013A00276301	32002877	3	IP	6650J0037000	BLADE RING 2 GOV	43.0		LMC111	FITF116
617.9	616.6	RDPO116	013A00271101	61990000	3	IP	3600B0523001	CROSSOVER RING	4.0		LMC111	FITF111
617.9	613.2	RDPO116	013A00908605	10220002	8	LA	666 J0935002	CYL BASE & CVR EXH STBD	9.2		FITF116	LMC116
611.0	610.2	VBMK116	013A00270701	11092021	1	LP	365060130001	FLOW GUIDE RING	9.0		***	LMC132
611.1	604.8	VBMK116	013A00265601	61992825	1	IA	788 D0226001	SEALING DIAPH	23.0		***	LMC111
611.4	610.2	VBMK116	013A00278201	04150231	1	LA	785 D0637002	O1SC	30.3		***	YBK121
611.8	611.0	VBMK116	013A00278201	06050094	1	LP	785 D0630002	GEN#5	39.6		***	YBK121
612.3	607.0	VBMK116	013A00266401	11070000	3	HP	6610J0804001	MOZ CHAMBER	11.5		LMC111	VBK116
612.5	611.6	VBMK116	013A00278201	06050094	2	LP	785 D0630002	GEN#5	43.7		VBK121	FITF121
613.4	613.4	VBMK116	013A00248501	22001117	10	IA	668 J0193002	BLADE RING GEN	25.2		***	FITF116
613.9	613.4	VBMK116	013A00265601	11092001	3	LP	6480D0645000	FLOW GUIDE	11.0		HM5116	LMC132
613.9	605.2	VBMK116	013A00275801	13004644	5	HA	669 J0313001	DUMPHY RG	94.7		ROP111	FITF111
615.3	611.2	VBMK116	013A00270601	22001852	5	HA	667 J0857002	BLO RG =2 GEN	40.6		FITF111	FITF111
615.4	616.2	VBMK116	013A00276401	11092191	1	HA	665J00130001	GUIDE RING	9.0		***	LMC132
616.4	611.2	VBMK116	013A00271601	31001043	24	LP	671 J0621001	BLO RG GOV 1	25.2		VAC6146	FITF121
617.0	612.2	VBMK116	013A00270601	21001819	11	HA	667 J0835002	BLO RG =1 GEN	25.3		FITF111	FITF111
617.4	612.2	VBMK116	013A00271101	23000000	5	IP	6340J0489001	BLADE RING 3 GEN	28.7		FITF116	FITF116
611.0	604.8	VBML116	013A00265601	63992883	1	IA	788 D0226001	SEALING DIAPH	23.0		***	LMC111
611.0	591.6	VBML116	013A00906406	10211743	23	MA	6670J0671000	HP CYL FWD DE	345.0		***	FITF116
611.3	607.2	VBML116	013A00275801	21003910	10	HA	663 J0029601	BLO RG =1 GEN END	27.2		***	FITF111
611.8	604.4	VBML116	013A00271001	11000052	8	HA	666 J0125001	BLO RING GOV	76.0		FITF116	BMF5132
612.7	610.4	VBML116	013A00271601	31001043	12	IP	6650J008000	INNER CYL ASSY	64.8		FITF116	VBML116
613.6	611.4	VBML116	013A00276301	11001040	13	IP	6650J008000	INNER CYL ASSY	62.3		VBML116	BMF5132
614.4	603.0	VBML116	013A00248701	31002108	10	LA	668 J0186001	BLO RG	19.6		BLD3134	FITF116
614.7	597.4	VBML116	013A00265601	32001412	21	IA	668 J0178001	BLO RING GOV	28.9		BLD3134	FITF116
615.1	608.0	VBML116	013A00908801	99110058	12	GA	906 J0442002	RSG ASSY	21.7		FITF111	LMC111
615.3	611.2	VBML116	013A00270601	21001819	6	HA	667 J0835002	BLO RG =1 GEN	39.6		LMC111	FITF111
615.4	607.4	VBML116	013A00275801	21003910	18	HP	663 J0029601	BLO RG =1 GEN END	23.8		100/236	FITF116
615.7	608.8	VBML116	013A00275301	92001333	18	DA	669 J0985001	MOZILE CHAMBER 2ND EXTRA	7.4		FITF111	FITF111
616.8	614.0	VBML116	013A00248501	22001117	21	IP	668 J0193002	BLADE RING GEN	11.0		100/236	FITF116
616.6	612.4	VBML116	013A00270601	22001852	10	HA	667 J0857002	BLO RG =2 GEN	21.0		FITF111	FITF111
617.1	612.2	VBML116	013A00271101	33000000	5	IP	6340J0489001	BLADE RING 3 GOV	28.7		FITF116	FITF116
617.1	612.8	VBML116	013A00271001	22005631	5	HA	671 J0666001	O10 RING-2	33.0		FITF111	FITF111
611.2	605.6	VBPM116	013A00276101	11002691	8	HA	664 J0018003	CYL INNER ASSY	42.8		FITF116	BMF5132
611.8	602.6	VBPM116	013A00275901	11090728	4	LA	664 J0688001	FLOW GUIDES	32.3		HM5116	HMAR141
612.2	599.2	VBPM116	013A00911101	0000260	10	DA	672 J0416001	EXH NOZZ	14.9		FITF116	LMC132
612.4	603.0	VBPM116	013A00270601	11001923	19	HA	668 J0132601	CYL INNER ASSY	36.0		FITF116	ROP0116

Short Term Schedule

rules tends to implement certain policies, at some rejection of others. This applies equally well to the schedule analyst, who is also operating under a set of decision rules. If some criteria could be found for establishing a cost of being late, then schedules could be truly optimized and probably fully determined by the computer. Additional research in this area could yield rich returns.

One of the most important functions of the schedule analyst is to learn the sensitivity of the system to changes in different variables. Assume he wishes to accelerate schedules, and is following the aforementioned procedure of adding men to a machine group with a troublesome queue. As one would expect from the lessons of queueing theory, or marginal economic analysis, each successive addition of a man should result in a small absolute reduction in the average queue delay there. In turn, depending on where the jobs are routed to after they leave that machine group, the reduction in delay may result in substantial, little, or no reduction in general aggregated order lateness. At some point it will pay to divert the next intended man increment to some other troublesome machine group. Depending on its queue, its level of utilization, and its importance as a

path in the order networks, the overall system may be highly or hardly sensitive to changes in capacity of a given machine group. While it is often startling to see the difference wrought by the addition of one man-shift at certain machines, it can also be very revealing to see the fruitlessness of another placement elsewhere. It will be up to the schedule analyst to know these sensitivities.

One procedure, which is the converse to the one above, has also been tried to advantage. Here, the analyst begins with the simulation of a shop where all possible capacity is available, no matter what the utilization percentages. This gives an upper limit to the possible schedule performance of the shop. From this point, successive selective cutbacks can be made on the machine groups showing low utilization in the Shop Performance report. Again, as utilizations go up, the extent to which the cutbacks are continued, with their commensurate deterioration in schedules, would be a matter of local management decision. This decision would be based on comparison of the cost of the retained man to the reduction in lost time attributable to him.

The above procedures depict a situation which appears to incur extensive computer

costs, since repeated runs would be made in order to establish the proper balance between load and capacity. Actually, in practice this will not be the case. If, for instance, new para-schedules were generated weekly, then much knowledge gained from last week's run can be put to use in the current run. In addition, orders don't suddenly get into trouble. They either start out in trouble, or work their way into it gradually. In most cases there will be sufficient lead time for the analyst to observe the impact of his corrective measures in succeeding simulations.

Para-schedules can also be influenced by changes in decision rules, such as dispatch rules, subcontracting policies, and others. However, their efforts are studied by separate experiments. If sensitivity to their change proves to be high, then they should be programmed to respond internally as the schedule analyst would manipulate them externally.

It has been shown above that by manipulating machine capacities in the model, and manipulating the priorities within the model, the para-schedules generated are being manipulated. Coming problems are recognized and corrective actions tested weeks or months before the time they will actually be encountered in the shop. If it is impossible to find a para-schedule in which all of the orders meet all of their due dates, then a pretty strong case is made that some commitments will have to be changed. All of the due dates simply cannot be met. Again, the model is available to help pick and choose. All this is done without the slightest gamble in the shop itself. When the analyst has arrived at the best resolution of limited capacities and disappointed customers, he *adopts* the para-schedule which represents the decisions which gave the best result. At the same time he has in hand the major instrument for implementing these decisions: The para-schedule itself and the dispatch sequence which should produce it.

#### *Dispatch Lists and the Fuzzy Future*

The dispatch list is the sequence in which operations are to be performed on each machine. An example is seen in Appendix II under the title "Short Term Schedule." The jobs are listed in the order in which they would be dis-

patched, with start time, job times, and other pertinent data. They may be printed for any interval of time, but what the time span for the Short Term should be is a matter particular to the shop being simulated. The dispatch list is the tool to be used in the shop to implement the desired para-schedule. In theory, if the shop followed it precisely the para-schedule adopted would be met precisely. Obviously, however, this cannot be done. From the instant the list is prepared, things will be happening in the shop to prevent this. Parts may not arrive. Men may report in sick, leaving machines unmanned. Work may be scrapped. Metals may be extra hard, and machining therefore might take a little longer. Nevertheless, in general, the lists may be followed. If the dispatcher always goes to the highest remaining job on the list, he will be implementing the adopted para-schedule.\*

This leads us to the question: For how long into the future are these lists useful? Failure to do one job at the appointed time can have perturbations throughout the shop, since other parts of the list for other machines are based on that job being done at the proper time. One failure compounding another and another can lead to substantial misalignments, when it comes to trying to sequence the machines as the list says. However, the actual results should not be so bad. Here are some reasons why.

Suppose that the job missed was to have proceeded on to a machine where a large bottleneck exists. In such a case, the expected queue time itself at the bottleneck machine will act as a buffer, allowing extra time for the work to actually arrive at the machine before the dispatch time appearing on the dispatch list. On the other hand, suppose the missed job was to have proceeded on to a relatively slack machine. Such a slack machine would be quite adaptive, and capable of handling the work whenever it did arrive, or shortly thereafter. Similar reasoning would hold for machine groups with high traffic. If many machines are working on many jobs, then machines empty and are ready for new work frequently, and at any such point

\* At the El Segundo Division of the Hughes Aircraft Co. a job shop simulator prepares daily dispatch lists for their Fabrication Shop. They are apparently being used with great success. See 11.

previously scheduled work arriving late will be accommodated, tending to put it back on schedule.

One situation which is not easily resolved is the case of a machine group of one or few machines that processes very long operations. Here openings are comparatively rare, and when a scheduled job hasn't arrived to be dispatched, a high level human evaluation is called for. Tying up the only machine for a long time on the wrong job could cause a substantial deviation from the intended schedule. Obviously at the present state of the art, although dispatch sequences are prepared in great detail, they still require human supervision. The need is reduced, but it still remains. Now, when the question is raised of for how far into the future dispatch lists are good, it really means for how long will they continue to require less supervision, rather than more. This remains to be seen.

#### *Feedback and Relations to Support Units*

The setting of due dates for shop orders begins as a matter of management preference. Once established, the standard backward scheduling procedure can be used to determine due dates for supporting components and a scheduled start date for each operation\*. The scheduled start dates are used by the dispatch rules in the simulator. It was seen above that the simulated results may indicate some due dates cannot be met, but the formal act of rescheduling an order may be forestalled, depending on how its dispatch priority within the simulator is influenced. However, the simulator results also bear a relationship to the components included in the order which come from external sources, such as support shops or aisles, or outside suppliers. The para-schedule acknowledges them fully, by means of the "Call-out" mechanism. If information is received that a component will be late, the computer program holds up the order in the model until the time

\* Rowe's flow allowance<sup>9</sup> could be used nicely here, since information on queue time distributions would be available from previous simulations. It is obvious from the lateness figures that the sample reports reflect runs where the simulator is pitted against a backward schedule having a theoretical minimum of slack. These figures should in no way reflect on the ability of the described facility to deliver their orders on time!

the component is expected to arrive. The Order History Diagnostic shows the expected arrival times of the called out components, and shows the slippage, if any, suffered because of delay introduced by them. This means that any component holding up an order is brought to the attention of the analyst so that appropriate pressure may be applied to the source while there is still time. There is still another aspect to this. If an order is running very late, either because of late components or bad queue experience, and nothing can be done about it, then judicious resetting of the due dates for the components may be warranted, especially for those coming from support shops that are already overloaded. New due dates would be based on the required times indicated in the para-schedule.

## V. PROBLEMS OF IMPLEMENTATION

As may be expected, a large number of problems loom in trying to set up such a system. We will touch on a few of them.

### *Supporting Systems*

The validity of the answers generated by the simulator, as in all systems, depends in large measure on the amount of information available to it. This information classifies easily into three parts: configuration information about the shop, manufacturing information on orders comprising the load, and third, the intersection of the first two, which is present shop status. The first consists of such things as machines available, manning, scheduled down-time, and overtime plans. This information is converted into parameters for the simulator. Such data is comparatively easy to capture.

The second set of information consists of all necessary data on all the orders that compose the coming load in the shop. Depending on the size of the shop, this data can be quite massive. It is embodied in a very large "work ahead" file, and the maintenance of this file requires a number of supporting systems. Most of its information is acquired directly from the manufacturing line-up. For the bulk of the items in a job shop, the transfer of this data into the "work ahead" file is not difficult.

Because simulations often run far out into the future, and because job shops must fabri-

cate many products which are newly designed, very often orders must be included in the simulation which are booked or are forecast, but for which no manufacturing data is yet available. Therefore a system of representing this work to the simulator is needed. Several systems have been devised. One is based on composing prototype manufacturing line-ups from similar orders previously completed. Another is based on using Monte Carlo to generate samples from a transition matrix constructed from historical line-ups.\*

The use of component availability dates was discussed above. Such information necessitates still another system to capture these availability dates from the support shops and outside suppliers involved.

The shop status information may be quite difficult to acquire. Historically, in job shops most operational information is not in computer sensitive form. But if simulation is to be used with the precision that has been inferred, then at simulation beginning, the model shop has to be set up to reflect as accurately as possible the present status of the real shop. Such information can come from an extensive data collection system.

A final word about supporting systems. No device, no matter how precisely it processes data, will generate valid answers if the input data is not valid. Everywhere that supporting systems fail to supply useful information, subjective evaluations begin to reappear. There will always be a need for some interpretation and evaluation when appraising the para-schedules generated. Unfortunately such interpretations will grow more and more subjective as the model gets less and less information to work with. If too much interpretation is required, the model has defeated its own purpose as far as generation of schedules is concerned.

#### *Frequency of Simulations*

In discussing dispatch lists, it was brought out that the shop must be expected to go considerably astray of the sequences generated. In addition, it may be expected that the older the dispatch lists become, the less valid they will be. With the passage of time, more and more things will happen to disturb the schedule.

\* See 1.

These would be things that neither the simulator nor any other device could have anticipated. The questions arise then: a) How often should the model be set up again, based on the latest information, and a new simulation run? and b) How far into the future should the simulation run? Much depends on the uses to which the answers are to be put. Generation of dispatch lists should be frequent, but need not extend too far into the future. Long range simulations for load leveling, manpower planning, and facilities planning would extend very far into the future, but are run infrequently. In the simplest sense these considerations are a function of the production cycle time, the number of operations done per day, and the accuracy of the data. There can be, in addition, many other factors such as engineering changes and repair orders, whose effects are more difficult to determine. At present, the answers must be determined empirically for the shop in question.

#### *Unrepresented Production Patterns*

As may be expected, the usefulness of the results of the simulator will be colored by how closely the model is an analogue to the real production patterns in the shop. A case in point is the working of operations out of sequence. The manufacturing line-up infers that this should not be allowed, yet sometimes foremen on the shop floor find cases where they can circumvent this restriction to advantage. The simulator does not recognize such a possibility at all.

Other places where the simulator fails to accurately represent some standing shop practices is in lap phasing, bumping, and the saving of set-ups. Simulators can be written which will do such things. The problem is that the capturing of the necessary information which they need is prohibitive.

#### *Replication*

Each run in a simulation is only one sample from the joint distribution describing possible results from the shop complex. Clearly a greater number of samples, or replications, will improve the accuracy of the predictions embodied in the para-schedule\*. The displays in

\* How much is not certain. See 8.

Appendices I and II reflect the results of a single run, but could easily represent the results of many replications. It is not a difficult data processing problem to merge the results of many runs, and format average figures (with accompanying dispersion measures) instead of a single value.

Unfortunately, the number of runs required to allow for a reasonably stated prediction with very modest confidence limits is prohibitively high in cost.\*\* Naturally, in an industrial environment the cost determines the use. Because of this, the number of replications will be low, and scientifically justified statements on accuracy will be limited. However, it is felt that this does not entirely impugn the usefulness of the simulator as an operational tool, for several reasons. The first is the above-mentioned fact that queues tend to act as buffers. Another is that given a feasible schedule to shoot at, management will put a marked bias on the play of otherwise random events in the shop. Finally, the dispatch sequence generated is at least likely to lead to the goals represented by the adopted para-schedule. Some dispatch sequences which wouldn't lead to the desired schedule have been revealed in prior runs, and have already been eliminated by manipulation of the capacities and priorities in the model. A large number of possible sequences may yet remain which might lead to equally desirable para-schedules, but it is questionable how much would be gained by seeking them out.

### *Scaling*

A major limitation of the scheduling scheme may appear with the inability to fit some shops into the core of the computer. LeGrande reports on problems of this type in <sup>6</sup>. The problem revolves only partly around the number of machines in the shop. A more predominant consideration is the amount of load which must be represented. Each sub-order is on the average half-finished, and all of the unfinished part is in core. Thus, one-half the average number of operations per sub-order, times the average number of orders on the floor, gives the minimum number of operations which must be kept in core during simulation. The program

\*\* The proper number of replications per simulation could be in the thousands. For example, see <sup>14</sup>.

can require from one to two words of core per shop operation, depending on the complexity of the networks. To this must be added the core required for programs (upwards of 8K) and core for tables, which is approximately

$$44G + 3M + 2F + 154 + \sum_{i=1}^G (q_i + k \sigma_i)$$

where

- G is the number of machine groups,
- M is the number of machines,
- F is the maximum number of jobs which will be forced into a queue overflow zone when the queue area for a particular machine group is full,
- $q_i$  is the mean number of jobs in queue for the  $i$ -th machine group,
- $\sigma_i$  is the standard deviation of the number of jobs in queue for the  $i$ -th machine group, and
- k is an arbitrary constant which trades room for speed when searching for jobs in queue.

There are several remedies which may be tried when the model shop becomes too large for core, but they are too involved for discussion here.

## VI. CONCLUSION

Job shops by their very nature make automatic procedures difficult. Yet it is hoped that these experiments will lead to efficiently mechanizing one of the most difficult of the shop's control problems: the schedule-sequencing determination. For the first time it appears possible that at one central logical control point the entire shop, with all of the interaction among orders competing for facilities, can be examined as a unified whole. Ideally, this should reduce the need for segmenting and assigning to the various departments the resolution of the scheduling problems that occur there. This has been done in the past because of the great complexities involved. Those familiar with this practice know that it adds to lead times, and encourages suboptimization.

We have outlined above a method that might be considered somewhat unorthodox to current scheduling concepts, since the detailed sequenc-

ing determines the schedule, instead of vice versa. The full extent of its value remains to be proven. We have tried to accurately portray the problems that exist in implementing such a scheme. It should be pointed out that most of these problems are not especially difficult, or beyond the capabilities of present practices. They are typical implementation problems.

#### REFERENCES

1. BARNES, W. E., "A Case History on Job Shop Simulation," Proceedings, 16th Annual S.A.M.—A.S.M.E. Management Engineering Conference, April 1961.
2. BURKHART, L. J., "Application of Simulation to Production and Inventory Control," 15th Annual S.A.M.—A.S.M.E. Management Conference, April 1960.
3. CONWAY, R. W., and MAXWELL, W. L., "Network Scheduling by the Shortest-Operation Discipline," Dept. of Industrial and Engineering Administration, Cornell University.
4. I.B.M. Math and Applications Dept., "Job Shop Simulation Application," M&A-1., I.B.M., White Plains, N. Y.
5. KERPELMAN, H. C., "Solution to Problems of Assembly and Disassembly Operations in a Job Shop," 19th Annual Meeting, Operations Research Society of America, May 1960.
6. LEGRANDE, EARL, "The Development of a Factory Simulation Using Actual Operating Data," *Management Technology*, May 1963.
7. MARKOWITZ, H. M., HAUSNER, B., and KARR, H. W.; "SIMSCRIPT: A Simulation Programming Language," RM-3310-PR, RAND Corporation, Santa Monica, Calif., November 1962 (also Prentice-Hall, 1963).
8. MORGANTHALER, G. W., "The Theory and Application of Simulation in Operations Research," Chapter 9 of *Progress in Operations Research*, Vol. 1, R. L. Ackoff, ed., N. Y., John Wiley & Sons, 1961.
9. ROWE, A. J., "Toward a Theory of Scheduling," Report S.P.-61, System Development Corp., Santa Monica, California.
10. ———, "Application of Computer Simulation to Production System Design," in *Modern Approaches to Production Planning and Control*, R. A. Pritzker and R. A. Gring, eds., New York, N. Y., American Management Association, Inc.
11. SISSON, R. L., "Sequencing Theory," Chapter 7 of *Progress in Operations Research*, Vol. 1, R. L. Ackoff, ed., New York, John Wiley & Sons, 1961.
12. STEINHOFF, H. W., JR., "Daily System for Sequencing Orders in a Large Scale Job Shop," 6th Annual ORSA/TIMS Joint Western Regional Meeting, Orcas Island, Wash., April 1964.
13. TRILLING, D. R., "Job Shop Simulation of Orders that are Networks," Westinghouse Electric Corporation, Pittsburgh, Pennsylvania.
14. VAN SLYKE, R. M., "Monte Carlo Methods and the PERT Problem," *Operations Research*, September-October 1963.

# HYTRAN\*—A SOFTWARE SYSTEM TO AID THE ANALOG PROGRAMMER

*Wolfgang Ocker and Sandra Teger  
Electronic Associates, Inc., Princeton, New Jersey*

## I. INTRODUCTION

In recent years much attention has been given to combined analog/digital computation, dividing the problem on hand into an analog and a digital part and letting each task be performed most economically by the part of the system which suits it best. This philosophy not only applies to simultaneous analog and digital computation, but also to a sequential use of these two means of computation. One such application is the use of digital computers in the programming and checking of analog computers, a task ideally suited to a digital machine and especially practical in a hybrid installation where the digital computer is most readily available to the analog programmer. HYTRAN, a system of software programs has been developed to provide quick digital assistance in the programming of the analog part of the HYDAC 2400 hybrid computer system \*\*, even to the analog programmer unfamiliar with digital computers.

Since the function of the HYTRAN system is to replace or complement certain manual procedures, it is necessary to briefly consider the manual method of analog programming and checking on which it is based<sup>1</sup>. This method features a two-way static check and can be broken down into the following steps:

- (1) In order to comply with the voltage range of the analog computer every

problem variable is multiplied by a scale factor to form the machine variable in volts.

- (2) Parameters stated implicitly have to be evaluated.
- (3) A computer diagram is prepared, showing the analog hardware implementation of the equations, component modes and the expressions represented on pot-sheets and amplifier sheets.
- (4) Potentiometer settings are determined by evaluating the constant expressions of the scaled equations.
- (5) To check proper scaling and the correctness of the computer diagram, an off-line static check is carried out as follows:
  - (a) The theoretical calculations are performed by substituting into the original equations a test initial condition for each variable that is represented by an integrator output, and solving for their highest derivatives.
  - (b) After the chosen test initial conditions are scaled and entered into the computer diagram as integrator output voltages, the programmer calculates and records on the diagram all component outputs using the voltages present at their input.
  - (c) The computed voltages representing the highest derivatives are com-

\* A service Mark of Electronic Associates, Inc.

\*\* The HYDAC 2400 includes the digital DDP-24 and the analog PACE 231R.

pared with their respective theoretical values, any discrepancies are traced back to their origin and corrected, and finally the static check voltages are recorded on the potentiometer, multiplier, and amplifier sheets.

- (6) The analog patch-panel is wired according to the computer diagram, and the potentiometers are set.
- (7) To insure correct patching and operational hardware, all amplifiers are read out on-line and compared with their respective, computed static check values.

HYTRAN has been written to process digitally the following rather tedious routines of analog programming:

- (1) The calculation of the theoretical *static check values*, voltage static check values, and potentiometer settings.
- (2) The performance of the static check itself, both off-line and on-line\*.
- (3) The complete documentation of the analog program.

Figure 1 shows the steps required to program an analog problem with HYTRAN.

The scaling of the physical equations and the preparation of the computer diagram are still performed by the programmer who thereby maintains direct control over the analog implementation of the problem. In order to permit the calculation of the theoretical static check values, HYTRAN must be given the original problem statement and a set of test initial conditions. To calculate the static check voltages and to perform the static checks, the input has to include the patching according to the computer diagram, component settings or modes, and the highest derivatives with their corresponding integrators and scale factors. Expressions representing other component outputs may be optionally inputted to aid in the automatic pin-pointing of errors during the off-line static check. All inputs are punched on paper

\*Since the use of an ADIOS desk for input/output to the analog computer is of particular advantage with HYTRAN, this discussion presumes its availability for the automatic setting of potentiometers and performance of the on-line static check from data punched on paper tape.

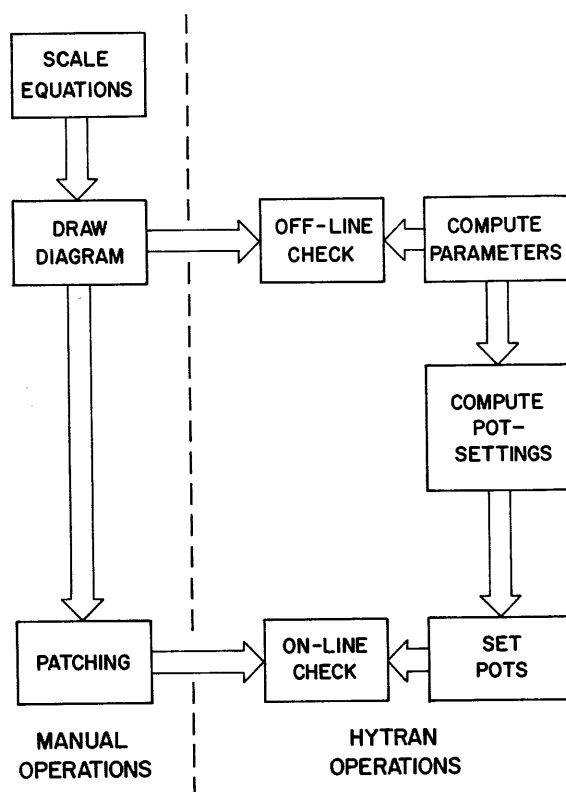


Figure 1. Programming of an Analog Computer.

tape in an analog oriented format, which uses patch-panel terminology and allows complex algebraic expressions.

HYTRAN outputs include the conventional amplifier and potentiometer sheets, cross-referencing and symbol tables. Potentiometer settings and static check values are put out as typewriter documents and on paper tape. The format of this tape allows automatic setting of potentiometers as well as automatic read-out and check of static check values by means of the ADIOS input/output desk. For a more thorough check of the analog set-up and hardware, the ADIOS tape of measured static check values can be processed by HYTRAN, providing a rapid means of locating mismatching or component failures (Figure 2).

## II STATIC CHECK

The practice of computing two independent sets of check values has been used as a basis for the HYTRAN off-line static check. The conventional analog circuit diagram states for at least some components their expected outputs, by an



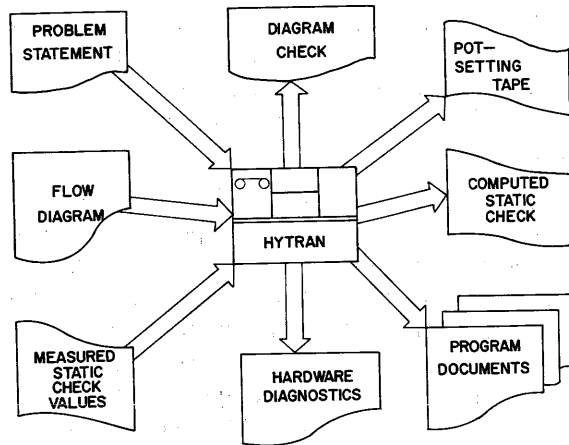


Figure 2. HYTRAN Inputs and Outputs.

expression in terms of parameters, variables and scalefactors. By substituting for the parameter and variable names in these expressions their values in physical units, the so-called *theoretical static check value* is obtained.

Further input defines the analog component interconnections or patching information, which is used to calculate the voltage check values. In this computation all input voltages to a component, the kind of input to which they are connected and the transfer function of the component are used to determine its voltage output.

When both voltage and theoretical values are available for a component output they are compared and, if in agreement, they yield the off-line static check value for that component. If the values do not agree then the error is isolated by retaining the theoretical value as static check value for all subsequent use and an error message is given. Values exceeding the voltage range of the computer will also cause error-messages, but will be retained for further static check calculations.

The amount of input information required depends upon the degree of checking desired. If the expression for some component output is omitted the voltage value is retained as static check value. This has the disadvantage of allowing any undetected error to propagate, rather than isolating it to one component as is done when the expression is given. Conversely, if the patching connections are not stated, the theoretical values are used as static check

values. In this case this particular portion of the computer diagram cannot be checked for consistency with the problem statement.

In the special case of a high gain amplifier, checking can be performed only if the expression at the component output is explicitly stated. The program here checks that the sum of the component inputs is zero. Similarly for an algebraic loop, the output expression of some component in the loop must be given.

The statements defining the analog connections need not be given in any particular order. Since a connection statement can only be evaluated once all input voltages to the component have become defined, the resulting static check values are computed in an order that is different from the order of their input statements. The static check values are punched in this computational order in ADIOS tape format for all components with voltage outputs. This is important for the on-line static check, which if performed by ADIOS in a conventional way cannot distinguish propagated errors from original errors. However, discrepancies typed out in computational sequence are always preceded by the original error, thus eliminating any tracing for the error source. In addition a typeout of the errors in computational sequence and an alphabetical listing of all component names and their static check values is given.

### III THE ON-LINE STATIC CHECK

While in systems without digital access to the analog computer the on-line static check must be performed by manual comparison, the availability of a digital input/output system provides the HYTRAN user with a choice of two automatic procedures in systems using ADIOS. One method is to feed the HYTRAN generated static check tape into ADIOS to obtain an automatic comparison between the calculated and the measured values. It is used whenever the digital computer is not available at the time of the analog on-line check.

However if the DDP-24 is available at on-line check-out time, the use of HYTRAN allows an improved *consistency check* that is expected to become an invaluable tool for debugging of complex problems as well as for preventive maintenance checks. Rather than

merely comparing the measured component outputs with their respective, computed voltages, the HYTRAN on-line check tests the transfer-functions of each component used. To accomplish this the voltages present at the inputs of a particular component as well as its output voltage must be known. The outputs are easily measured since ADIOS allows automatic punching on paper tape of the output voltages of the components used. The inputs on the other hand are either zero (unused) or equal to the voltage at the output of some component, and are therefore easily determined from the connection statements given by the programmer. HYTRAN then computes the output voltages of each component used in the program and compares it with the measured output, using the tolerances stated in the individual component specifications.

In these computations only the measured voltages are used. The resulting theoretical outputs are discarded immediately after the comparison with the measured value. Therefore, errors do not propagate, but are always pin-pointed at the component level. As a result, accumulated errors do not necessarily cause an error message (as they would in the conventional on-line check), but the contribution of each component involved is investigated and checked against its specifications.

It is sometimes desirable to repeat the on-line check at some later time after initial conditions and parameters have been changed. This is possible without the manual entry of such changes, as they are reflected in a change of potentiometer settings. These settings are updated by reading in a paper tape containing the actual settings of all potentiometers used, which can be automatically punched by ADIOS. All algebraic loops, including high-gain amplifier circuits can be checked in closed-loop fashion even if their output is unknown, because the on-line check never uses the given component expressions.

It is also possible to use the on-line check after switching the analog computer into hold mode during a problem run for a reading of component outputs. Because HYTRAN simulates integrators in initial condition mode (i.e., the output is solely determined by the voltage of the initial condition input), a read-out dur-

ing hold causes an error at every integrator output. But since HYTRAN can distinguish between actual errors and propagated errors, the remainder of the program can be checked correctly.

Error messages generated in the on-line check state the erroneous component, the correct output voltage, and the actual voltage measured. We recall that the correct voltage is computed from the component function and the connection statement. Therefore, discrepancies can be caused by component failure as well as by patching errors. These error sources can be separated if the analog computer hardware is thoroughly checked before the program is set-up. This hardware check too can be performed with the HYTRAN on-line diagnostic program, using an artificial problem which has been correctly and permanently wired on an analog patch-panel.

#### IV THE HYTRAN LANGUAGE

The input language used in HYTRAN is oriented toward the analog programming procedure rather than towards the digital machine. Input data can be written in a form which closely resembles the programmer's own way of documenting analog programs. The HYTRAN inputs fall into two main categories: inputs which describe the problem to be solved, and inputs describing its implementation on the analog computer.

##### 1. The Problem Statement

The problem is usually stated in mathematical notation. HYTRAN therefore accepts the problem statement in a mathematically oriented language which bears resemblance to the programming languages ALGOL<sup>6</sup> and FORTRAN<sup>7</sup>.

The problem information to be inputted includes parameter values, variable initial conditions, and a set of algebraic and differential equations—all in terms of physical units. Parameters can be defined by expressions containing numerical values as well as other parameters, while initial conditions of variables can be given in terms of numerical values parameters, or other initial conditions. All algebraic and differential equations have to be in explicit form with respect to the unknown vari-

able, or the highest derivative, respectively. Otherwise, the equations should be inputted in their original form so that any errors that may occur during further manipulations on the problem equations will be revealed or the theoretical static check.

HYTRAN accepts expressions and equations containing not only the basic algebraic operations, but also the functions sine, cosine, arctangent, square root, logarithms to the bases ten and e, and the exponent of e. In addition, the program processes certain discontinuous functions which are often used in analog computation such as absolute value, sign, limits and dead zones.

Relational operators are another means of obtaining step-functions. They are represented by the relations *less than* and *greater than* (for practical purposes, *equal* to does not exist in analog computation). While a logical *and* can be performed by multiplication, the *or* is an explicit Boolean operator in HYTRAN.

## 2. Circuit Diagram Information

An analog program is generated in the form of an analog circuit diagram by which the programmer states the outputs of components, their modes, their interconnections (patching), and in case of potentiometers and switches, their setting or position. Inputting this information enables HYTRAN to check the analog program both against the original problem input and against the physical set-up on the analog computers. The general form of a component statement is:

Component Name and Number, Mode =  
Expression ; Connection Statement

A console number is given only when a change from one console to another occurs. By *mode* one designates the configuration in which a component is used, or any special connections not involving problem variables. For example amplifiers can be connected in integrator, summer or high gain mode. The expression gives the (static check) output of the analog component in terms of problem variables and scalefactors, written in the format used in the problem statement. Finally, a connection statement is defined as a sequence of up to 32 input statements. Each

input statement consists of an input designation (usually identical with the pre-patch panel input name), followed by the name of the analog component which is connected to the specific input.

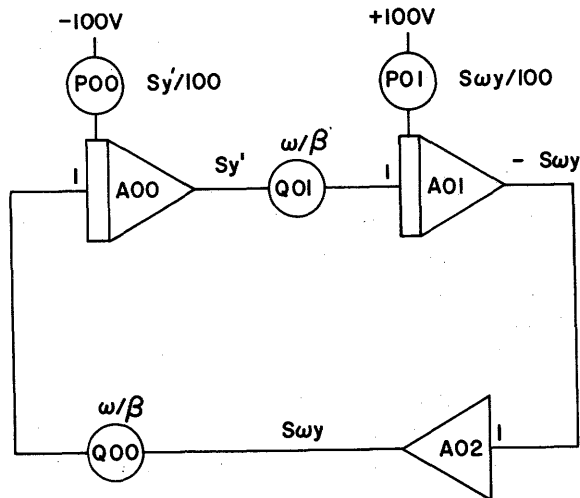
All computer input data are punched on paper tape, each type of information being preceded by one of the following keywords. These input sections must be presented in the order given below, although within any section the statements can be in arbitrary order.

- (1) The **PARAMETER** and **VARIABLE** keywords are followed respectively by parameters as used for the static check, and all variable initial conditions. Parameters and variables may be referred to by mnemonic names which, in turn, can be defined in terms of other parameters, initial conditions, or by numerical values. A name thus defined need not be referred to beforehand, but must be specified within the same keyword section.
- (2) The **EQUATION** portion contains the set of theoretical differential and algebraic equations to be implemented on the analog computer.
- (3) The keyword **COMPONENTS** is followed by the information contained in the circuit diagram which represents the analog program.

As a simple example of how the input information to HYTRAN is written let us consider the case of a second order equation (see Figure 3). The inputs shown in Figure 4 should be provided if a complete check is desired. Note that comments pertinent to the program input, but meaningless to the digital program, can be inserted if preceded by a tab. The input begins with the problem identification, which contains any information the programmer wishes to use as a heading for all typewriter outputs. The resulting outputs are shown in Figures 5 through 7.

## V. THE INDIVIDUAL HYTRAN PROGRAMS

The HYTRAN system presently consists of three programs which together provide the



ANALOG IMPLEMENTATION OF

$$\frac{d^2y}{dt^2} + \omega^2y = 0$$

Figure 3. Sample Problem and Analog Circuit Diagram.

features described above (see Figure 8). These programs are:

- (a) an interpretive (off-line) static check generator,
- (b) an on-line diagnostic program,
- (c) a documenting program.

Each program can be contained in memory, yet allows enough data storage to process three, 120 amplifier, analog consoles in an 8K core.

The off-line static check generator converts the information on the programmer's input tape into a compact form suitable for digital processing. At the same time, it checks the input for proper format, typing format-error messages when necessary. Defined expressions are evaluated immediately, while expressions containing undefined symbols are stored in memory until they become defined by subsequent input statements. Static check voltages are evaluated in a similar manner. Here the *connection statements* may be stored awaiting the calculation of the static check values of all components mentioned in the statement. During the entire process, an *intermediate tape* is punched containing in compact form all in-

## PARAMETERS

S = 5  
A = 20  
OMEG = 10  
BETA = 10  
TO = -1/10

S = SCALE FACTOR  
A = AMPLITUDE  
BETA =  $\beta$  = TIME SCALE

## VARIABLES

Y' = A\*SIN(OMEG\*TO)  
Y = A\*COS(OMEG\*TO)/OMEG

## EQUATIONS

Y'' = -OMEG\*\*2\*Y

## COMPONENTS

P00 = S\*Y'/100;+REF  
A00, I=S\*Y'; IC:P00  
C00 = S\*Y''/(BETA\*10);I:Q00  
Q01 = OMEG/BETA;A00  
P01 = S\*OMEG\*Y/100; -REF  
A01, I = -S\*OMEG\*Y; IC:P01  
C01 = -S\*OMEG\*Y''/(BETA\*10); I:Q01  
A02, S=S\*OMEG\*Y; I:A01  
Q00 = OMEG/BETA; A02

Figure 4. HYTRAN Input Format.

formation necessary to run the remaining HYTRAN programs, including possible future extensions of the HYTRAN system.

The inputs to the on-line diagnostic generator are the intermediate tape punched by the off-line program, a tape of measured potentiometer settings, and a tape of measured component outputs. The intermediate tape provides the connection statements and potentiometer settings necessary for the computation

## STATIC CHECK VALUES

A00 = 47.48  
A01 = 87.99  
A02 = 87.99  
C00 = -8.79  
C01 = 4.75  
P00 = -47.48  
P01 = 87.99  
Q00 = 87.99  
Q01 = 4.74

Figure 5. HYTRAN Static Check Output.

PARAMETERS

A = 2.00000 E1  
 BETA = 1.00000 E1  
 OMEG = 1.00000 E1  
 S = 5.00000 E0  
 TO = 1.00000 E-1

VARIABLES

Y = 8.799416 E-2  
 Y' = 4.74835 E-1  
 Y'' = 8.799416 E0

Figure 6. HYTRAN Symbol Table Output.

of the exact output voltages. The two remaining tapes are direct outputs from the ADIOS input/output desk. Any measured potentiometer setting read replaces the corresponding theoretical setting which was read previously from intermediate tape, however, the reading of the measured settings is optional and can be suppressed by the use of a console switch.

The documenting program sorts and converts the compact information on the inter-

COMPONENTS

COMP	MODE	EXPR	SETT
A00	I	A*S*Y'	
A01	I	-A*S*Y	
A02	S	A*S*Y	
P00		A*S*Y'/100	.4748
P01		A*S*OMEG*Y/100	.8799
Q00		OMEG/BETA	1.0000
Q01		OMEG/BETA	1.0000

CROSS REFERENCE

PARAM	OCCURRENCE
A	P00, P01
OMEG	Q00, Q01
S	P00, P01
Y	P01
Y'	P00

Figure 7. HYTRAN Component Sheets and Cross Reference.

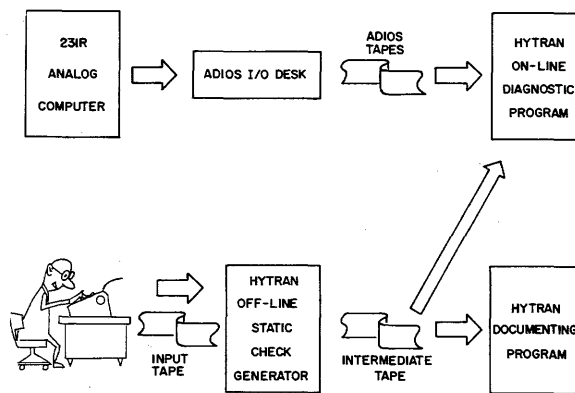


Figure 8. Flow of Information in HYTRAN.

mediate tape, resulting in *component sheets* that contain the analog components in an orderly sequence, together with their modes, and their outputs or settings in terms of problem parameters, variables and scalefactors. In addition, an alphabetic list of values of parameters and variables is typed out, and ADIOS tapes are generated. One ADIOS tape contains all potentiometer settings in a format which allows the automatic settings of potentiometers, the other one contains the computed static check values for on-line static check by ADIOS.

The documenting program finally generates a *cross-reference* sheet which consists of an alphabetic list of parameters and variables. Each name is followed by a list of potentiometers, the settings of which are dependent upon the parameter in question.

VI CONCLUSIONS

HYTRAN is expected to become an important tool to the analog programmer as it increases programming efficiency and justifies a high degree of confidence in the analog solution. Some of the reasons that lead to this conclusion are:

- (1) The automatic evaluation of algebraic expressions saves programming time and prevents arithmetic errors.
- (2) The generation of pot-set tape saves time and eliminates the errors which could occur when transferring the numerical settings from the desk-calculator to the ADIOS keyboard.

- (3) The documentation generated by HYTRAN provides a complete, error-free, and standard documentation of analog programs. Cross-referencing is expected to speed-up the changing of parameters and scalefactors.
- (4) The off-line static check will check every component in the computer diagram. Automatic static check calculations save time and are error free. The checking on the component level allows one to omit the checking of any selected portion of the computer diagram, such as circuits that must be checked dynamically or that are considered standard routines. Such an omission does not prevent the checking of other components in the same algebraic chain.
- (5) Simple changes of connections, parameters or scalefactors may change the majority of the static check values and pot-settings in a program, but only a simple change is necessary on the HYTRAN input tape in order to generate a complete, updated set of ADIOS tapes and documents. Obviously, when a new static check is not required, changes can be made in a conventional way and there is no need to update the input tape for every minor change.
- (6) When the *on-line static check* is performed by ADIOS, the computational sequence of the *static check values* on the HYTRAN generated tape eliminates tracing for the error sources.
- (7) The use of the *on-line diagnostic program* allows pin-pointing of errors on the component level, even for closed algebraic loops of unknown output.
- (8) In conjunction with a permanently wired test problem, the on-line diagnostic program can be used for daily maintenance checks.

Man-power savings from the above benefits out-weight by far the additional effort involved in preparing the HYTRAN input tape, a job that is comparable in size to that of manually preparing potentiometer and amplifier sheets.

The present system is easily expanded to include new features (such as the generation of a digital check solution for example), some of which may evolve from the practical use of the present HYTRAN system. Since such additions will require little additional input information, their benefits will be available at small extra cost and therefore further increase the over-all economy of the system.

#### REFERENCES

1. CARLSON, A. M. and HANNAUER, G., "Handbook of Analog Computation" (Electronic Associates, Inc., 1964).
2. PAYNTER, H., and SUEZ, J., "Automatic Digital Set-up and Scaling of Analog Computers", ISA Transactions, Jan. 1964.
3. GREEN, G., DEBROUX, A., DEL BIGIO, G., and D'HOOP, H., "The Code APACHE intended for the Programming of an Analogue Problem by Means of a Digital Computer", Proc. Int. Assoc. Analog Computation, Vol. 5, April 1963, No. 2.
4. OHLINGER, L., "ANATRAN—First Step in Breeding the DIGNALOG," Proc. WJCC, Vol. 17, May 1960.
5. PROCTOR, W., and MITCHELL, M., "The PACE Scaling Routine for Mercury," Computer Journal, Vol. 5, No. 1, 1962.
6. NAUR, P., et al., Report on the Algorithmic Language ALGOL 60, Com. ACM., Vol. 5, No. 5 (1960).
7. FORTRAN General Information Manual (International Business Machines Corporation, 1961) ed. F 28—8074-1.

# "PACTOLUS"—A DIGITAL ANALOG SIMULATOR PROGRAM FOR THE IBM 1620

*Robert D. Brennan and Harlan Sano  
International Business Machines Corporation  
Research Laboratory  
San Jose, California*

## I. INTRODUCTION

Perhaps the most formidable challenge in the field of digital computer applications is the development of equipment and programs which will extend the creative power of scientific users. The crux of the problem, of course, is intimate man-machine communication—a most elusive and difficult-to-define characteristic. The engineer requires a conveniently manageable system; the scientist requires sufficient intimacy to provide real insight into the complex interplay of problem variables; the creative user requires computing power and flexibility to permit imaginative use of the computer and graphic display to permit recognition of inventive solutions. The development of computers designed specifically for such applications has been slow due to the difficulty of ascertaining the proper man-machine relationship.

Simulation is perhaps unique. It does require close man-machine communication but the nature of this interplay is fairly well known. Years of experience are available in which analog computers were used for simulation in the entire spectrum of scientific disciplines. The structural organization of the analog computer—its collection of specialized computing elements which can be interconnected in almost any desired configuration—makes it a most flexible tool for the engineer, who is trained to visualize a system as a complex of subsystems. The forte of the analog computer is in

the very intimacy of the man-machine communication it permits. Since the nature of the task can be so well defined for this particular application, digital simulation seems a most logical starting point in the quest for more creative use of computers. Only when we are able to perform this task well—with ease and flexibility comparable to simulation using analog computers—only then, should we proceed to those applications where the man-machine relationship is more nebulous.

Digital analog simulator programs—programs which affect the elements and organization of analog computers—are no novelty. Since the first attempt by Selfridge<sup>1</sup> in 1955, there have appeared a number of such programs; best known perhaps are DEPI,<sup>2</sup> ASTRAL,<sup>3</sup> DEPI 4,<sup>4</sup> DYSAC,<sup>5</sup> PARTNER,<sup>6</sup> DAS,<sup>7</sup> JANIS,<sup>8</sup> and MIDAS.<sup>9</sup> Significant improvements have been made in both the interconnection languages and the computational aspects. The latest and most sophisticated of these programs is MIDAS; it incorporates the best features of its predecessors while presenting several important innovations. However, all these previous programs seem to share a common failing in that while they succeed to a greater or lesser extent in using block-oriented languages to express the simulation configuration, they fail to provide the on-line operational flexibility of the analog computer. PACTOLUS is an attempt to focus attention

upon this seemingly ignored aspect of digital simulation.

According to ancient myth, everything that King Midas touched turned into gold. This was fine until dinner time when his food and drink also turned into gold and what had seemingly been a boon wrested from the gods became a curse. To remedy this golden problem, he was advised to bathe in the River Pactolus. Digital simulation programs, particularly MIDAS, have certainly seemed auroreous to many users; yet they must be regarded with mixed feelings, at best, by the engineer who is accustomed to "twisting a pot" or "throwing in a lag circuit" at an analog computer console. To such a user, the remoteness of the digital computer and "turn-around" time seem something of a curse on digital simulation. PACTOLUS is intended to demonstrate that with an appropriate terminal as an operating console, this "curse" can be remedied. PACTOLUS is designed to permit the user to "modify the patchboard," "twist pots," and "watch the plotter" as is the wont of the analog devotee. The name of the program was chosen in deference to the structural and computational excellence of the MIDAS program and to suggest a direction for future development.

## II. GENERAL DESCRIPTION

Simulation is a well-established engineering and scientific tool with applications ranging from simple mechanics to hydrodynamics, aerospace, and bio-medical research. It is presently so widely used that most definitions are unduly restrictive. An excellent definition which avoids this fault is the following: "Simulation is the act of representing some aspects of the real world by numbers or symbols which may be easily manipulated to facilitate their study."<sup>10</sup> Analog and digital computers obviously differ markedly in the manner in which they operate; thus, considerable differences are to be expected between analog and digital simulation. Digital simulation offers significant advantages which have been capably described in References 1-9. The consensus of these is that for many types of problems, digital simulation can provide more reliable and accurate results with less over-all engineering time and effort. Much of this potential has already been

achieved in the various digital analog simulator programs. In part, the development of PACTOLUS was intended as a commentary on the language and computational aspects of these programs. Primarily, however, we are concerned with the second half of the definition; that is, improving the ability of the digital computer user to manipulate the numbers or symbols.

PACTOLUS might be described as a block-oriented interpretive program with on-line control and input-output capabilities. The program, written in FORTRAN II-D, was developed for the IBM 1620 computer with the 1627 Plotter and the Card Read-Punch. This is a comparatively small, scientific computer and several features which might have been incorporated with a larger computer had to be foregone. The overriding concern, however, in the development of this program was the demonstration of operational flexibility. For this purpose, the 1620 with its plotter, typewriter, and sense switches has been an excellent choice. The configuration and parameter specifications may be prepared in advance of a problem session in the form of a deck of cards, or this data may be entered via the typewriter in a simple, convenient manner. If "patchboard wiring" is performed at the console by means of the typewriter, the punch automatically produces a card which may be added to the previously prepared deck. The user, observing the primary output as it is plotted, may interrupt the run at will to modify the configuration, parameters, or initial conditions. The typewriter provides a neat, permanent record of the configuration and parameter values and any modifications specified by the user. In addition, the user specifies those variables of secondary interest which will be recorded by the typewriter at specified intervals during the run.

The program incorporates all the standard analog computer elements—summing amplifiers, inverters, integrators, multipliers, relays—plus many special purpose analog circuits—absolute value, bang-bang, dead space, limiters, clippers, zero order hold circuits. Table I contains the complete list of the available elements and symbols. The program is presently limited to a maximum of 75 blocks in a simulation; no more than 25 integrators or 25 unit delay





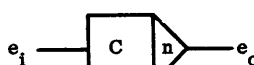
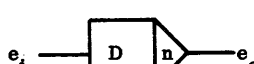




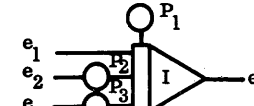
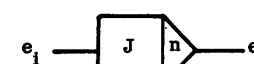
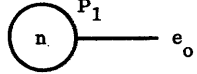
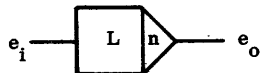

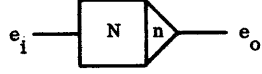
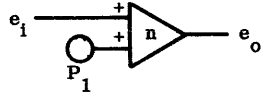
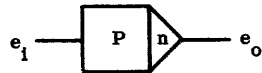
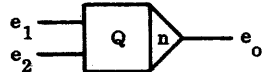
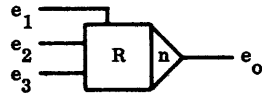
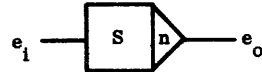

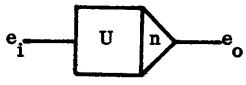
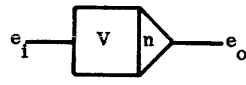
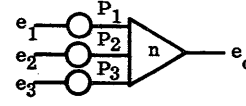
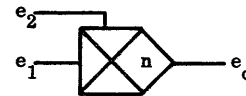
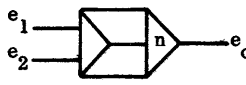
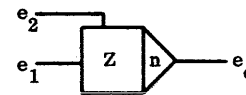
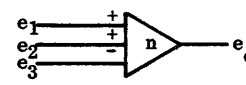
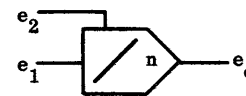
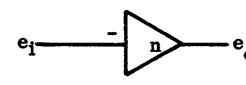
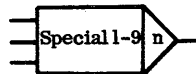
NAME	TYPE	SYMBOL	DESCRIPTION
ARCTANGENT	A		$e_o = \text{ARCTAN}(e_i)$
BANG-BANG	B		$e_o = \begin{matrix} +1 & > 0 \\ 0 & \text{for } e_i = 0 \\ -1 & < 0 \end{matrix}$
COSINE	C		$e_o = \cos(e_i)$ ARGUMENT IN RADIANS
DEAD SPACE	E		$e_o = 0$ $e_i = 0$ $e_o = \text{Max}(0, e_i - P_1)$ for $e_i > 0$ $e_o = \text{Min}(0, e_i - P_2)$ $e_i < 0$
EXPONENTIAL	E		$e_o = \exp(e_i)$
FUNCTION GENERATOR	F		LINEAR INTERPOLATION 10 EVEN SEGMENTS $e_o = f(e_i)$ BETWEEN $e_i = 0$ AND $e_i = 100$
GAIN	G		$e_o = P_1 e_i$
HALF POWER	H		$e_o = \sqrt{e_i}$ SQUARE ROOT
INTEGRATOR	I		$e_o = P_1 + \int (e_1 + e_2 P_2 + e_3 P_3) dt$
JITTER	J		RANDOM NUMBER GENERATOR BETWEEN $\pm 1$

Table I. Definition of PACTOLUS Elements.

NAME	TYPE	SYMBOL	DESCRIPTION
CONSTANT	K		$e_o = P_1$
LIMITER	L		$e_o = \begin{cases} \text{Min}(e_i, P_1) & \text{for } e_i > 0 \\ \text{Max}(e_i, P_2) & \text{for } e_i < 0 \end{cases}$
MAGNITUDE	M		$e_o = \text{ABSOLUTE VALUE } (e_i)$
NEGATIVE CLIPPER	N		$e_o = \begin{cases} 0 & \text{for } e_i \leq 0 \\ e_i & \text{for } e_i > 0 \end{cases}$
OFFSET	O		$e_o = e_i + P_1$
POSITIVE CLIPPER	P		$e_o = \begin{cases} 0 & \text{for } e_i \geq 0 \\ e_i & \text{for } e_i < 0 \end{cases}$
QUIT	Q		QUIT (TERMINATE RUN) IF $e_1 > e_2$
RELAY	R		$e_o = \begin{cases} e_2 & \text{for } e_1 \geq 0 \\ e_3 & \text{for } e_1 < 0 \end{cases}$
SINE	S		$e_o = \sin(e_i)$ ARGUMENT IN RADIANS
TIME PULSE GENERATOR	T		GENERATES PULSE TRAIN WITH PERIOD EQUAL TO $P_1$ FIRST PULSE OCCURS WHEN $e_i \geq 0$

NAME	TYPE	SYMBOL	DESCRIPTION
UNIT DELAY	U		$e_o = e_i(t - \Delta t/2)$
VACUOUS	V		$e_o \neq f(e_i)$ USED IN CONJUNCTION WITH ELEMENT WYE
WEIGHTED SUMMER	W		$e_o = P_1 e_1 + P_2 e_2 + P_3 e_3$
MULTIPLIER	X		$e_o = e_1 e_2$
WYE	Y		LOGICAL BRANCH ELEMENT USED FOR IMPLICIT OPERATIONS
ZERO ORDER HOLD	Z		SAMPLES WHENEVER $e_2 > 0$
SUMMER	+		$e_o = \pm e_1 \pm e_2 \pm e_3$ ONLY ELEMENT WHERE NEGATIVE SIGN IS PERMISSIBLE IN CONFIGURATION SPECIFICATION
DIVIDER	/		$e_o = e_1 / e_2$
SIGN INVERTER	-		$e_o = -e_i$
SPECIAL 1-9			SUBROUTINES SUPPLIED BY USER

n REPRESENTS THE BLOCK NUMBER

elements may be used. The program is also restricted at present to a maximum of three function generators. The structure of the program is sufficiently simple that modifications or additions to the set of PACTOLUS elements may easily be made to accommodate the requirements of particular users.

The innovation in PACTOLUS is its attention throughout to operational flexibility. In most other respects, PACTOLUS represents a conscious synthesis of those features which in our opinion are the best of the many previous programs. Its interconnection language is modeled on that of ASTRAL which is flexible yet simple. Each block is identified by number which also identifies the output from the block. The type of element for each block is specified by a single alphanumeric character or mathematical symbol. The inputs to a block are specified by listing the block numbers of those elements which provide the inputs. Structurally, PACTOLUS is an interpretive program and is closely modeled on MIDAS. Like MIDAS, it uses the excellent sorting procedure—a logical test for determining the proper order for the block computations—which had been a feature of the ASTRAL program. The second-order Runge-Kutta integration scheme used in PACTOLUS is a compromise between the Euler integration advocated by the developers of DAS and PARTNER and the more sophisticated formulas used in many of the other programs. For those simulations which commonly involve discontinuous functions, the use of higher-order numerical integration formulas seems unwarranted; apart from accuracy considerations, the requirements of the output plotter demand fairly small time increments even at the cost of prolonged solution time.

In addition to the complement of computing elements provided with most of the digital analog simulator programs, PACTOLUS includes an element similar to the Implicit Function element which is one of the outstanding features of MIDAS. This element is used for the solution of equations of the form  $Y = f(Y)$ . It permits iteration without advancing the time clock until convergence is (hopefully) achieved. The implementation of this feature seems somewhat superior in PACTOLUS; if the convergence criterion is not satisfied, the computa-

tion proceeds again through those blocks in the algebraic loop. MIDAS appears to recompute all the preceding blocks on the sort list, regardless of whether or not they pertain to the Implicit Function. This modification would seemingly result in a significant reduction of solution time for large problems involving Implicit Functions.

Another modest contribution in PACTOLUS is the incorporation of a number of special elements of unspecified function. An interconnection specification for one of these special elements results in a subroutine call during the interpretive portion of the program. The user may design any complex function for this element by development of the appropriate subroutine. This permits the user to add elements to the standard complement without reprogramming the main PACTOLUS program. The JANIS program, although structurally quite different, must be credited with first utilizing subroutines to achieve this “do-it-yourself” capability.<sup>11</sup>

### III. OPERATING PROCEDURE AND EXAMPLES

The simulation configuration is the specification of the interconnection of the computing blocks, where each block is one of the standard set of PACTOLUS elements or one of the nine special element blocks which the user may prepare for his particular requirements. Each block has but a single output and no more than three inputs. The program is incapable of handling algebraic loops; the existence of such a loop will result in a “sort failure” message. Since the digital computer is a serial device, the various computations specified by the simulation configuration must be performed for each time cycle in some particular order. The computation for any block should not be attempted until all its inputs have been computed during that iteration. The program automatically performs a sorting operation to achieve this ordering after each configuration change. It is presumed that each simulation involves at least one integrator; the program uses a simple second-order Runge-Kutta numerical integration formula to approximate the outputs of the integrators during the specified integration interval.

*Example 1:* Use of the program is perhaps best understood by consideration of several simple examples. Figure 1 is a simulation diagram for a second-order system. The program presumes that ordinarily the user will approach the computer with most of the "patchboard" pre-wired; that is, with a deck of cards specifying the configuration and parameter values. For this simple example, however, it is as easy to put a deck of blank cards in the card reader and do the "wiring" at the console. Figure 2 shows the typewriter record from this problem session. The user first turned on Sense Switch #1 to indicate his intention to enter configuration specifications from the typewriter. The computer then typed the following:

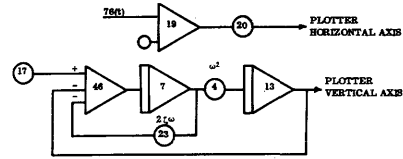


Figure 1. Simulation Diagram for Second-Order System Example.

CONFIGURATION SPECIFICATION  
 BLOCK TYPE INPUT 1 INPUT 2 INPUT 3  
 ( ) ( ) ( ) ( ) ( )

The user then turned the typewriter paper roller back one line and inserted the first specification within the parentheses. After he pressed the RS (release and start) key, the

```

PACTOLUS DIGITAL ANALOG SIMULATOR PROGRAM

          CONFIGURATION SPECIFICATION
BLOCK   TYPE   INPUT 1   INPUT 2   INPUT 3
(17)    (K)    ( )         ( )         ( )
(46)    (+)    ( 17)      (-13)      (-23)
( 7)    (I)    ( 46)      ( )         ( )
(23)    (G)    ( 7)       ( )         ( )
( 4)    (G)    ( 7)       ( )         ( )
(13)    (I)    ( 4)       ( )         ( )
(19)    (O)    ( 76)      ( )         ( )
(20)    (G)    ( 19)      ( )         ( )

          INITIAL CONDITIONS AND PARAMETERS
BLOCK   IC/PAR1   PAR2   PAR3
(17)    ( 0.0 )   ( )     ( )
(13)    (-100.0) ( )     ( )
( 4)    ( 1.0 )   ( )     ( )
(23)    ( 0.8 )   ( )     ( )
(20)    ( 20.0 ) ( )     ( )
(19)    (-5.0 )   ( )     ( )

( 0.05 ) ) INTEGRATION INTERVAL
( 10.0 ) ) TOTAL TIME
( 2.0 )  ) PRINT INTERVAL
(20)    ) HORIZONTAL AXIS
(13)    ) VERTICAL AXIS

          TIME          OUTPUT(46)   OUTPUT( 7)   OUTPUT(13)   OUTPUT(23)
0.000   100.00000   0.00000     -100.00000   0.00000
2.000   -30.57587   47.30857     -7.27098     37.84686
4.000   -13.03217   -11.02521    21.85234     -8.82016
6.000    9.20057    -6.96698     -3.62699     -5.57358
8.000    4.82822    3.84609      -3.55969     3.07687
10.000  -1.96715        .50806       1.56070      .40645

          INITIAL CONDITIONS AND PARAMETERS
BLOCK   IC/PAR1   PAR2   PAR3
(23)    ( 1.2 )   ( )     ( )

          TIME          OUTPUT 46   OUTPUT 7    OUTPUT 13    OUTPUT 23
0.000   100.00000   0.00000     -100.00000   0.00000
2.000   -23.43237   37.60320     -21.69146    45.12384
4.000   -8.64924   -6.54644     9.43481      -7.78556
6.000    2.27228   -3.39439     1.80038      -4.07327
8.000    .74380    .11838       -.88587       .14206
10.000  -.21880       .30537       -1.14764     .36645

          INITIAL CONDITIONS AND PARAMETERS
BLOCK   IC/PAR1   PAR2   PAR3
(23)    ( 0.4 )   ( )     ( )

          TIME          OUTPUT 46   OUTPUT 7    OUTPUT 13    OUTPUT 23
0.000   100.00000   0.00000     -100.00000   0.00000
2.000   -38.09478   63.26133     12.79025     25.30453
4.000   -25.50784   -32.19054    38.38406     -12.87621
6.000    30.08132   -12.01936    -25.27358     -4.80774
8.000   -3.85580     20.56715     -4.37106      8.22686
10.000  -11.54220     -5.06982     13.57013     -2.02792
    
```

Figure 2. Typewriter Record for Second-Order System Example.

typewriter automatically typed another line of parentheses in anticipation of the next specification. In addition, as each configuration specification was entered from the typewriter, the punch produced a card with the identical data. The collection of these cards forms the wired "patchboard" which may then be saved for subsequent problem sessions, thereby eliminating the necessity of re-typing the specifications. Just as the analog user merely needs to get the patch-cord into a hole to make a connection, the PACTOLUS user need only get the proper block number within the parentheses; he is not distracted at the console by complicated input format requirements.

The equivalence of the format of the configuration specifications shown in Figure 2 to the simulation diagram of Figure 1 should be obvious. Block 17 is a Constant input (K). Block 46 is a Summer (+) with inputs from blocks 17, 13, and 23; sign inversion is indicated for the latter two. In a similar manner, blocks 4 and 23 are Gain potentiometers (G) and blocks 7 and 13 are Integrators (I). Thus, each component is uniquely identified by a block number. The type of element for each block is specified by a single letter or symbol. These have been assigned as either the first letter of the name of the element or as the common mathematical symbol for the operation. Table I contains the complete list of elements and the corresponding symbols. The inputs to a block are specified by the block numbers of the components which provide the inputs. Block numbers may be assigned arbitrarily between 1-75. Block 76 by definition is the time variable; it appears as the input to block 19 since it is desired to plot the time response of the second-order system versus time. It should be noted that the plot size is fixed at 10 inches square. The origin is at the center and each axis is scaled for a maximum of  $\pm 100.0$ . In this example, blocks 19 and 20 correspond to the horizontal position and gain controls of the conventional X-Y recorder.

After entering the last configuration specification, the user turned off Sense Switch #1 and turned on #2 to enter initial condition and parameter values from the typewriter. Once again, the typewriter typed a line of parentheses to indicate the proper format. Figure 2

shows that the user specified an initial condition for Integrator 13 at  $-100.0$  and the damping factor control, Gain potentiometer 23, at  $0.8$ . After each of these parameter specifications, the punch again produced a card; thus, both the configuration and the parameters became part of the permanent deck. After entering the last of these specifications, the user turned off the sense switch. The typewriter then requested timing data for the run and specification of those outputs which were to be plotted and those which were to be printed. These entries are simply performed since it is only necessary that the typing be within the parentheses provided. The actual time required to "wire at the console," "set the pots," and "adjust the output devices" for this example was  $6\frac{1}{2}$  minutes. This time cost would seem to be comparable to that required for the equivalent analog setup, particularly if we insist that the analog operator prepare a neat, permanent record of the interconnection and all parameter values.

The plotter record is shown in Figure 3. Three runs were made, each for a different value of the damping factor. Figure 2 shows the parameter changes for Gain block 23. Each run required 1 minute, 45 seconds; the "pot" changes required 15 seconds each. Run times might have been shortened by increasing the

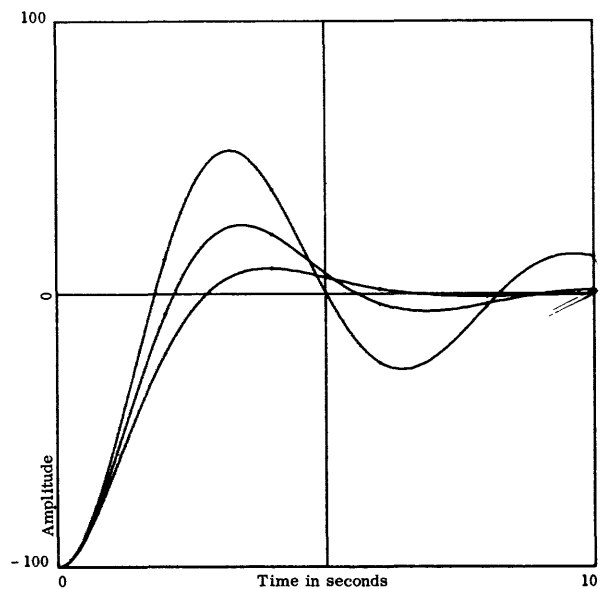


Figure 3. Plotter Record for Second-Order System Example.

typewriter print interval or the integration step.

*Example 2:* Let us next consider a slightly more complicated example which illustrates how PACTOLUS might be used for a study in speech synthesis. This particular simulation is concerned simply with the response of the first speech formant when driven by a glottal pulse. The first formant corresponds roughly to that portion of the speech signal which remains after passing the signal through a 1 KC low-pass filter. The glottal pulse is closely modeled by the triangular waveform produced by block 6 of Figure 4; the effect of the vocal tract is simulated by blocks 7–9. During utterance of a stop consonant such as the “b” of the word “bah,” the lips must open quickly with a resulting rapid rise in the natural frequency of the vocal tract. This variation in frequency of the first formant is controlled by blocks 10–12. The user might experiment with various frequency changes to match actual speech records. To add the second or third formants, it would only be necessary to add additional second-order filters and their associated frequency controllers.

In preparation for this problem session, the user entered the configuration and parameter specifications on special coding forms which minimize concern with data format; from these coding forms, a deck of cards was punched. In

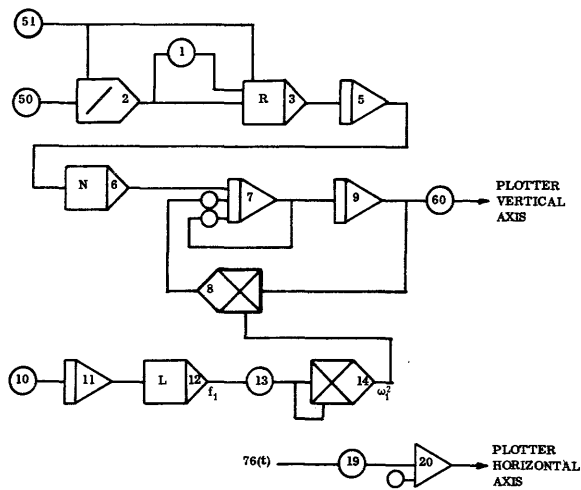


Figure 4. Simulation Diagram for Speech Synthesis Example

this sense, the user approached the console with a “pre-wired patchboard.” The specifications for blocks 10–12 were omitted, however, as might have occurred from oversight or from early indecision with the form of this formant frequency controller portion of the simulation. The program caused the configuration specification cards to be read until a blank card was encountered; each of these statements was also printed by the typewriter for the convenience of the user. Since a block 12 had been specified as the input to block 13, but block 12 was as yet unspecified, the program recognized an operator error. After listing the sort failures as an aid for debugging, the typewriter produced a line of parentheses, anticipating that the user would wish to correct the error or omission. The user then turned on Sense Switch #1 until he had entered blocks 10–12. The program then proceeded to read the parameter specification cards until a blank card was encountered. During this period, the user had turned on Sense Switch #2 to permit typewriter entry of the parameters for blocks 10–12.

Figure 5 shows that the user neglected to depress the numeric shift key of the typewriter when entering the total time for the run. To recover from this or any other typing error, the user simply turns on Sense Switch #4 prior to pressing the RS key. He would then turn the switch off and reenter the data on the next line. Figure 6 shows the plotter output from this run. The total time for reading the input deck, adding the three blocks, adjusting their parameters and performing the run was 11 minutes. The user finally changed the configuration and repeated the run in order to plot the “glottal pulse.” Note that for this purpose, it was not necessary to change parameters, timing, or output data; a series of runs can be performed with a minimum of effort.

*Example 3:* The third example is a simulation of a sampled-data feedback control system. Figure 7 is a diagrammatic representation of the system; Figure 8 shows the corresponding simulation diagram. The objective of this study was to obtain a digital compensator design which would permit the system output to respond in an acceptable manner to the flat-topped ramp input. Of particular interest is the manner in which the Time Pulse Generator

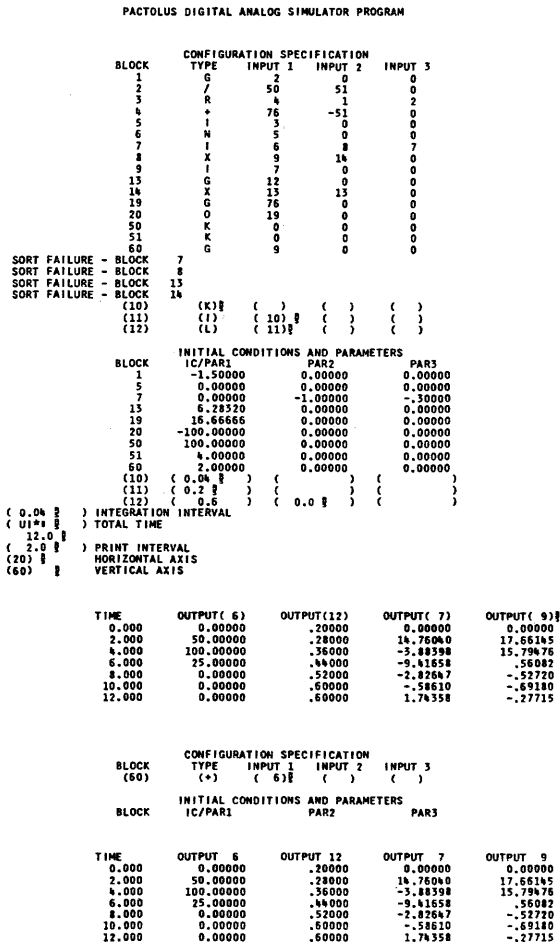


Figure 5. Typewriter Record for Speech Synthesis Example.

element is used in conjunction with Zero Order Hold and Unit Delay elements to implement the digital compensator. Block 50 produces a series of pulses at the sampling rate; these pulses trigger the hold elements. By alternating hold and delay elements, one may obtain the sampled-data operators  $z^{-1}$ ,  $z^{-2}$ , etc. The output of the system is shown in the plotter record, Figure 9. A number of runs were conducted with various values for the parameters of blocks 12 and 13 which determine the gain and weighting factors for the digital controller. Each of these runs required 1½ minutes, exclusive of the time required to decide upon the next set of values.

*Special Elements 1-9*

When a user finds recurring use for a particular operation, it may be advantageous to

define a Special element rather than resort to a number of the standard PACTOLUS elements. For instance, one might prepare a defining subroutine for element Special 1 to perform the function of the digital controller of the third example:

$$\frac{E_o(z)}{E_i(z)} = \frac{k(1-a z^{-1})}{(1-b z^{-1})}$$

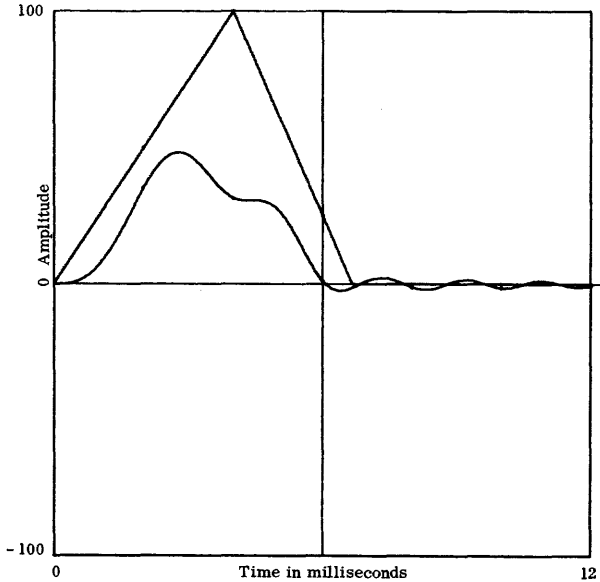


Figure 6. Plotter Record for Speech Synthesis Example.

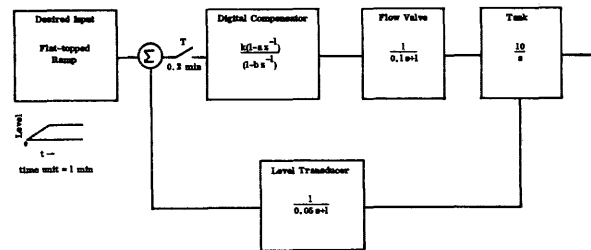


Figure 7. Diagrammatic Representation of Sampled-Data Feedback Control Problem.

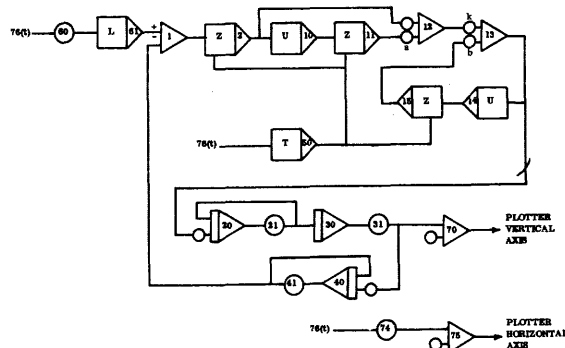


Figure 8. Simulation Diagram for Sampled-Data Feedback Control Problem.



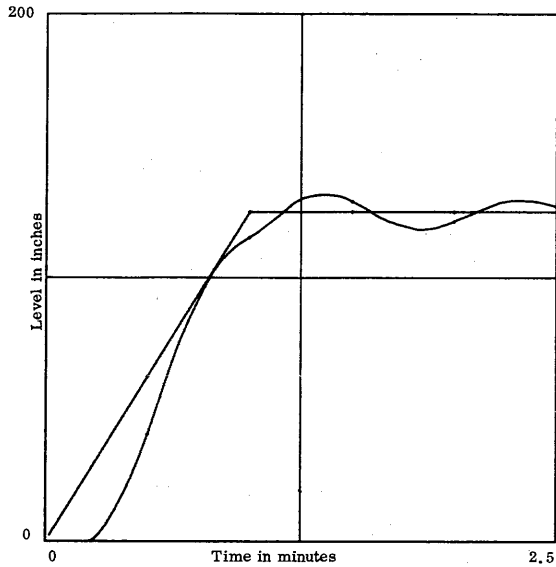


Figure 9. Plotter Record for Sampled-Data Feedback Control Problem.

Such a program might be easily written in FORTRAN or, for a large simulation in which speed was critical, in symbolic machine lan-

```

SUBROUTINE SUB1(C,PAR,I,J,K,L)
C   INPUT IS C(J)
C   OUTPUT IS C(I)
C   DIMENSION C(76),PAR(75,3)
C   IF ( C(76) ) 2,1+2
C   INITIALIZE AT T = 0
1   CAY = PAR(I,1)
   A = PAR(I,2)
   B = PAR(I,3)
   EO = 0.0
   ZM1EI = 0.0
   ZM1EO = 0.0
2 IF ( C(K) ) 4,4+3
C   SAMPLING OCCURS WHEN SECOND INPUT IS POSITIVE
3   EI = C(J)
   EO = CAY*(EI - A*ZM1EI) + B*ZM1EO
   ZM1EO = EO
   ZM1EI = EI
4   C(I) = EO
RETURN
END
    
```

Figure 10. FORTRAN Program Example for Definition of Element "Special 1."

guage. Figure 10 shows a FORTRAN program which would perform this operation; it is important to note that due to the simplicity of the PACTOLUS program, only modest programming skill is necessary for the definition of the element. This new element, Special 1, can then replace blocks 2-15 of Figure 8. The typewriter record illustrating use of this element as block 13 is shown in Figure 11.

PACTOLUS DIGITAL ANALOG SIMULATOR PROGRAM

BLOCK	CONFIGURATION TYPE	INPUT 1	INPUT 2	INPUT 3
1	+	61	-41	0
13	1	1	50	0
20	1	21	13	0
21	G	20	0	0
30	1	21	0	0
31	G	30	0	0
40	1	41	31	0
41	G	40	0	0
50	T	76	0	0
60	G	76	0	0
61	L	60	0	0
70	O	31	0	0
74	G	76	0	0
75	O	74	0	0

BLOCK	IC/PAR1	PAR2	PAR3
20	0.00000	-1.00000	0.00000
21	-10.00000	0.00000	0.00000
30	0.00000	0.00000	0.00000
31	10.00000	0.00000	0.00000
40	0.00000	-1.00000	0.00000
41	-20.00000	0.00000	0.00000
50	.20000	0.00000	0.00000
60	125.00000	0.00000	0.00000
61	125.00000	0.00000	0.00000
70	-100.00000	0.00000	0.00000
74	80.00000	0.00000	0.00000
75	-100.00000	0.00000	0.00000

( 0.02 ) INTEGRATION INTERVAL  
 ( 2.5 ) TOTAL TIME  
 ( 0.5 ) PRINT INTERVAL  
 (75) HORIZONTAL AXIS  
 (70) VERTICAL AXIS

TIME	OUTPUT(13)	OUTPUT(21)	OUTPUT(31)	OUTPUT(41)
0.000	0.00000	0.00000	0.00000	0.00000
.500	24.87666	21.64980	40.72684	31.04116
1.000	6.95500	4.65402	115.54003	112.51337
1.500	-5.72807	-3.77466	128.97023	130.19467
2.000	3.14409	3.20176	121.58805	170.33700
2.500	-3.21131	-2.35386	127.13260	128.00924

Figure 11. Typewriter Record for Sampled-Data Feedback Control Problem.

*Implicit Operations:* The “Wye,” logical branch element is used in conjunction with the Vacuous element V for simulations which involve implicit equations of the form  $Y = f(Y,X)$ . On a digital computer, such an equation is usually solved by iteration. In a realistic problem, Y will often be merely an intermediate variable used in subsequent equations of the form  $Z = g(Y,X)$ . It is the function of the Wye element to determine whether the iteration on Y is sufficient to satisfy the error criterion. If not, further iterations must be made until the test is (hopefully) satisfied. This iterative procedure must be performed within each of the integration time steps.

There is no need, however, to recompute X at each iteration since it is independent of the iteration. Similarly, the computation of Z ought not be attempted until the iteration on Y is satisfied. The MIDAS program was the first of its kind to have a capability for implicit equations, but it does not presently incorporate these considerations. The implementation in PACTOLUS is presented (as a peace offering?) for possible improvement of MIDAS and subsequent programs.

The manner in which the elements are to be used is indicated in Figure 12. Although block numbers may be assigned arbitrarily, for illustrative purposes they have been assigned in the same sequence as that determined by the

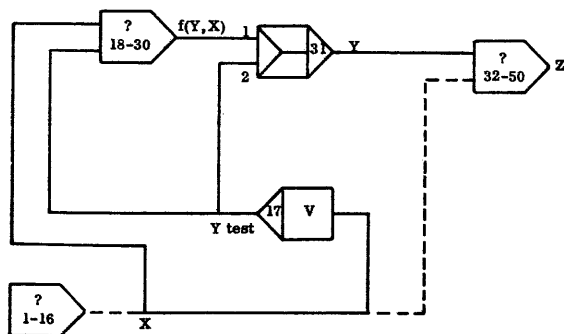


Figure 12. Example Use of the Wye and Vacuous Elements for Simulations Involving Implicit Equations.

sorting operation. Blocks 1–16 compute a quantity X. The initial estimate for Y is given as an initial condition for V block 17. Blocks 18–30 compute  $f(Y,X)$ . Block 31 compares  $f(Y,X)$  with the previous estimate of Y. Parameter 1 of the Wye element specifies the relative error criterion. If the relative error between the outputs of blocks 30 and 17 is less than specified by that parameter, then the output of block 31 is set equal to that from block 30. Computation of Z then proceeds in normal manner.

If the error criterion is not satisfied, the operation of Wye block 31 is as follows:

- (1) A new estimate for Y is computed using parameter 2 of the Wye element
 
$$Y_{n+1} = (1 - P2) f(Y_n, X) + P2Y_n ;$$
- (2)  $Y_{n+1}$  replaces the previous  $Y_n$  as the output of block 17;
- (3) The program “branches back” to the computation of that element on the sort list which follows the V block—in this case, it branches to block 18.

The inputs, if any, to the Vacuous element V do not affect its output. Its initial output is specified as an initial condition; its subsequent output is determined by the associated Wye block. The position of the V element in the sort list follows that of each of its inputs. Its purpose is to ensure that, when a “branch back” occurs, these preceding blocks will not be recomputed. There is no requirement that the implicit equation actually involve any of the inputs to the Vacuous block.

### Sense Switches

An important factor in the flexibility of analog simulation is the ease with which the operator can control the run, starting and stopping it at will. PACTOLUS uses the sense switches of the 1620 to achieve the same measure of control provided by the usual “Standby-Initial Condition-Operate” switch of the analog computer. The operator may terminate a run at any time by momentarily turning on Sense Switch #4. He then sets the sense switches in accordance with the operational option he desires and presses the Start switch to continue.

The sense switch settings for the various options are as follows:

Sense Switch #1 *on* permits the operator to modify both the configuration and initial condition/parameter specifications;

Sense Switch #2 *on* permits the operator to modify the initial condition/parameter specifications;

Sense Switch #3 *on* permits the operator to modify the timing and output specifications;

Sense Switch #4 *on* causes the plotter to move the paper beyond the present plot and prepare a new plot frame.

Sense Switch #3 may be used independently or in conjunction with Sense Switches #1 or #2. Sense Switch #4 is always used in conjunction with one or more of the other switches. If the computer is restarted with all switches off, the program presumes that an entirely new simulation is to be started and attempts to read the configuration specifications.

#### IV. SUMMARY

Our objective in developing PACTOLUS has been twofold: to make a critical evaluation of the various techniques employed in previous digital analog simulator programs and to demonstrate that a modus operandi comparable to that of analog computer users could be obtained for digital simulation. PACTOLUS embodies the conclusions of that evaluation. No attempt has been made herein to detail the reasons for accepting certain features while rejecting others. With the hope that this effort will be of value in the development of subsequent simulation programs, we merely wish to state that this is our considered opinion after serious study.

Whether PACTOLUS does indeed represent an innovation in operational flexibility will only be known after the program achieves much wider usage. It has been used for a number of

small applications,<sup>12</sup> but we readily admit a measure of bias. We do feel that PACTOLUS demonstrates that our objective can be obtained. To the user of the small scientific computer, the program offers a means of conveniently solving many of those problems for which an analog computer would otherwise have been required. The techniques employed in PACTOLUS are hopefully suggestive of the manner in which digital simulation might be provided at remote terminals serviced by a large digital computer. It is our thought that digital simulation is on the brink of significant expansion. The availability of appropriate terminals and visual display units will herald this event.

#### REFERENCES

1. SELFRIDGE, R. G.: "Coding a General-Purpose Digital Computer to Operate as a Differential Analyzer," Proc. 1955 Western Joint Computer Conference (IRE).
2. LESH, F.: "Methods of Simulating a Differential Analyzer on a Digital Computer," J. of the ACM, Volume 5, Number 3, 1958.
3. STEIN, M. L., ROSE, J., and PARKER, D. B.: "A Compiler with an Analog-Oriented Input Language," Proc. 1959 Western Joint Computer Conference.
4. HURLEY, J. R.: "DEPI 4 (Differential Equations Pseudo-Code Interpreter)—An Analog Computer Simulator for the IBM 704," internal memorandum, Allis Chalmers Mfg. Co., January 6, 1960.
5. HURLEY, J. R.: "Digital Simulation I: DYSAC, A Digitally Simulated Analog Computer," AIEE Summer General Meeting, Denver, Colorado, June 17-22, 1962.
6. STOVER, R. F., and KNUDTSON, H. A.: "H-800 PARTNER—Proof of Analog Results Through a Numerical Equivalent Routine," Doc. No. U-ED 15002, Minneapolis-Honeywell Regulator Co., Aero. Divn., April 30, 1962.
7. GASKILL, R. A., HARRIS, J. W., and MCKNIGHT, A. L.: "DAS—A Digital Analog Simulator," Proc. 1963 Spring Joint Computer Conference.

8. BYRNE, E. R.: "JANIS, A Program for a General-Purpose Digital Computer to Perform Analog Type Simulations," ACM Nat. Conf., Denver, Colorado, August 27, 1963.
9. HARNETT, R. T., SANSOM, F. J., and WARSHAWSKY, L. M.: "MIDAS Programming Guide," Tech. Doc Report SEG-TDR-64-1, Analog Computation Divn., Wright-Patterson Air Force Base, January 1964.
10. MCLEOD, J.: "Simulation is Wha-a-t?," SIMULATION, Vol. 1, No. 1, Fall 1963.
11. LINEBARGER, R. N.: "Digital Simulation Techniques for Direct Digital Control Studies," Proc. 19th Annual Conf. of ISA, New York, October 12, 1964.
12. COWLEY, PERCY E. A.: "An Active Filter for the Measurement of Process Dynamics," Proc. 19th Annual Conf. of ISA, New York, October 12, 1964.

# MIDAS – HOW IT WORKS AND HOW IT'S WORKED

*Harry E. Petersen, F. John Sansom, Robert T. Hartnett,  
and L. Milton Warshawsky  
Directorate of Computation  
Wright-Patterson AFB, Ohio*

## INTRODUCTION

The possibility of using a digital computer to obtain check solutions for the analog was recognized by many people at the dawn of our 15 year old history. Unfortunately several problems existed then, mainly at the digital end, which made this impracticable. Digital computers of that day were terribly slow, of small capacity and painfully primitive in their programming methods. It was usually the case when a digital check solution was sought for an incoming analog problem, that it was several months after the problem had been solved on the analog computer and the results turned over to the customer before the digital check solution made its appearance. The fact that the two solutions hardly ever agreed was another deterrent to the employment of this system.

As we all know, digital computers have made tremendous strides in speed, capacity and programmability. In the area of programming—and throughout this paper—we're talking of scientific problems expressible as differential equations; the main effort has been in the construction of languages such as Fortran, Algol, etc. to permit entering the problem in a quasi-mathematical form, with the machine taking over the job of converting these to the individual serial elemental steps. While the progress along this line has been truly awe-inspiring to an analog man (usually an engineer), the resulting language has become quite foreign to him so that if he wishes to avail himself of the

digital computer he must normally employ an interpreter in the form of a digital programmer (usually a mathematician). This means that he must describe his engineering problem in the required form, detail, and with sufficient technical insight to have the digital programmer develop a workable program on the first try. This doesn't happen very often and it is the consensus of opinion among computing facility managers that a major source of the difficulty lies in the fact that the engineer does not always realize the full mathematical implications of his problem. For example, in specifying that a displacement is limited, he might not state what happens to the velocity. This can lead to all sorts of errors as an analog programmer would know. It is, of course, possible for an analog programmer to learn to program a digital computer by studying Fortran. This has been attempted here at Wright-Patterson AF Base with little success, mainly because, unless used very often, the knowledge is lost so that each time a considerable relearning period is required. Some computing facilities have even embarked on cross-training programs so that each type of programmer knows the other's methods. While this has much to recommend it, it is often impracticable.

In March of 1963, Mr. Roger Gaskill of Martin-Orlando explained to us the operation of DAS (Digital Analog Simulator),<sup>1</sup> a block diagram type of digital program which he intended for use by control system engineers who

did not have ready access to an analog computer. We immediately recognized in this type of program the possibility of achieving our long-sought goal of a means to obtain digital check solutions to our analog problems by having the analog programmer program the digital computer himself! We found that our analog engineers became quite proficient in the use of DAS after about one hour's training and were obtaining digital solutions that checked those of the analog.

At this point several limitations of this entire method should be acknowledged. First, the idea that obtaining agreement between the digital and analog solutions is very worthwhile is based mainly on an intuitive approach. After all both solutions could be wrong since the same programming error could be made in both. Secondly, the validity of the mathematical model is not checked, merely the computed solution. Finally, it might be argued that the necessity of the analog man communicating the problem to his digital counterpart has the value of making him think clearly and organize his work well. This is lost if he programs the digital computer himself. In spite of these limitations we thought it wise to pursue this idea.

Although DAS triggered our activity in the field of analog-type digital programs, several others preceded it. A partial list of these and other such programs would include:

DEPI <sup>2</sup>	California Institute of Technology
DYSAC <sup>3</sup>	University of Wisconsin
DIDAS <sup>4</sup>	Lockheed-Georgia
PARTNER <sup>5</sup>	Honeywell Aeronautical Division
DYNASAR <sup>6</sup>	General Electric, Jet Engine Division

Almost all of these—with the possible exception of PARTNER (*Proof of Analog Results Through a Numerical Equivalent Routine*)—had as their prime purpose the avoidance of the analog computer. They merely wished to borrow the beautifully simple programming techniques of the electronic differential analyzer and apply them to the digital computer.

While DAS proved to be very useful to us, certain basic modifications were felt to be necessary to tailor it better to our needs. Principal among these modifications was a rather sophisticated integration routine to replace the simple fixed interval rectangular type of DAS. Other important changes were made but the change in the integration scheme and our wish to acknowledge our debt to DAS, led us to the choice of the name MIDAS, an acronym meaning *Modified Integration Digital Analog Simulator*. In this paper a brief description of the method of using MIDAS will be given, followed by a summary of our experience in using it in a large analog facility for about 18 months.

#### *How MIDAS Works*

To a large degree, programming in MIDAS closely resembles the methods used in DAS and, therefore, an analog computer. There are usually three steps we go through in obtaining a solution. First we prepare a block diagram very similar to an analog schematic indicating the operational elements required to solve the problem. Next we prepare a listing which is our means of directing the punching of the cards, one for each line. This listing indicates the source of the inputs to each element and defines the values of the required numerical data. After the cards have been punched and checked they are turned in to our IBM 7094 operations group where the information is placed on magnetic tape and this tape, along with the MIDAS subroutine, is used to solve our particular problem. The results are given to us on printed form according to a rather fixed format.

As an illustration of the steps involved in preparing a MIDAS program, let us set up the classical second-order linear differential equation for the mass, spring and damper system.

The equation is:

$$M\ddot{x} + B\dot{x} + Kx = 0 \quad (1)$$

with initial conditions:

$$\begin{aligned} x(0) &= A \\ \dot{x}(0) &= 0 \end{aligned}$$

The MIDAS block diagram for this equation is shown in Figure 1. The following points should be noted:

DESCRIPTION OF MIDAS ELEMENTS

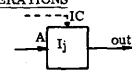
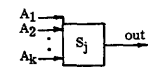
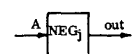
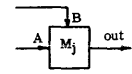
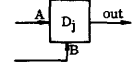
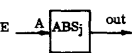

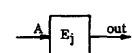
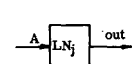
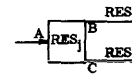

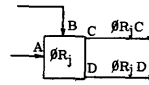
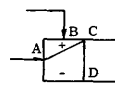
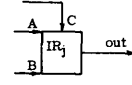
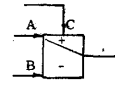
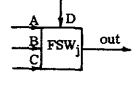
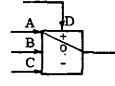
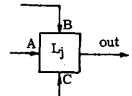
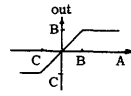
	SYMBOL & NAME	OUTPUT	REMARKS									
<b>1. MATHEMATICAL OPERATIONS</b>												
INTEGRATE		$Out = \int_0^t Adt + IC$	1. Only one input can be accepted. 2. The initial condition, IC, is transmitted via an IC card and its corresponding data card.									
SUM		$Out = \sum_{i=1}^K A_i$ $K \leq 6$										
NEGATIVE		$Out = -A$										
MULTIPLY		$Out = A \cdot B$										
DIVIDE		$Out = A/B$	The order of listing the inputs is very important. The numerator, A, must be listed first.									
ABSOLUTE VALUE		$Out =  A $										
SQUARE ROOT		$Out = +\sqrt{A}$	Defined only for $A \geq 0$									
EXPONENTIAL		$Out = e^A$										
NATURAL LOGARITHM		$Out = \ln A$	Defined only for $A > 0$									
RESOLVER (sin & cos)		$B \text{ out} = \sin A$ $C \text{ out} = \cos A$	1. Input angle, A, must be in radians. 2. Since there are two outputs, these must be specified as RES_j B or RES_j C depending on whether the sine or cosine is required.									
ARC TANGENT		$Out = \tan^{-1} A$	1. Output is an angle in radians. 2. Defined only for the 1st and 4th quadrants i.e., $-\pi/2 < \text{out} < \pi/2$									
<b>2. SWITCHING ELEMENTS</b>												
OUTPUT RELAY		<table border="1" data-bbox="771 1302 925 1386"> <tr> <td></td> <td><math>B \geq 0</math></td> <td><math>B &lt; 0</math></td> </tr> <tr> <td>C out</td> <td>A</td> <td>0</td> </tr> <tr> <td>D out</td> <td>0</td> <td>A</td> </tr> </table>		$B \geq 0$	$B < 0$	C out	A	0	D out	0	A	Equivalent to: 
	$B \geq 0$	$B < 0$										
C out	A	0										
D out	0	A										
INPUT RELAY (SPDT)		$out = \begin{cases} A & C \geq 0 \\ B & C < 0 \end{cases}$	Equivalent to: 									
FUNCTION SWITCH (SP3T)		$out = \begin{cases} A & D > 0 \\ B & D = 0 \\ C & D < 0 \end{cases}$	Equivalent to: 									
LIMITER		$out = \begin{cases} B & A > B \\ A & C \leq A \leq B \\ C & A < C \end{cases}$	Equivalent to:  $B > C$									

Table I. Description of MIDAS Elements.

TABLE I DESCRIPTION OF MIDAS ELEMENTS (cont'd)

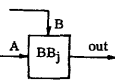
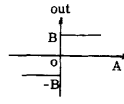
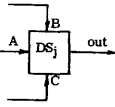
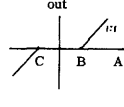


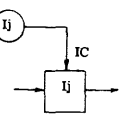
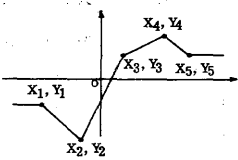

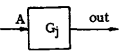
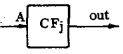
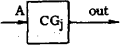
BANG-BANG		$\text{out} = \begin{cases} +B & A \geq 0 \\ -B & A < 0 \end{cases}$	Equivalent to:	
DEAD SPACE		$\text{out} = \begin{cases} A-B & A > B \\ 0 & C \leq A \leq B \\ A-C & A < C \end{cases}$	Equivalent to:	
<b>5. RUN TERMINATION</b>				
FINISH		None		<ol style="list-style-type: none"> <li>1. When <math>A \geq B</math>, computation is stopped.</li> <li>2. Every program must contain at least one FIN statement.</li> <li>3. Numbering of FIN statements is not req'd.</li> </ol>
<b>6. INSERTION OF NUMERICAL ITEMS</b>				
CONSTANT or PARAMETER		Per Data Card		<ol style="list-style-type: none"> <li>1. The name of a constant or parameter can be composed of at most six alphanumeric symbols excluding blanks and commas.</li> <li>2. The names must not be the same as that of a functional element used in the problem.</li> <li>3. The name will appear on a CØN or PAR card and its numerical value on a data card.</li> <li>4. Do not use these special names: IT, TR, MININT, ØPTIØN.</li> </ol>
INITIAL CONDITION		Ij(0)		<ol style="list-style-type: none"> <li>1. The name must be the same as the integrator with which it is associated.</li> <li>2. The name must appear on an IC (or PAR) card and its value on a data card.</li> <li>3. Only non-zero IC's need be specified.</li> <li>4. Such IC's must be specified for every run even though they do not change.</li> </ol>
<b>SPECIAL STATEMENTS</b>				
HEADER	HDR	None		<ol style="list-style-type: none"> <li>1. Contents of the HDR statements are printed at the beginning of each run.</li> <li>2. Normally used to name the variables recorded in each column.</li> <li>3. HDR cards should precede the RØ cards.</li> </ol>
READOUT	RØ	None		Specifies the sources of the variables to be recorded for each run.
END	END	None		Signifies the end of the MIDAS symbolic program. Numerical data follows.
<b>SPECIAL NAMES</b>				
INDEPENDENT VARIABLE	IT	Independent Variable		<ol style="list-style-type: none"> <li>1. Gives the current value of the independent variable in the appropriate units.</li> <li>2. Generated internally, this variable can be obtained by specifying its source as IT.</li> </ol>
TIME BETWEEN READOUTS	TR	None		<ol style="list-style-type: none"> <li>1. When listed on a CØN or PAR card, TR gives the increment of the independent variable, usually time, between successive readouts.</li> <li>2. If no value of TR is given, a standard value of 0.1 units will be used.</li> </ol>
MINIMUM INTERVAL OF INTEGRATION	MININT	None		<ol style="list-style-type: none"> <li>1. When listed on a CØN or PAR card, MININT specifies the smallest interval that the integration system is permitted to use.</li> <li>2. If no value of MININT is given, a value of ZERO is used.</li> </ol>
INTEGRATION OPTION	ØPTIØNj	None		When an integration method with an error criterion other than the standard one is desired, call for ØPTIØNj in columns 1-7 somewhere within the MIDAS program.

Table I



	SYMBOL & NAME	OUTPUT	REMARKS
3. <u>ARBITRARY FUNCTIONS</u>			
For all 4 types of function generators:			
			<ol style="list-style-type: none"> <li>Up to 50 sets of x-y coordinates can be used.</li> <li>Spacing of breakpoints is arbitrary.</li> <li>Slope of the function is zero above and below the set of specified points.</li> <li>Method of introducing data is in the text.</li> </ol>
FUNCTION		Out = f(A)	<ol style="list-style-type: none"> <li>Linear interpolation.</li> <li>Data needed for <u>every</u> run.</li> </ol>
CONSTANT FUNCTION		Out = f(A)	<ol style="list-style-type: none"> <li>Linear interpolation.</li> <li>Data needed for first run only.</li> </ol>
CURVE FOLLOWER		Out = f(A)	<ol style="list-style-type: none"> <li>Quadratic interpolation.</li> <li>Data needed for <u>every</u> run.</li> </ol>
CONSTANT CURVE FOLLOWER		Out = f(A)	<ol style="list-style-type: none"> <li>Quadratic interpolation.</li> <li>Data needed for first run only.</li> </ol>

4. ITERATIVE ELEMENT


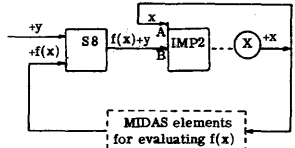
IMPLICIT FUNCTION		No direct output	<ol style="list-style-type: none"> <li>The inputs must be listed in the order A, B.</li> <li>If <math> A - B  &gt; 5 \times 10^{-6}  A </math>, iteration occurs by transferring the value of B into A (B → A), recomputing a new value of B, transferring it into A, etc., until the error criterion is satisfied.</li> <li>Criterion for convergence is <math> \frac{\partial B}{\partial A}  &lt; 1</math>.</li> <li>An initial estimate of A must be given via a CON or PAR card and its associated data card.</li> <li>When A is needed elsewhere in the problem, it can be taken from the CON or PAR source. (See suggested method of usage at the left.)</li> <li>For multiple runs, A must be named on a PAR card.</li> </ol>
Example:	$x = f(x) + y$		
			

Table I

- (1) S1 is a *summer* which adds the quantities  $-Bx$  and  $-Kx$  to yield  $M\ddot{x}$  in accordance with equation 1.
- (2) D1 is a *divider* whose dividend is  $M\dot{x}$  (the output of S1) and whose divisor is a constant called M. Its output is then  $+\dot{x}$ .
- (3) I1 is an *integrator* whose input is  $+\dot{x}$  and output is  $+x$ . Unlike the case of analog integrators and summers, there is no change of sign in the equivalent MIDAS elements. Since no initial value is specified the output of I1 will start at zero.
- (4) I2 is another integrator whose input is  $+x$  and whose output is  $+x$ . The initial value of  $x$  is indicated by the dashed line extra input to I2.
- (5) M1 and M2 are *multipliers* which multiply  $+\dot{x}$  by  $-B$  and  $+x$  by  $-K$  to provide the required inputs to S1. Unlike on analog computers the same type of multiplier is used whether two constants, a variable and a constant, or two variables are being multiplied.
- (6) FIN is a *finish element*. This element will stop computation when its A input (in this case  $t$ ) equals or exceeds its B input (in this case the quantity named STOP). Computation can be caused to halt by any of several conditions. A FIN box is required for each termination criterion.
- (7) When numerical data corresponding to M, B, K, STOP and  $x(0)$  are furnished, the problem is completely specified. *No scaling* will be required since numerical values ranging between  $10^{-37}$  and  $10^{+37}$  can be handled.
- (4) One card for each MIDAS element in the block diagram, identifying it by type and number, and indicating the source of its inputs (for example S2 meaning summer number 2).
- (5) At least one FIN card specifying a condition for finishing a run.
- (6) One or more HDR cards and RO cards specifying the *header* names to be placed on top of the columns of output data to be read out at specified increments of the independent variable.
- (7) An END card indicating the end of the symbolic program and the start of numerical data.
- (8) One or more cards assigning numerical value to the constants, parameters and initial conditions named in (3) above.
- (9) Cards defining arbitrary functions, if any.

An example of a listing is shown in Figure 2 for the mass, spring, and damper system.

Note the use of a comment card by the programmer to identify the problem. Also of significance is the columnar location of various types of entries. For example, comment cards have their first letter in column 1. Operational elements, constant and parameter naming cards, HDR and RO cards all have their first character in column 7. Inputs to these elements are listed starting in column 15. Numerical data is listed starting in columns 1, 11, . . . 51.

It should be pointed out that in MIDAS, the programmer need not concern himself with the order in which he prepares the listing since a built-in sorting routine will automatically line up the program properly. This is another important difference from MIDAS' predecessor, DAS.

The particular listing shown will result in three runs being made, each starting with  $x(0) = 20$  and terminating when  $t = 5$ . The mass M will have a value of 10 for each case since M was named on a *constant* card. The three runs will have the following values of  $-B$  and  $-K$ , each of which was specified as a *parameter*.

Next the listing is prepared. It will contain the following information:

- (1) One card identifying the problem.
- (2) A few cards calling out the MIDAS program.
- (3) Cards giving names to the constants and parameters of the problem, including integrators with non-zero initial values (up to 6 per card).

PROGRAMMER SANSOM PHONE 33281 DATE 20 JULY 64 PAGE      OF      PAGES

PROGRAM MASS, SPRING, DAMPER SYSTEM

1	FUNCTION	INPUTS	74
	*ID, NAME/SANSOM, PROB/64-224, TIME/5, LINES/500		
#	XEQ		
	CALL, MIDAS		
	END		
#	DATA		
	SOLUTION, OF, MASS, SPRING, DAMPER, SYSTEM		
	CON	M, STOP	
	PAR	-B, -K	
	IC	I2	
	S1	M1, M2	
	D1	S1, M	
	I1	D1	
	I2	I1	
	M1	-B, I1	
	M2	I2, -K	
	FIN	IT, STOP	
	HDR	TIME, ACC., VEL., DISP.	
	HDR		
	R0	IT, D1, I1, I2	
	END		

ASD FORM SEP 63 0-4905

MIDAS DATA FORM

PROGRAMMER SANSOM PHONE 33281 DATE 20 JULY 64 PAGE      OF      PAGES

PROGRAM MASS, SPRING, DAMPER SYSTEM

1	11	21	31	41	51	60
10.	5.					
-2.5	-8.6					
20.						
-3.2	-8.6					
20.						
-2.5	-15.					
20.						

Figure 2. Original Listing.

- Run 1 -B = -2.5 , -K = -8.6
- Run 2 -B = -3.2 , -K = -8.6
- Run 3 -B = -2.5 , -K = -15.0

Finally a portion of the printed output of the IBM 7094 is shown in Figure 3. Several points are worthy of note.

- (1) The printout of the problem listing including the data for Case 1. Actually some machine mapping and storage information precedes this but has been omitted for clarity.
- (2) The format of the output. Note the headers and the spacing of the four columns of output. Provision exists for six columns but only four components were specified to be read out in this simple problem.
- (3) The MAXIMA-MINIMA table. This feature, unique to MIDAS, provides the

analog programmer with all the information needed to scale his analog schematic, both in amplitude and time. It shows the maximum and minimum values achieved by every component during the course of a run, whether these values occurred at read out intervals or not.

- (4) The printout of the parameter and IC data for Cases 2 and 3, followed by their output.
- (5) The job accounting summary. The three cases took a total of 44 seconds.

This completes the description of this problem. Although considerable detail has been presented, in retrospect it can be seen that the main idea was simple. An analog-type block diagram was drawn and a listing prepared describing its interconnections. Information providing numerical data was also furnished.



This was then converted to punched cards, turned in to the 7094 operators who subsequently supplied the printed output.

This example utilized only a few of the available MIDAS elements. The entire complement of them is shown in Table I, along with some description of their use. The A,B,C lettering system has significance where an element has more than one input. In this case the order of the sources of the inputs as given in the listing (starting in column 15) should be A,B,C, etc.

#### *Integration System in MIDAS*

The integrations in MIDAS are performed by a variable-step, fifth-order, predict-correct integration routine.<sup>7</sup> This variable-step feature represents the basic departure of MIDAS from its predecessor, DAS. It relieves the programmer from the chore of having to specify a fixed increment of the independent variable, an increment which must be small enough to handle the highest frequency phenomena in a problem but not so small as to cause inordinately long solution times. The step size in the MIDAS integration routine adjusts itself to meet a certain error criterion, a factor which allows it to take large steps for those portions of the solution "when not much is happening" and small steps for those portions when one or more variables are changing at rapid rates.

The net result is that time-scaling, as the analog programmer knows it, is eliminated in MIDAS. However, he must still face the time scaling problem when he prepares his analog schematic, especially when certain variables in the problem drive narrow bandwidth analog components such as servo-multipliers, X-Y plotters, etc. MIDAS helps him here, though, because he can observe the maximum values of the derivatives of these variables from the MAXIMA-MINIMA list and he can then make any necessary time base changes. Fortunately, for every variable appearing at the output of a MIDAS integrator, its derivative must appear at the output of the element feeding the integrator since integrators can accept only a single input.

#### *Miscellaneous Information on MIDAS*

The MIDAS program has limitations on the number and characteristics of certain compo-

nents. This information may be of value when a very large problem is to be solved on MIDAS.

<i>Item</i>	<i>Maximum Number</i>
1. Operational Elements	750
2. Symbols (operational elements, constants, and header names)	1000
3. Integrators	100
4. Function Generators	40
5. Points for each function generator	50
6. Inputs for each summer	6

#### *Summary of How MIDAS Works*

This has been of necessity a brief review of the method of using MIDAS. A much more complete description including several fully worked out examples is given in Technical Documentary Report No. SEG-TDR-64-1, "MIDAS PROGRAMMING GUIDE," dated January 1964.<sup>8</sup> Information on obtaining the MIDAS program can be obtained by contacting the authors.

#### *How MIDAS Has Worked*

In the following paragraphs, a brief summary of the experiences of the Wright-Patterson AF Base Computation Facility in the use of MIDAS will be given. Actually, at this writing, approximately 100 computing facilities throughout the U.S. are using MIDAS; thus only a small segment of the total experience can be reported on.

The Analog Computation facility at Wright-Patterson has used MIDAS in almost every problem submitted for solution on its large analog computer. Generally the MIDAS solution is attempted prior to the analog in order to achieve the maximum benefits as regards scaling. Another side benefit of MIDAS has been the calculation of Problem Check values. It has been found in many cases that the extra time involved in programming a MIDAS check is saved in checkout time on the analog computer. The increased confidence in the validity of the solutions when a check between MIDAS and the analog solution is obtained is the most important benefit of the program. Another side benefit is the broadened horizons achieved

by the analog programmers in giving them the ability to program the digital computer. This should be very valuable when hybrid computers come into their own. Until now the problem of finding people with the required capability in both areas has proven to be the greatest deterrent to the growth of these new devices.

While everything mentioned above is on the positive side, there are quite a few aspects of MIDAS that are annoying and time consuming. One thing that the analog programmer learns is that the digital computer brooks no mistakes. If a "zero" is punched when the letter "O" is required, the problem comes back—generally the following day—with a diagnostic telling him of an undefined symbol. A little thing like a missing decimal point in numerical data will cause a day to be lost. Another thing that an analog programmer encounters is that when an error exists in a MIDAS program (however not of the type to prevent its running), the solutions look just as good (as many significant figures of output) as they would if the solution were perfect. On the analog computer errors of the same type would cause overload lights to flash, etc.

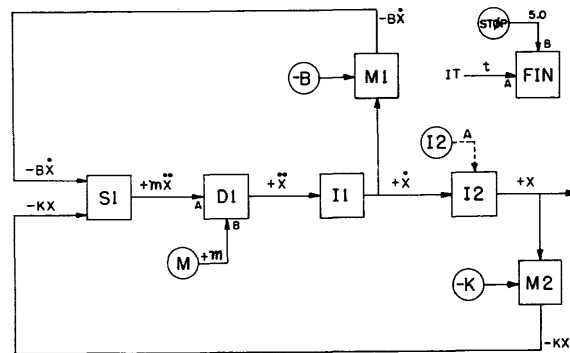
The very sophisticated integration routine of MIDAS introduces problems at times. For example, discontinuities in derivatives some-

times make it impossible for the error criterion to be met even though the increment of the independent variable is halved many times. This will cause a solution to "hang up." Provision has been made to overcome this problem by the use of MININT (see Table I) but it usually requires one or more unsatisfactory runs before the programmer is made aware of the difficulty.

One rather interesting discovery was the fact that an operation that was very easy to perform on an analog computer was very bothersome on the digital. Specifically, a rather large missile simulation was performed first on the analog computer and later using MIDAS. Quite a few first order lags were present in the mathematical model in the form of

$\frac{1}{\tau S + 1}$ . On the analog computer this offers no problem. For small values of  $\tau$  one way to handle this to parallel the feedback resistor of a summer with a capacitor of  $\tau$  microfarads. In fact, far from *creating* a difficulty, it generally is beneficial to the analog simulation by reducing some of the high frequency "noise." Using MIDAS, small values of  $\tau$  can cause considerable increase in solution time. For example, in this particular problem, when such transfer functions with  $\tau$  of .001 secs. were

Figure 1. Block Diagram.



included, the solution time was extrapolated to be 5½ hours for 26 seconds of problem time. This was reduced to 12½ minutes for the same length of problem time, simply by neglecting these small delays, and the effect on the results was insignificant. Incidentally, the analog solution was in "real time," i.e., 26 seconds.

The subject of solution time is rather important to a digital programmer. We have attempted to gather data on the relative speed of a MIDAS run compared with a Fortran program produced by a skilled programmer. With the conditions of the test equalized, the solution time of the Fortran program was approximately half as long; however, the programming time for MIDAS was much less. The question of solution time is not very important for the typical problem handled with MIDAS because usually we are interested in one or two runs, so whether they take 3 minutes or 5 minutes each is of academic interest only.

There have been a few problems handled by MIDAS alone without recourse to the analog computer. In these cases, program efficiency was of considerable importance since many runs were required. Here, in the present stage of the development of MIDAS, a specially tailored digital program should receive serious consideration.

At this point the question should be considered of whether MIDAS or a similar digital computer program will take over the role of the analog computer in the areas where the latter shines. Since a MIDAS-type program has appropriated one of the best features of the analog computer, viz., simple block diagram programming and the speed and capacity of digital computers have developed so much, there is certainly reason to consider this question.

While anyone would be foolhardy to give an answer to hold for all time, it is our opinion that MIDAS, rather than threatening the existence of analog computers, has reinforced their position by increasing confidence in their output. There are quite a few advantages to the use of an analog computer which MIDAS doesn't touch. Among these are:

- (1) The intimate relationship between the engineer and his problem which enables

him to design his system by observing graphical outputs and changing parameters as required.

- (2) The ability to tie in physical hardware to the mathematical simulation.
- (3) The ability to use portions of the computer in direct relationship to the size of the problem.
- (4) The fact that certain mathematical operations are performed better, e.g., integration.
- (5) The very fact that it is a distinctly different technique of solution, thus making possible a checking means.

While some progress has been and is being achieved in items 1, 2, 3 and 4, item 5 will always remain.

#### *Future of MIDAS*

Although MIDAS has proven to be very effective in accomplishing its purpose, certain improvements could be made without materially changing its simple programming rules. Among such improvements would be the following:

- (1) Increased efficiency, i.e., shorter solution times without losing programming simplicity.
- (2) Additional flexibility in naming outputs.
- (3) Permit the use of fixed point literals in the body of the program.
- (4) A greatly expanded operation list that would include logical operations such as AND, OR, NOT, etc. and others equivalent to the elements found in a hybrid computer.

A new program is being developed at Wright-Patterson AF Base which already includes the improvements listed above. In addition, it is anticipated that the following features will be included:

- (1) Ability to add new functions external to the basic program.
- (2) Additional controls that would
  - (a) Allow the results of one run to dictate automatically the conditions for the next.

- (b) Permit more "hands on" control of operation of the program as advocated by Mr. R. Brennan in his PACTOLUS<sup>9</sup> program.

It is further hoped that an investigation of various integration routines will result in an integration system that will automatically account for discontinuities and thus prevent the solution from "hanging up."

The new program, MIMIC, is completely different from MIDAS in concept but it retains the programming ease of MIDAS. It will be written as a system to operate under IBOB control on an IBM 7090/7094 computer.

It is an assembler type program that generates machine language code equivalent to the original problem. The instruction format is very similar to MIDAS but has been designed to appeal to both analog and digital programmers. If and when this occurs and both analog and digital programmers employ MIMIC regularly, a very significant first step in breaking down the communications barrier between the two will have been taken since they will, for the first time, be speaking the same language. Furthermore, just as MIDAS has made the digital computer accessible to the analog man, this new program might serve to educate the digital programmer in analog methods. The day of the omniscient, triple-threat programmer might be on the way!

#### REFERENCES

1. GASKILL, R. A., HARRIS, J. W., and MCKNIGHT, A. L., "DAS—A Digital Analog Simulator," AFIPS Proceedings, 1963 Spring Joint Computer Conference.
2. LESH, F. H., and CURL, F. G., "DEPI: An Interpretive Digital-Computer Routine Simulating Differential-Analyzer Operations," Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, March 22, 1957.  
(Note: DEPI is an acronym for *Differential Equations Pseudocode Interpreter*.)
3. HURLEY, J. R., and SKILES, J. J., "DYSAC: A Digitally Simulated Analog Computer," AFIPS Proceedings, 1963 Spring Joint Computer Conference.
4. SLAYTON, G. R., "DIDAS: A Digital Differential Analyzer Simulator," Twelfth National Meeting of the Association for Computing Machinery, June 1958.
5. KNUDTSON, H. A., and STOVER, R. F., "PARTNER—Proof of Analog Results Through a Numerical Equivalent Routine," Aeronautical Division, Minneapolis-Honeywell Regulator Co., MH Aero Document U-ED 15001, August 22, 1961.
6. MARVIN, I. E., and DURAND, H. P., "Jet Engine Control Representation Study," Jet Engine Division, General Electric Company, Cincinnati, Ohio, Air Force Technical Documentary Report ASD-TDR-63-650, July 1963.  
(Note: DYNASAR is an acronym for *Dynamic Systems Analyzer*).
7. MILNE, W. E., and REYNOLDS, R. R., *Journal of the ACM*, January 1962.
8. HARNETT, R. T., SANSOM, F. J., and WARSHAWSKY, L. M., "MIDAS Programming Guide," Air Force Technical Documentary Report SEG-TDR-64-1, January 1964. DDC (formerly ASTIA) Report No. AD-430892. Also available from Office of Technical Services, U.S. Dept. of Commerce.
9. BRENNAN, R. D., and SANO, H., "PACTOLUS—A Digital Analog Simulator Program for the IBM 1620," IBM San Jose Research Laboratory, San Jose, California, IBM Research Report RJ-297, May 6, 1964.



# THE RAND TABLET: A MAN-MACHINE GRAPHICAL COMMUNICATION DEVICE\*

*M. R. Davis and T. O. Ellis  
The RAND Corporation  
Santa Monica, California*

Present-day user-computer interface mechanisms provide far from optimum communication, considerably reducing the probability that full advantage is being taken of the capabilities of either the machine or of the user. A number of separate research projects are underway, aimed at investigating ways of improving the languages by which man communicates with the computer, and at developing more useful and more versatile communication channels. Several of these projects are concerned with the design of "two-dimensional" or "graphical" man-computer links.

Early in the development of man-machine studies at RAND, it was felt that exploration of man's existent dexterity with a free, pen-like instrument on a horizontal surface, like a pad of paper, would be fruitful. The concept of generating hand-directed, two-dimensional information on a surface not coincident with the display device (versus a "light pen") is not new and has been examined by others in the field. It is felt, however, that the stylus-tablet device developed at RAND (see Fig. 1) is a highly practical instrument, allowing further investigation of new freedoms of expression in direct communications with computers.

The RAND tablet device generates 10-bit x and 10-bit y stylus position information. It

is connected to an input channel of a general-purpose computer and also to an oscilloscope display. The display control multiplexes the stylus position information with computer-generated information in such a way that the oscilloscope display contains a composite of the current pen position (represented as a dot) and the computer output. In addition, the computer may regenerate meaningful track history on the CRT, so that while the user is writing, it appears that the pen has "ink." The displayed "ink" is visualized from the oscilloscope display while hand-directing the stylus position on the tablet, as in Fig. 1. Users normally adjust within a few minutes to the conceptual superposition of the displayed ink and the actual off-screen pen movement. There is no apparent loss of ease or speed in writing, printing, constructing arbitrary figures, or even in penning one's signature.

To maintain the "naturalness" of the pen device, a pressure-sensitive switch in the tip of the stylus indicates "stroke" or intended input information to the computer. This switch is actuated by approximately the same pressure normally used in writing with a pencil, so that strokes within described symbols are defined in a natural manner.

---

\* This research was supported by the Advanced Research Projects Agency under contract No. SD-79. Any views or conclusions should not be interpreted as representing the official opinion or policy of ARPA or of the RAND Corporation.

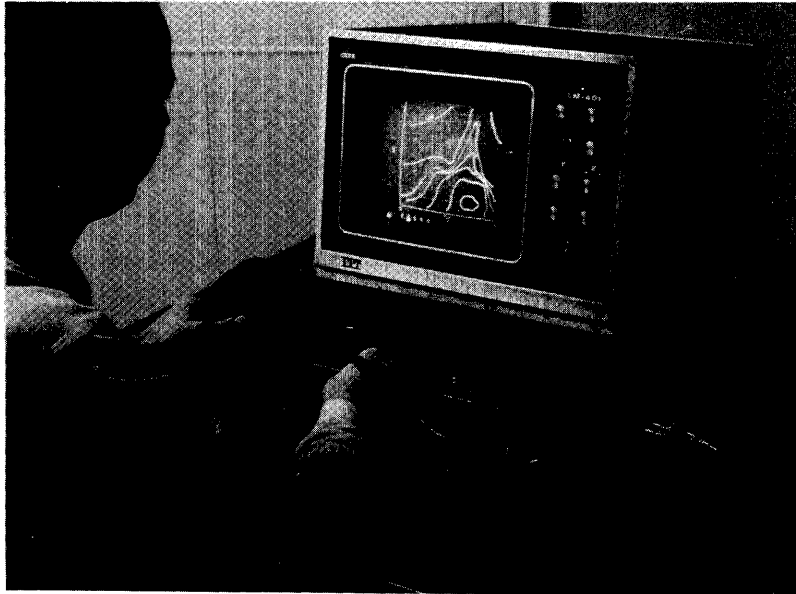


Figure 1. Complete System in Operation.

In addition to the many advantages of a "live pad of paper" for control and interpretive purposes, the user soon finds it very convenient to have no part of the "working" surface (the CRT) covered by the physical pen or the hand.

The gross functioning of the RAND tablet system is best illustrated through a general description of the events that occur during a

major cycle (220  $\mu\text{sec}$ ; see timing diagram, Fig. 2). Figure 3 is the system block diagram with the information flow paths indicated by the heavier lines. The clock sequencer furnishes a time sequence of 20 pulses to the blocking oscillators. During each of the 20 timing periods, a blocking oscillator gives a coincident positive and negative pulse on two lines attached to the tablet.

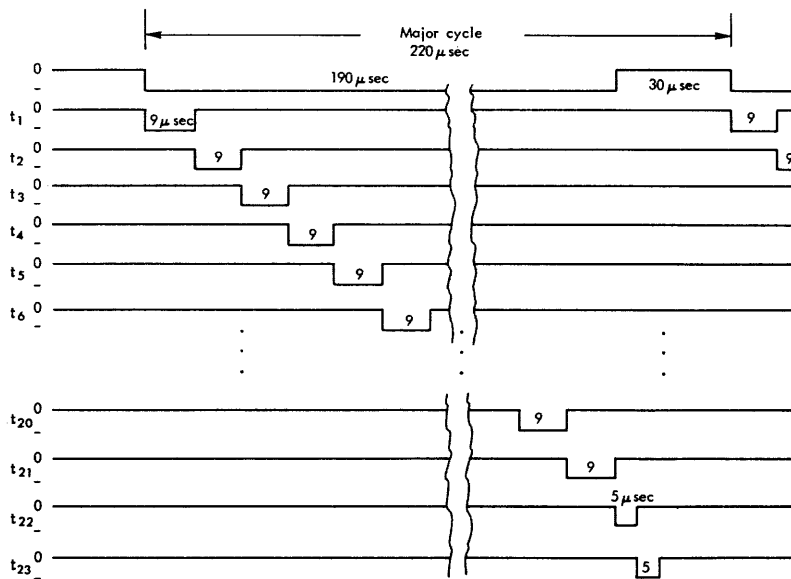


Figure 2. Timing Waveforms ( $\mu\text{sec}$ ).

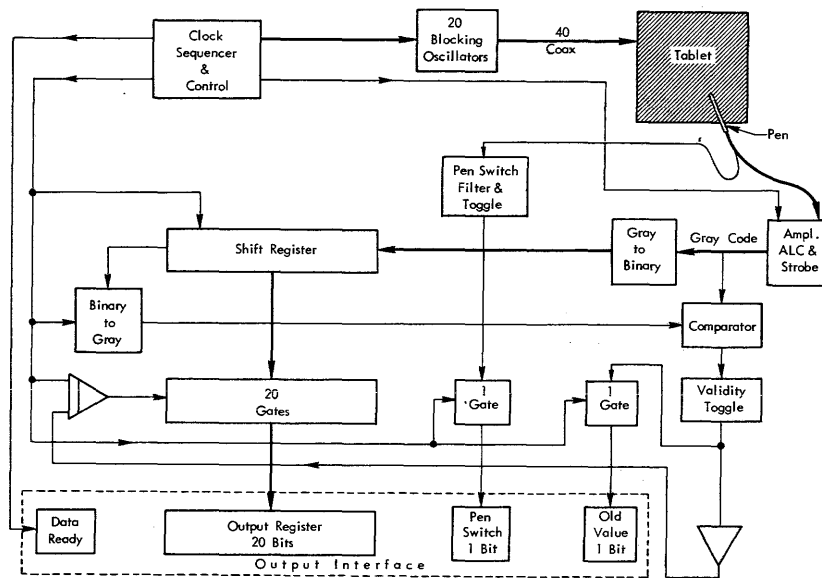


Figure 3. Graphic Input System Block Diagram.

The pulses are encoded by the tablet as serial (x,y) Gray-code position information which is sensed by the high-input-impedance, pen-like stylus from the epoxy-coated tablet surface. The pen is of roughly the same size, weight, and appearance as a normal fountain pen. The pen information is strobed, converted from Gray to binary code, assembled in a shift register, and gated in parallel to an interface register.

The printed-circuit, all digital tablet, complete with printed-circuit encoding, is a relatively new concept made possible economically by advances in the art of fine-line photoetching. The tablet is the hub of the graphic input system, and its physical construction and the equivalent circuit of the tablet itself will be considered before proceeding to the system detail.

The basic building material for the tablet is 0.5-mil-thick Mylar sheet clad on both sides with  $\frac{1}{2}$ -ounce copper (approximately 0.6 mils thick). Both sides of the copper-clad Mylar sheets are coated with photo resist, exposed to artwork patterns, and etched using standard fine-line etching techniques. The result is a printed circuit on each side of the Mylar, each side in proper registration with the other. (Accurate registration is important only in the encoder sections, as will be seen later.) Figure 4 is a photo of the printed circuit before it has

been packaged. The double-sided, printed screen is cemented to a smooth, rigid substrate and sprayed with a thin coat of epoxy to provide a good wear surface and to prevent electrical contact between the stylus and the printed circuit. The writing area on the tablet is  $10.24 \times 10.24$  in. with resolution of 100 lines per inch. The entire tablet is mounted in a metal case with only the writing area exposed, as can be seen in Fig. 1.

Although it would be very difficult to fully illustrate a  $1024 \times 1024$ -line system, it does seem necessary, for clarity, to present all the details of the system. Thus, an  $8 \times 8$ -line system will be used for the system description and

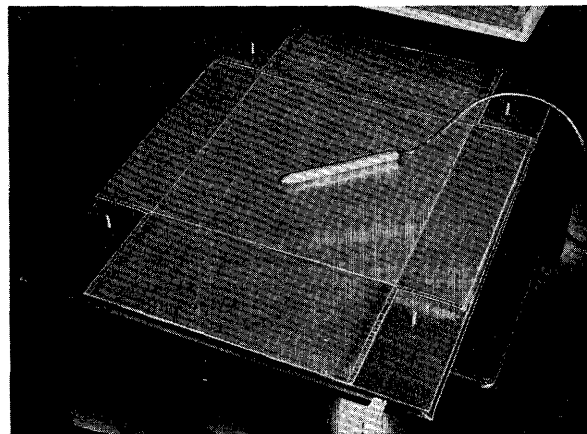


Figure 4. Unmounted Printed Circuit.

expansion of the concept to larger systems will be left to the reader.

Figure 5 shows the detailed, printed circuit on each side of the 0.5-mil Mylar for an  $8 \times 8$ -line system. The top circuit contains the x position lines and the two y encoder sections, while the bottom circuit has the y position lines and the two x encoder sections. It should be noted that the position lines are connected at the ends to wide, code-coupling buses. These buses are made as wide as possible in order to obtain the maximum area, since the encoding scheme depends on capacitive coupling from the encoder sections through the Mylar to these wide buses. It should be further noted that the position lines are alternately connected to wide buses on opposite ends. This gives symmetry to the tablet and minimizes the effect of registration errors.

With reference to Fig. 5, at time  $t_1$  encoder pads  $p_{1+}$  are pulsed with a positive pulse and pads  $p_{1-}$  are pulsed with a negative pulse. Pads  $p_{1+}$  are capacitively coupled through the Mylar to y position lines  $y_5, y_6, y_7,$  and  $y_8$ , thus coupling a positive pulse to these lines. Pads  $p_{1-}$  are capacitively coupled to y position lines  $y_1, y_2, y_3,$  and  $y_4$ , putting a negative pulse on these lines. At time  $t_2$ , encoder pads  $p_{2+}$  and  $p_{2-}$  are pulsed plus and minus, respectively, putting positive pulses on y position lines  $y_3, y_4, y_5,$  and  $y_6$ , and negative pulses on y position

lines  $y_1, y_2, y_7,$  and  $y_8$ . At the end of time  $t_3$ , each y position line has been energized with a unique serial sequence of pulses. If positive pulses are considered as ones and negative pulses are zeroes, the Gray-pulse code appearing on the y position wires is as follows:

$y_1$	000
$y_2$	001
$y_3$	011
$y_4$	010
$y_5$	110
$y_6$	111
$y_7$	101
$y_8$	100

The x encoder pads are now sequentially pulsed at times  $t_4, t_5,$  and  $t_6$ , giving unique definitions to each x position line.

If a pen-like stylus with high input impedance is placed anywhere on the tablet, it will pick up a time sequence of six pulses, indicating the  $(x,y)$  position of the stylus. It should be pointed out again that the stylus is electrostatically coupled to the  $(x,y)$  position lines through the thin, epoxy wearcoat.

If the stylus is placed on the tablet surface at a point  $(x_1, y_5)$ , the pulse stream appearing at the pen tip would be as indicated in Fig. 6. This detected pulse pattern will repeat itself every major cycle as long as the stylus is held in this position. If the stylus is moved, a differ-

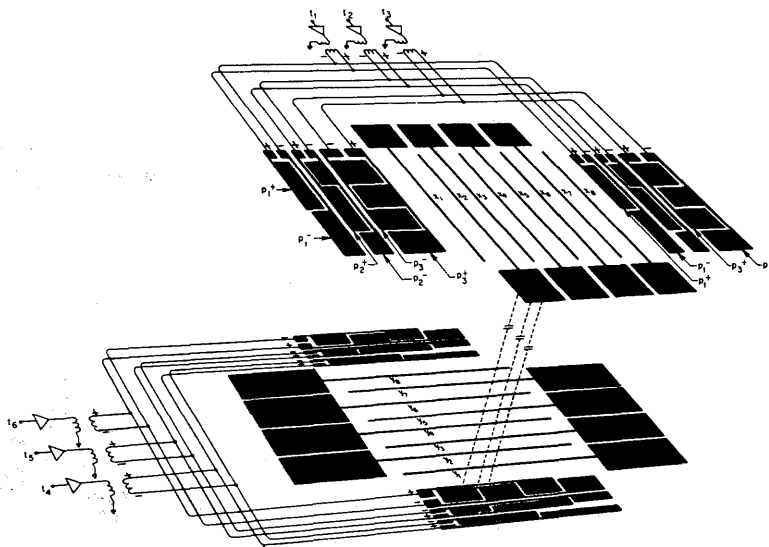


Figure 5. Double-sided Printed-circuit Layout for  $8 \times 8$  System.

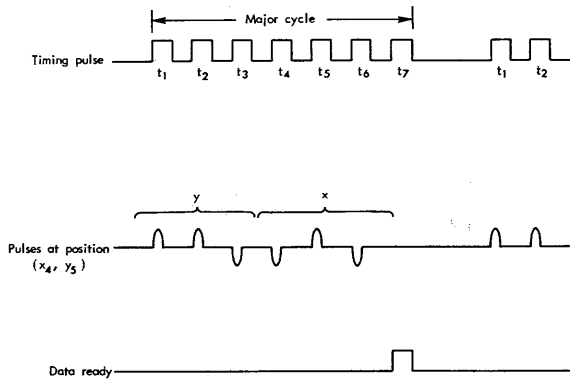


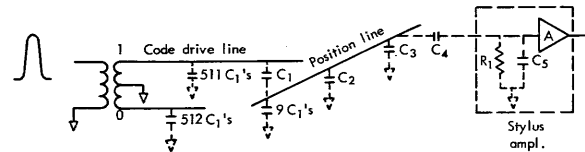
Figure 6. Timing Diagram and Pen Signals for the Example  $8 \times 8$  System.

ent pulse pattern is sensed, indicating a new  $(x,y)$  position.

Since there are 1024  $x$  position lines and 1024  $y$  position lines, 20 bits are required to define an  $(x,y)$  position. The actual timing used in the RAND system was shown in Fig. 2. Timing pulses  $t_{21}$ ,  $t_{22}$ , and  $t_{23}$  are additional pulses used for bookkeeping and data manipulation at the end of each major cycle.

The position lines on the full-size tablet are 3 mils wide with a 7-mil separation. The code-coupling pads are 16 to 17 mils wide with a 3- to 4-mil separation. Figure 4 shows that the encoding pads which couple to the lower set of position lines ( $y$  position lines) are enlarged. This greater coupling area increases the signal on the lower lines to compensate for the loss caused by the shielding effect on the upper lines (since they lie between the lower lines and the stylus pick-up). The encoding pad for the two least-significant bits in both  $x$  and  $y$  was also enlarged to offset the effect of neighboring-line cancellations. With these compensations, all pulses received at the stylus tip are of approximately the same amplitude.

Figure 7 is an illustration of the approximate equivalent circuit of the encoder-tablet-stylus system, along with typical system parameter values. It is clear that the values of  $C_1$  vary with encoder-pad size, and the value  $C_4$  varies according to whether top or bottom lines are being considered. The value of  $C_4$  is also dependent on the stylus-tip geometry and wear-coat thickness of the tablet. The signals arriving at the input to the stylus amplifier are ap-



- $C_1$  = Encoder pad coupling capacity  $\sim 5$  pf
- $C_2$  = Capacity to adjacent parallel wires in tablet  $\sim 10$  pf
- $C_3$  = Capacity to crossing lines in screen  $\sim 100$  pf
- $C_4$  = Stylus-to-tablet coupling capacity  $\sim .5$  pf
- $C_5$  = Stylus input shunt capacity  $\sim 5$  pf
- $R$  = Stylus input resistance  $\sim 200$  K $\Omega$

Figure 7. Equivalent Circuit of Encoder-Tablet-Stylus Coupling and Attenuating Elements.

proximately  $1/300$  of the drive-line signals. The character of the signals at the stylus input is greatly dependent on the drive-pulse rise time.

Figure 8 is an oscilloscope pattern of the amplified signals at the stylus output.† These signals are amplified again and strobed into a Gray-code toggle. An  $x$  bit at  $t_8$  and a  $y$  bit at  $t_{17}$  are smaller than the rest. This indicates that the stylus tip is somewhere between lines and these are the bits that are changing.

Since the final stages of the amplification and the strobing circuit are dc-coupled, the system is vulnerable to shift in the dc signal level. For this reason, an automatic level control (ALC) circuit has been provided to insure maximum recognizability of signals. During the first 180  $\mu$ sec of a major cycle, the stylus is picking up bits from the tablet. During the last 40  $\mu$ sec, the tablet is quiet—i.e., the stylus is at its quiescent level. During this 40- $\mu$ sec interval, the quiescent level of the pen is strobed into the ALC toggle. If the quiescent level is recognized as a zero, the ALC condenser changes slowly into the proper direction to change the recognition (via a bias circuit) to a one, and vice versa. For a perfectly balanced system, the ALC toggle would alternate between 1 and 0 with each major cycle.

A Gray code was selected so that only one bit would change value with each wire position, giving a complete and unambiguous deter-

† It will be noted in the oscilloscope pattern of Fig. 8 that the pulsing sequence is  $x$  first and  $y$  last. This is mentioned only because it is the opposite order of that shown in the  $8 \times 8$ -line example system discussed above; otherwise, it is unimportant.

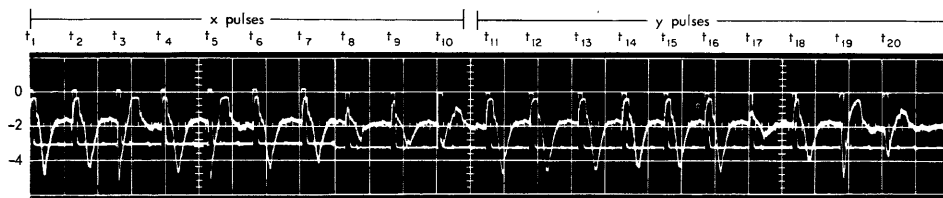


Figure 8. Oscilloscope of Pen Signal and Strobe.

mination of the stylus position. Furthermore, a reflected Gray code facilitates serial conversion to binary. The conversion logic for an  $N$ -bit number, when  $N$  is the most significant bit, is:

$$\text{Binary}_N = \text{Gray}_N$$

$$B_j = (\overline{B_{j+1}} \wedge G_j) \vee (B_{j+1} \wedge \overline{G_j}) \quad . j < N$$

Time-wise, the bits are received from the stylus in the order  $N, \dots, j+1, j, \dots, 1$ . When all 20 bits have been assembled in the shift register, they are gated to the output register.

As a new  $(x,y)$  value is being converted to binary and shifted into one end of the shift register, the old binary value is being shifted out the other end. This old binary information is serially reconverted to Gray and compared to the new, incoming Gray value, one bit at a time. If the old Gray number and incoming Gray number differ in more than one bit in either  $x$  or  $y$ , a "validity" toggle is set to indicate an error. If the two Gray-code series differ in more than one bit, this indicates that the pen has moved more than one line during the 220- $\mu$ sec interval. As this is not probable during normal usage, it is assumed that an error has occurred. If a set of data are determined as not valid, the output register is left with its previous value, and an "old-value" toggle is flagged.

The binary-to-Gray conversion logic is:

$$G_N = B_N$$

$$G_j = (\overline{B_{j+1}} \wedge B_j) \vee (B_{j+1} \wedge \overline{B_j}) \quad . j < N$$

In practice, the validity check rarely detects errors while the pen is in contact with the tablet. The pen validity check is used to suppress the display of the pen position as the pen is lifted off the tablet.

The logic and clock systems are made up principally with state-of-the-art NOR circuits and univibrators. The blocking-oscillator circuit shown in Fig. 9 was designed to drive the encoder pads. This use of transformer coupling was found to be important since well-matched positive and negative pulses were required to obtain proper cancellation at the tablet surface. The stylus amplifier has a gain of approximately 30 db with an additional 30-db gain in the principal electronic package.

The total electronic system is assembled in a  $5'' \times 5'' \times 19''$  printed-circuit card cage and contains some 400 transistors and about 220 diodes; however, little attempt has been made to minimize the number of components. Also, the electronics could be shared with a number of tablets in a multiple-tablet system.

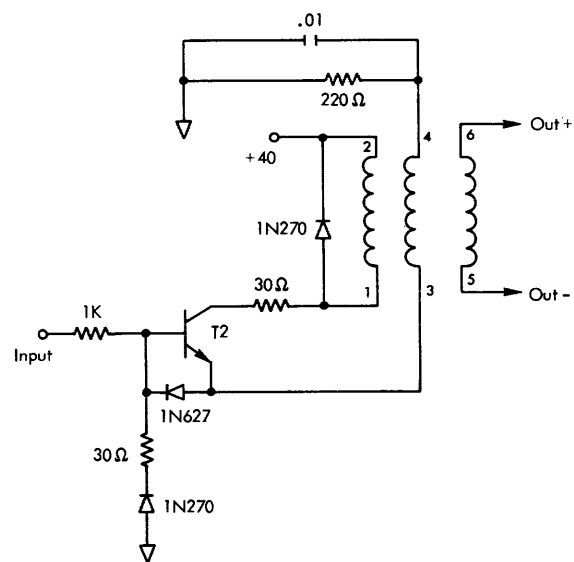


Figure 9. Blocking Oscillator.

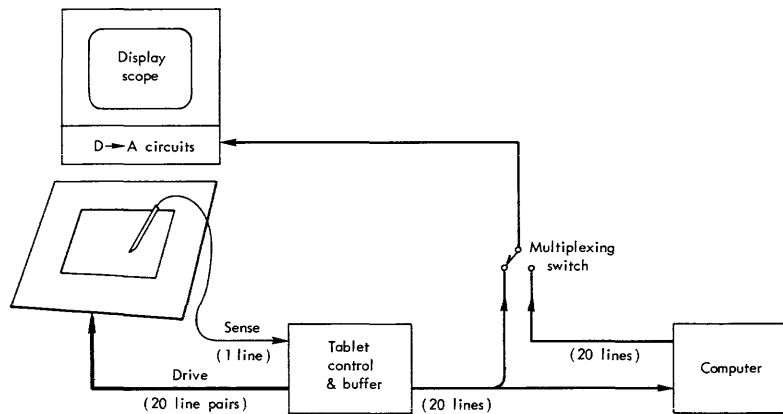


Figure 10. Information Paths in Graphic I/O System.

Figure 10 is a block diagram showing the graphic input-output system as used at RAND for the evaluation of hardware, human engineering studies, and investigation of programming implications. The computer used was the JOHNNIAC, a tube machine of the Princeton class.

Preliminary studies indicate that with a great amount of care in construction, a 200-line-per-inch tablet could be achieved. The resolution of this line density would not present a major problem; on the other hand, 100 lines per inch is adequate for all current intended applications.

It is certainly within the state of the art to decrease the major cycle time; however, in usage at RAND, the 4.5-kc rate has been adequate. When the stylus is swept rapidly across the surface of the tablet, it has been found that an average of two or three complete sets of position data are obtained for each line. Setting the multiplexing switch (Fig. 10) to display the stylus position on the scope every 10 msec has proved adequate, and since only 50  $\mu$ sec are required to display the point, 99.5 per cent of the display scope time is left for the computer.

The tablet currently is in regular use at RAND in studies toward the development of on-line graphical programming languages and on-line interaction with problem parameters. In addition to its use at RAND, several copies of the tablet have been supplied to other researchers in the field.

The tablet has been found to be particularly valuable in applications where excellent line-

arity and accuracy are important. Normal-thickness C.G.S. maps have been placed over the tablet to digitize contours by manual tracing with the pen.

Development of the stylus-tablet device has been carried to the point where, we feel, it represents a practical and economical tool for use in many applications. Additional application areas might be served by more development effort in directions such as providing for rear-projection of images onto the (translucent) tablet panel, provision for use of more than one sensing element, extension of the surface dimensions, etc.

#### BIBLIOGRAPHY

1. LICKLIDER, J. C. R., and CLARK, W. E., "On-Line Man-Computer Communication," Proc. 1962 SJCC, 113.
2. LOOMIS, H. H., Jr., *Graphic Manipulation Techniques Using the Lincoln TX-2 Computer* (Lincoln Laboratory, MIT, Cambridge, November 11, 1960), 516-0017 (U).
3. MARRILL, T., *et al.*, "Cyclops-1: A Second-Generation Recognition System," Proc. 1963 FJCC, 27.
4. STOTZ, R., "Man-Machine Console Facilities for Computer-Aided Design," Proc. 1963 SJCC, 323.
5. SUTHERLAND, E. E., "Sketchpad: A Man-Machine Graphical Communication System," Proc. 1963 SJCC, 329.
6. TEAGER, H. R., Private Communication, MIT.





# A SYSTEM FOR AUTOMATIC RECOGNITION OF HANDWRITTEN WORDS\*

*Paul Mermelstein† and Murray Eyden  
Department of Electrical Engineering and Research  
Laboratory of Electronics  
Massachusetts Institute of Technology  
Cambridge, Massachusetts*

## INTRODUCTION

The recognition of handwriting can be considered an important problem in the general pattern recognition area because the set of patterns, say individual words, possesses a degree of variability that far exceeds that of problems where relatively good solutions have been previously found. Whereas in the case of character recognition the number of pattern classes considered different usually does not exceed one hundred, the number of pattern classes with which one finds himself confronted here is only limited by the vocabulary of the language. Furthermore, the problem is reasonably well-defined, i.e., in most cases the correct categorization choice is known by comparison with human performance. In some cases such performance by different people may not result in complete agreement, but even then the number of alternative results is restricted to a small number. Experimental data are readily available and their variability, insofar as they depend on subject and context, can be easily controlled.

The recognition of complex patterns by their subdivision into subpatterns deemed simpler to recognize has found wide application.<sup>1,2</sup> Since the probability for misrecognition of the individual subpatterns is finite, unless constraints are known to exist among the different subpattern categories, the likelihood for correct recognition of the composite pattern decreases rapidly as the number subpatterns is increased. The explicit rules for joining adjacent subpatterns are in many cases unknown and hence the constraints may be formulated only in terms of exhaustive listings of groups of subpatterns that may or may not exist. In other words, a collection of subpattern categories is determined to form a valid pattern if and only if there exists a pattern category which may be mapped into that collection of subpatterns. The task of testing whether a particular collection of subpatterns is a valid result is thereby reduced to the decision of whether such a collection may be generated from the allowable collection of patterns. Due to the large number of admissible pattern categories, in order to demonstrate the usefulness of such procedures,

\* This work was supported in part by the U. S. Army, Navy, and Air Force under Contract DA36-039-AMC-03200(E); and in part by the National Science Foundation (Grant GP-2495), the National Institutes of Health (Grant MH-04737-04), and the National Aeronautics and Space Administration (Grant NsG-496). It is taken in part from a thesis submitted by Paul Mermelstein in partial fulfillment of the requirements for the Doctor of Science degree to the Department of Electrical Engineering, Massachusetts Institute of Technology, January 14, 1964.

† Present address: Bell Telephone Laboratories, Inc., Murray Hill, New Jersey.

not only must we be able to recognize patterns with good reliability, but the utilization of the applicable constraints must be sufficiently efficient so that real time operation is feasible.

This paper reports experiments demonstrating feasibility for machine recognition of handwriting, given the pattern as a two-dimensional vector displacement of time. The time-dependent form of input presentation is used to limit the scope of the problem tackled. A concise numerical presentation for handwriting deemed more suitable for the programmed recovery of the message is presented. This representation preserves sufficient information so that with the aid of a suitable model for handwriting generation the two-dimensional pattern can be regenerated. The alphabetic representation of the message is recovered from the numerical representation with the aid of statistical estimates based on an ensemble of writing samples.

An algorithm for the segmentation of the pen displacement signal of handwriting into function segments corresponding to strokes has been presented.<sup>3</sup> This procedure, when repeatedly applied to a collection of handwriting samples yields function segments that may be classified into categories based on topological similarities, thereby yielding a representation for every cursive letter by means of a small number of permissible alternative sequences of stroke categories. Such categories are defined by means of statistical averages of the numerical representations of the member functions, and the likelihood of membership of new function segments in each particular category is estimated by means of the multi-dimensional distance between the representation of the new function and the average for the category.

Constraints are recognized to exist on two levels, those between stroke categories, sequences of which must form valid letter representations, and those between letters, sequences of which must form words within the vocabulary of the system. The result of the recognition process is that word which is generated from the stroke category sequence, the constituent strokes of which correspond to a maximum total likelihood. Methods are presented which implement the stroke sequence to word mapping in an efficient manner.

Preliminary experiments on the recognition of words selected from a limited vocabulary generated with the aid of only nine different letters have been previously reported.<sup>3</sup> These experiments have shown that good recognition is obtainable using the downstrokes of the writing only. A representation in terms of downstroke categories only is here given for the complete alphabet. Experimental results are reported for the recognition of 254 word samples under various conditions of machine learning. The test samples were generated by four writers from a vocabulary of 32 words in which each letter of the alphabet occurred at least twice. A dictionary of the 10,000 most frequently used English words<sup>4</sup> was used to limit the recognition results to words of the language.

## METHODOLOGY

A detailed study of the variations in writing speed for several subjects reveals points of speed minima predictably situated along the time axis. Examples of one test word written in turn by four subjects and the corresponding pen velocity functions are shown in Fig. 1. The writing segments delimited by such points of speed minima are found to correspond to regions in which the vertical component of the velocity does not change sign, i.e., they form the upstrokes and downstrokes of the writing. These segments or strokes are classified into categories on the basis of topological similarities, corresponding segments from the same letter as well as similar segments originating from different letters being assigned to one category, e.g., the first downstrokes of the letters *i*, *o*, *u*, *v*, and *w*. Similar writing segments are assigned to the same category unless such assignment precludes a unique representation for the letter by means of the categories to which its constituent stroke segments belong. An incidental property of this assignment is the fact that practically unique letter specification is obtainable by the downstrokes of the writing only. This representation is given for the lower case Latin alphabet in Table I. It should be noted that in going from a general stroke representation to one consisting of downstrokes only, a special downstroke category is added which consists of strokes introduced at points where two upstrokes are found to follow

each other. The representation so obtained is unique except for the letter pair *o-v* which usually can be resolved by the use of the contextual environment.

In order to define a measure of similarity between strokes we first transform the segment of the function giving pen displacement versus

time into a parameter vector. This vector contains sufficient information so that the displacement function corresponding to the stroke executed may be regenerated with the aid of a suitable model of handwriting generation<sup>3</sup> and will exhibit only minor differences with respect to the original function.

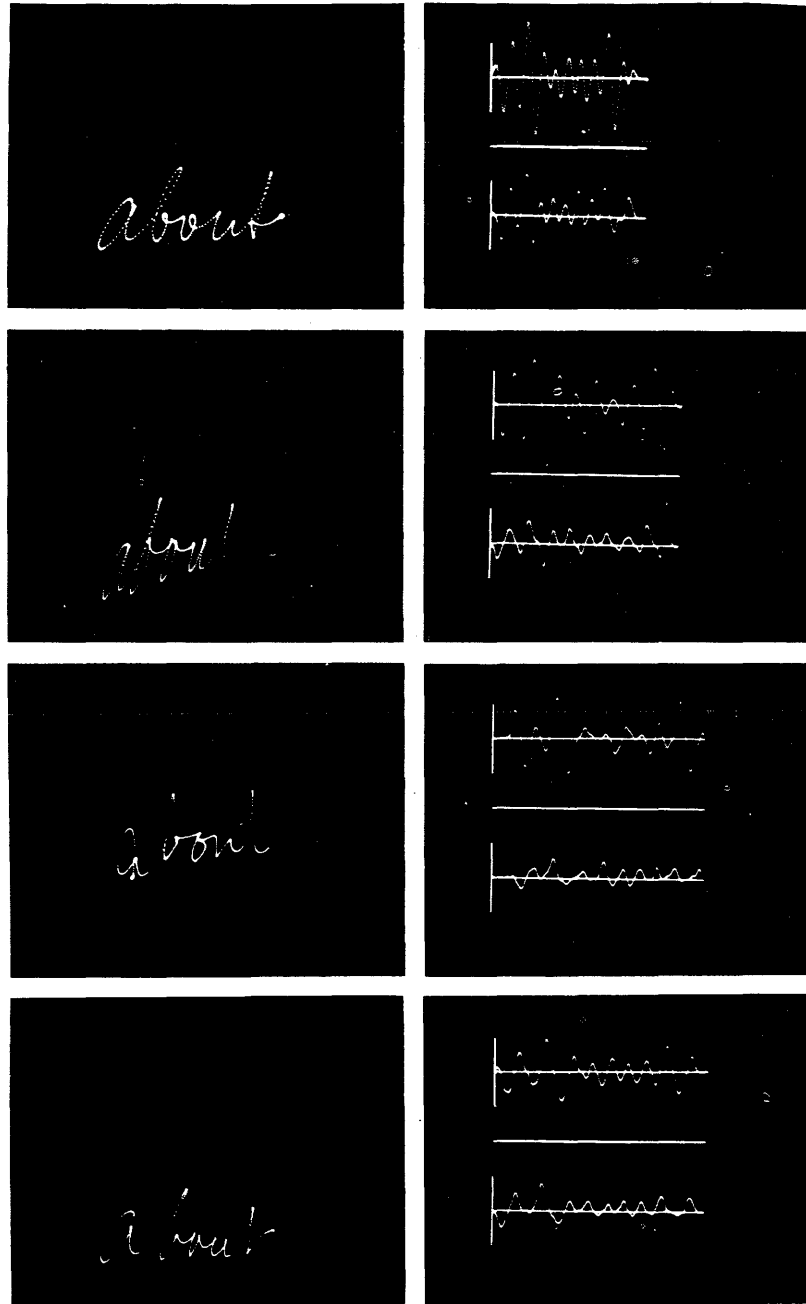


Figure 1. Handwriting samples and corresponding pen velocity functions redisplayed by the computer—top function  $y(t)$ , bottom function  $x(t)$ .

Table I. Stroke category representation for the lower case Latin Alphabet.

CHARACTER	LETTER VARIANT	DOWNSTROKE CATEGORY SEQUENCE	CHARACTER	LETTER VARIANT	DOWNSTROKE CATEGORY SEQUENCE
<i>a</i>	A1	D1 D2	<i>p</i>	P1	D6 D9
<i>a</i>	A2	SP D1 D2	<i>p</i>	P2	D6 D10 OD3
<i>ā</i>	A3	OD1 D1 D2	<i>q</i>	Q1	D1 D16 SP
<i>ɸ</i>	B1	D3 SP	<i>q</i>	Q2	D1 D16 OD2
<i>ɸ</i>	B2	D3 OD2	<i>q</i>	Q3	OD1 D1 D16 SP
<i>τ</i>	C1	D17	<i>q</i>	Q4	OD1 D1 D16 OD2
<i>τ</i>	C2	SP D17	<i>r</i>	R1	D15
<i>τ</i>	C3	OD1 D17	<i>r</i>	R2	OD4 D15
<i>d</i>	D1	D1 D3	<i>s</i>	S1	D18
<i>d</i>	D2	SP D1 D3	<i>s</i>	S2	D11 SP
<i>d</i>	D3	OD1 D1 D3	<i>s</i>	S3	D11 OD3
<i>e</i>	E	D4	<i>t</i>	T1	D12
<i>f</i>	F1	D5 SP	<i>t</i>	T2	D3 SP
<i>f</i>	F2	D5 OD2	<i>t</i>	T3	D3 OD2
<i>g</i>	G1	D1 D6	<i>u</i>	U	D2 D2
<i>g</i>	G2	SP D1 D6	<i>u</i>	V1	D2 SP
<i>g</i>	G3	OD1 D1 D6	<i>u</i>	V2	D2 OD2
<i>h</i>	H	D7 D8	<i>v</i>	V3	D14 SP
<i>i</i>	I	D2	<i>v</i>	V4	D14 OD2
<i>j</i>	J	D6	<i>w</i>	W1	D2 D2 SP
<i>k</i>	K1	D7 D9	<i>w</i>	W2	D2 D2 OD2
<i>k</i>	K2	D7 D10 OD3	<i>x</i>	X	D13
<i>l</i>	L	D3	<i>y</i>	Y1	D2 D6
<i>m</i>	M	D14 D14 D8	<i>y</i>	Y2	D14 D6
<i>n</i>	N	D14 D8	<i>z</i>	Z	D14 D16
<i>o</i>	O1	D2 SP	<i>z</i>	NULL STROKE	D17
<i>o</i>	O2	D2 OD2			

## LEGEND:

D1, D2, ---, D18 = STANDARD DOWNSTROKE CATEGORIES.

OD1, OD2, ---, OD4 = DOWNSTROKE CATEGORIES FOUND OPTIONAL IN SCRIPT.

SP = SPECIAL CATEGORY DENOTING VIRTUAL DOWNSTROKE OF ZERO LENGTH.

The model used for stroke generation is a modification of one first presented by Eden<sup>5</sup> and considers the velocity of the penpoint over any stroke to be representable by two pairs of quarter-wave sinusoidal segments, one for each of the horizontal and vertical components of the writing for the vertically accelerating and decelerating sections, respectively. The param-

eter vector components derived from that model such as displacement, curvature, pen velocity amplitude, etc. tend to prove useful in identifying the generated displacement patterns because they are likely to be invariant particularly with respect to those transformations of the patterns that leave the communicated information unchanged.

The parameters corresponding to an experimentally obtained function segment are determined by synthesizing a generating function that will best match the original function in the least mean square sense. There being seven independent parameters in the model, we impose six constraints on the functions to be generated, leaving one degree of freedom. Perturbations are applied to one parameter, the others being recalculated to satisfy the imposed constraints, until a local minimum is found for the difference measure as a function of that parameter. Further non-independent parameters, considered as possibly useful for recognition, are appended to the previous parameter set resulting in twelve component parameter vectors.

The recognition procedure is effected in two successive stages. First, an attempt is made at independent stroke recognition or the classification of each function segment independently into predetermined stroke categories. The reliability attained at this stage is strongly dependent upon previous knowledge of the subject's writing insofar as that information is represented by the statistics pertaining to the several categories. Second, ordered sequences of the selected strokes are considered and the applicable constraints are used to eliminate the sequences not corresponding to words, thereby increasing the likelihood for correct recognition. This stage is independent of the ensemble on which the machine's representation of the writing is based. It is implemented in a manner that allows the introduction of additional constraints while their effects are under investigation.

#### *Recognition of Strokes*

An optimal rule for recognition of membership in categories is one that calculates for all categories the conditional probability that a particular stroke parameter vector from a selected category takes on a value identical to that of the stroke to be identified and assigns the unknown stroke to that category for which this conditional probability is maximum.<sup>6</sup> In most pattern recognition problems, including this particular case, the required parameter statistics for the several categories are unavailable and may be estimated only by a prohibitive amount of sampling. By making assumptions about the nature of the probability density

functions of the parameters of the several categories on the basis of limited data, we may overcome the estimation problem, but the recognition results will fall below optimal to the extent that these assumptions fail to be valid. In particular, we assume that the parameters describing the strokes have multivariate normal distributions with generally unequal covariance matrices for the several categories.

Statistics consisting of the vector of parameter means and the covariance matrix of the parameters are computed for each stroke category from the parameter values of the stroke segments assigned to those categories. This assignment is carried out manually, and depending on the source of the samples, the resulting representations may correspond to a group of writers, or separate representations are obtainable for individual writers. When several subject dependent representations are available the recognition process may be modified to identify the script of a given word as corresponding most closely to a particular one of the available representations.

#### *Recognition of Letter Sequences and Words*

The constraints between adjacent strokes are formulated in terms of stroke sequences which may correspond to the letters of the alphabet, and a list of the valid letter sequences, i.e., the vocabulary of the system. Most stroke sequences generated at random are not mappable into words. We desire to arrive at the sequence which among those mappable into output words has maximum likelihood as given by the sum of the constituent stroke likelihoods. Unless the sequence formed by the most likely stroke candidates is mappable into a word, less likely stroke candidates must also be considered. Conceivably, whenever an unacceptable sequence is found, we could make the substitution that least decreases the total likelihood for the sequence. Since not only single, but also multiple substitutions, must be considered and the independent stroke recognition rate is not expected to be very high, repeated substitutions are necessary resulting in an impossibly time consuming procedure. Clearly, the number of ordered sequences of all stroke candidates is so large as to make an unordered consideration prohibitive.

We may find the most likely sequence with a high probability by establishing a threshold on the individual stroke likelihoods and examining all sequences generatable from the strokes exceeding that likelihood. If no sequence satisfying the constraints is found, the threshold value may be relaxed and the procedure repeated. This procedure will miss the most likely sequence if that sequence contains one member having a likelihood appreciably below the mean likelihood for the sequence, as well as below the threshold, and there exists a different valid sequence with somewhat lower mean likelihood all of whose members exceed the likelihood threshold. A strategy of this nature has been first suggested for the sequential decoding of information encoded for transmission over noisy channels.<sup>7</sup>

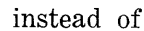
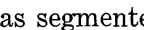
On the assumption that most of the hypothesized stroke sequences will not be transformable into output words, the following procedure for the exhaustive consideration of all possible ordered sequences of the stroke candidates proves efficient. Starting with the first choice for the first stroke, attempt a continuation with the first choice for the next stroke, continuing until either an illegal sequence (no corresponding letter sequence) is found or the last stroke of the word is reached. For illegal sequences, the last stroke choice selected is dropped and a continuation is attempted with the next choice, if any, for the same stroke. If no such continuation is found possible, one more stroke is deleted and further choices for that stroke are considered. Whenever enough strokes have been processed to complete an additional letter, the letter string found up to that point is checked against the output vocabulary for presence as the initial subsequence in an admissible letter sequence. Words are accepted as possible results if the last stroke of the sequence is found to terminate the last letter of the word and ordered by assigning likelihoods based on the sum of the constituent stroke likelihoods.

## EXPERIMENTAL RESULTS

Recognition experiments were performed on a set of 32 different words written repeatedly by four different subjects. The set of words was chosen randomly and therefore reflected

roughly the letter distribution in the language, but contained at least two occurrences of each letter. Diacritical marks were omitted and subjects were asked to write continuously, not crossing the letters *t* and *x* where it necessitated a break in the continuity of the writing. A dictionary of the 10,000 most frequently used English words was used to limit the recognition results to words of the language.

The words were written individually on a handwriting transmitter (Telautograph) having a 4" x 2" writing surface (see Figure 2). The transmitter signals corresponding to pen position were fed to the TX-O computer (Department of Electrical Engineering, Massachusetts Institute of Technology), the computer being used as a multiplexed analog-digital converter and for the recording of data on magnetic tape. The IBM 7090 and 7094 computers of the Computation Center, Massachusetts Institute of Technology, were used for all subsequent data processing.

Of the total of 254 samples, 249 were successfully segmented into stroke sequences. In the other five cases, as a result of the smoothing of the normal direction changes in certain contexts under conditions of rapid writing, the last upstroke-downstroke pair of a letter and the following ligature were found to be inseparable by the segmentation algorithm used, as illustrated by writing  instead of . In two other cases the words, as segmented by the program, were not recognizable with the aid of downstrokes alone because the last downstroke of the last letter of the word was not explicitly executed, e.g.,

Recognition was attempted on the remaining 247 words by using a stroke representation of the letters based on downstrokes only and compiled from the strokes constituting those words. The stroke partition utilized consisted of 22 downstroke categories, one of which corresponded to downstrokes found at the beginning and end of words which did not form part of the first or last letter, and was therefore assigned to the null letter. The stroke classification was carried out by using all 12 computed parameters and treating them as if they were independent. Eighty per cent of the words on which recognition was attempted were cor-

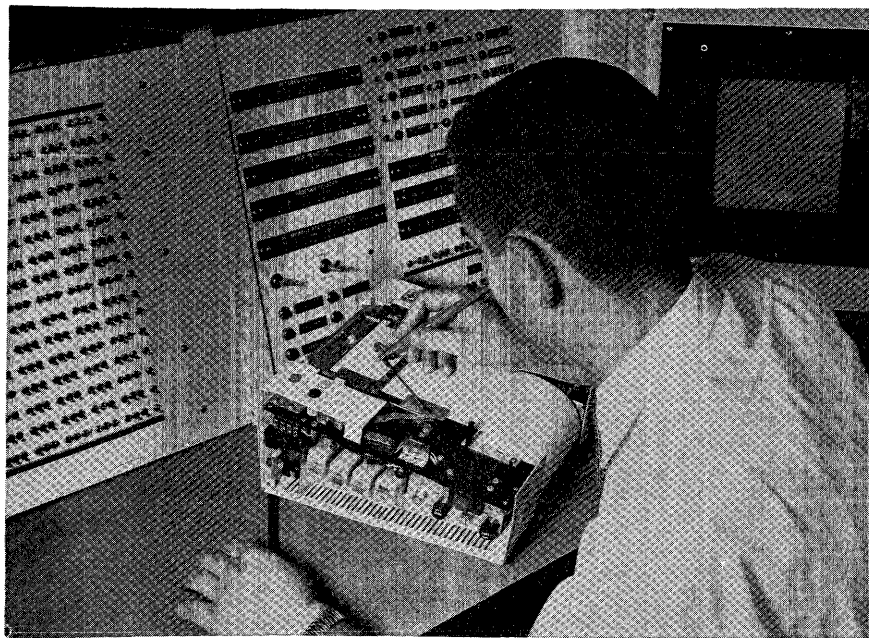


Figure 2. Writer using Telautograph transmitter.

rectly identified. Of the 49 samples incorrectly identified, 14 had the correct word selected as the second choice, 7 as choices lower than the second, 26 did not give the correct word as one of the 20 possible choices in the threshold range, and recognition of 2 samples had to be terminated when no result was obtained after 20 minutes of processing.

The experimental strategy for word recognition, namely, repeated attempts with successively lower stroke-likelihood thresholds, does not eliminate the possibility that if some lower initial threshold setting were used, a previously incorrect decision might be performed correctly or a correct decision might be upset. In order to observe the frequency of this phenomenon, 12 word samples for which the correct word was initially not selected were reprocessed by using a value for the initial threshold setting that permitted consideration of strokes having likelihoods down to half the previous minimum likelihood. Six of these words were now correctly selected as first choices in the recognition output. The reason for this improvement in performance is that in certain cases the correct category may lie just beyond the likelihood threshold value and therefore be missed, while all of the other strokes are correctly recognized. We may, of course, process all samples with the

higher threshold value initially, but this frequently results in an unwarranted sizable increase in the required processing time for word recognition.

Next, independent stroke statistics were computed for each subject, and recognition was attempted on the previously incorrectly recognized samples by using representations derived in each case from the test subject's handwriting. Forty-three of the previous 49 errors were now correctly recognized. Since 40 of the 198 samples previously correctly recognized were correctly recognized now as well, and none was misrecognized, we may assume that all of the 198 samples would have been correctly recognized thereby resulting in a subject-dependent recognition rate of 98 per cent. Time requirements for recognition in this experiment fell to an average of 9 sec on the IBM 7094 computer, as compared to 30 sec for the subject independent recognition.

One may gain insight into the magnitude of the contribution made by contextual information from the fact that in the set of strokes, when considered individually in the subject independent experiment, the calculated most likely category was in fact the correct category in only 58 per cent of the cases. Hence, the

probability that none of a sequence of say ten strokes was in error was  $(0.58)^{10}$  or 0.43 per cent.

Clearly, in carrying out the initial attempt at individual stroke recognition, differences may be expected between the relative utilities of the several parameters describing the strokes. The misidentification of a particular stroke may or may not result in the misrecognition of the complete word. The extent of any improvement in word recognition rates on the basis of specific improvements in the stroke recognition rate cannot be readily estimated because of the complexity of the applicable constraints. Therefore all errors were weighted equally in calculating the overall stroke recognition rates.

For the partition of the downstroke space corresponding to the alphabet representation given in Table I, a partial investigation of the relative contributions of the several parameters was carried out, based on stroke data from the above 249 samples. The effect of the assumption that the parameters may be treated as if they were independent was also investigated. Although the assumption of independence among the parameters is known to be invalid, their treatment as if they were independent is justifiable if the reduction in complexity of the required calculations and the amount of additional statistical data required can be obtained at the cost of only a small reduction in the recognition rate.

The stroke parameters used in the recognition experiments are defined below.

1. X component of stroke displacement
2. Y component of stroke displacement
3. X component of initial stroke velocity
4. X component of final stroke velocity
5. Frequency of sinusoid matched to initial stroke segment
6. Frequency of sinusoid matched to final stroke segment
7. Initial segment phase shift of x velocity relative to y velocity
8. Final segment phase shift of x velocity relative to y velocity
9. Amplitude of matched y velocity sinusoid

10. Amplitude of matched x velocity sinusoid for initial segment
11. Amplitude of matched x velocity sinusoid for final segment
12. Constant component of matched x velocity

When the dependence among the parameters is considered for the subject dependent case, the stroke recognition rate is increased from 58 to 62 per cent. Several subsets of parameters were considered and none yielded a recognition rate higher than 63 per cent. The corresponding stroke recognition rates for the four subjects treated individually ranged from 68 to 81 per cent, averaging 74 per cent. It should be noted that the increase in stroke recognition rate from 58 to 74 per cent was sufficient to raise the word recognition rate from 80 to 98 per cent. The parameter subset {5-12} containing only parameters used in the handwriting generation model, when compared to subset {1-4}, the one composed of the arbitrarily selected parameters, proved significantly poorer in the subject independent case, yet equally good in the subject dependent case. Evidently the detailed dynamic characteristics of the writing while consistent in the case of particular subjects reflect too much subject to subject variation to prove useful for subject independent recognition. For the subject independent case, once a minimal set of parameters is selected, the stroke recognition rate is not found to be a very sensitive function of the number of additional parameters.

## CONCLUSIONS

This report presents an approach to machine recognition of handwritten script words when written on-line, i.e., with the time information of the writing available. Experiments carried out by general purpose computer simulation of the recognition system reveal that the system is capable of recognizing well-formed, legible handwritten words with a reliability that depends on the correspondence between the script of the writing sample and that of the ensemble on which the machine's representation of handwriting is based. The resulting recognition rates are found to be significantly better than those previously reported.<sup>8</sup>



A detailed study of the behavior of the recognition system shows the following:

- (a) Recognition reliability approaching that of humans is attainable if the machine's representation is based on the writing of the same subject who produced the samples to be recognized.
- (b) When the machine is not given a representation originating from the same subject, but one based on an ensemble written by a number of individuals, samples from that group of individuals are recognizable, but with lower reliability.

Previous experiments<sup>3</sup> show that performance deteriorates further when we attempt to recognize writing samples of subjects not included in the ensemble from which the machine's representation is derived.

The problem of selection of a set of parameters for stroke description that is to a large extent independent of subject and context, one clearly of great importance for achieving reliable recognition, has not been solved with any degree of finality. The system establishes a framework within which the utility of the various parameters may be investigated. It also permits the evaluation of the contribution to recognition which word context provides.

Our simulation experiments demonstrate that good recognition is obtainable by means of the downstrokes of the writing only. There exist a number of ambiguities that are in general difficult to resolve by means of downstrokes alone, such as differentiation between the letters *p* and *k* or the pair *cl* and the letter *d*. It is suggested that use should be made of upstrokes as well only in environments giving rise to such ambiguities, and not otherwise, thereby reducing the complexity of and time requirements for the recognition task.

The program's failure to discriminate correctly between two strokes can in certain instances be due to the method of partitioning the stroke space. A point in the parameter space corresponding to a particular function segment of the test sample is assigned a likelihood with respect to each stroke category in the stroke alphabet. However, if the several hierarchies of constraints restrict the choice for a given segment to one of a very limited

number of stroke assignments, then we may possibly make improved discriminations on the basis of a restricted set of parameters particularly applicable to the discrimination to be made. It is therefore proposed that the final discrimination decision be based on the results of an iterative sequence of successive approximations to the desired likelihood measures.

The most important deficiency of the program appears to be the lack of facilities for modifying stroke likelihood decisions on the basis of the stroke environment. Stroke recognition based on information pertaining only to individual strokes is found frequently to be inadequate in low contextual information situations. For example, differentiation between the words *fall* and *full* could be appreciably aided by making available to the recognizer, the horizontal distance between the initial points of the first and second downstrokes of the letters *a* and *u*. The difficulties, which may in most cases be recognized by a likelihood ratio between the two word choices which is nearly one, are in many cases resolvable if further information pertaining to the specific letter-letter confusion can be supplied on demand from the recognizer.

The requirements for on-line input of writing samples places strong limitations on the practical applications of the methodology presented. The underlying stroke representation may, however, be adapted for such applications by using functions segmented at the zero spatial derivative of the *y* displacement function. By replacing time derivatives by spatial derivatives, we may arrive at a list of parametric descriptors which are adequate to differentiate among the stroke set. The immediate and most significant problem that remains is the design of practically applicable algorithms for spatial segmentation of handwritten words into strokes.

## REFERENCES

1. BLEDSOE, W. W., and BROWNING, I., Pattern Recognition and Reading by Machine, *Proc. Eastern Joint Computer Conference*, 1959 Eastern Joint Computer Conference, pp. 225-232, (1959).
2. GRIMSDALE, R. L., SUMNER, F. H., TUNIS, C. J. and KILBURN, T., A System for the

- Automatic Recognition of Patterns, *Proc. IEE (London)*, 106 pt.B., 210–221, (1959).
3. MERMELSTEIN, P. and EDEN, M., Experiments on Computer Recognition of Connected Handwritten Words, *Information and Control*, 7, 255–270, (1964).
  4. THORNDIKE, E. L. and LORGE, I., *The Teacher's Word Book of 30,000 Words*, Columbia University, New York, (1944).
  5. EDEN, M., Handwriting and Pattern Recognition, *IRE Trans on Information Theory*, IT-8, 160–166, (1962).
  6. MARRILL, T., and GREEN, D. M., Statistical Recognition Functions and the Design of Pattern Recognizers, *IRE Trans. on Electronic Computers*, EC-9, 472–477, (1960).
  7. WOZENCRAFT, J. M. and REIFFEN, B., *Sequential Decoding*, Technology Press and J. Wiley and Sons, New York, (1961).
  8. FRISHKOPF, L. S. and HARMON, L. D., Machine Reading of Cursive Script, *Information Theory, Fourth London Symposium*, C. Cherry, ed., Butterworths, London, pp. 300–315, (1961).

# A LABORATORY FOR THE STUDY OF GRAPHICAL MAN-MACHINE COMMUNICATION

*Edwin L. Jacks*

*Research Laboratories, General Motors Corporation, Warren, Michigan*

## INTRODUCTION

Engineering has evolved rapidly during the last fifteen years as analysis techniques geared to the computational power of a slide rule and desk calculator have been replaced by techniques which make extensive use of computers. During these years, however, graphical techniques for conversion of design ideas to final products have not changed significantly, nor has the role of drawings in engineering design changed. The drawing plays a vital role in each phase of the evolution of a product. The original design proposals, the engineering analysis, the design compromises, and the prototype product fabrication all depend on graphical communication among engineers and designers. Whether the product is to be machined, assembled, stamped, wired, welded or hand modeled, a drawing is made so that a two-dimensional representation of the product may be reviewed by the engineers concerned with the product. Prior to the final product drawing, many ideas are exchanged by the use of sketches, drawings, plots, and engineering reports.

The drawing board is used as the basic mechanism for resolving problems in design packaging. For instance, "Where can part B be located if part A is made larger?" and "Can part A be assembled to part C?" As the design evolves, many decisions are made by the engineer while the drawing representing the design is being produced by the draftsman. The questions are endless, and, in many design prob-

lems, are not finally resolved until after early prototypes of a product are made.

Two key points in this process are: (1) the engineer is an integral part of the graphic design process, and (2) the draftsman is doing a task that requires considerable attention to detail and mechanical precision. In many mechanical design situations the two functions of engineering and preliminary product drafting are done by the same man. A drawing serves as his way of exploring design ideas.

The dependence of engineering design on graphical techniques is fundamental to the design process. Graphics serves as a language of communication among design personnel and, as outlined above, as a mechanism for design evolution.

The General Motors Research Laboratories had been using digital computers for engineering and scientific analysis for several years dating back to a card-programmed calculator in 1952, but notably absent from the applications were problems relating to graphical design. In the late 1950's, the Research Laboratories addressed the question, "Could computer techniques significantly improve the design process?". To answer this question, a study was started on the potential role of computers in the graphical phases of design. Prototype hardware and software components were developed to investigate the problems of image processing. A 740 cathode ray tube recorder attached to the 704 computer was already being used to

plot results of engineering computations. It satisfied the requirement for graphical output. The associated 780 display unit provided a graphical on-line display which, along with a simple switchboard, became an elementary man-machine console. A program-controlled film scanner was devised using the 740 recorder; a photocell detector was substituted for the film magazine, and its output signal was connected to a computer sense switch. With this breadboard setup, lines on film could be digitized under program control. Programs were written for graphic input and output and for the manipulation of images in three dimensions. These early software and hardware components were integrated into an operating system that demonstrated the feasibility of using the computer as an aid in the graphic design process.

On the basis of this early feasibility demonstration, the decision was made to establish a more comprehensive laboratory for graphical man-machine communication experiments. The facilities were to permit the computational power of a large-scale digital computer to be brought to bear on the problems of graphical design in a manner which fully recognized the importance of the man in design. This project on Design Augmented by Computers has become known as DAC-I.

The initial goal of the Design Augmented by Computers project was the development of a combination of computer hardware and software which (a) would permit "conversational" man-machine graphical communication and (b) would provide a maximum programming flexibility and ease of use for experimentation. This goal was achieved in early 1963. This paper gives a broad outline of the computer technology which was developed to meet the above goal. Other papers<sup>1,2,3,4</sup> present approaches to solutions and examples of performance of the various hardware and software components of the system.

The present hardware complex consists of an IBM 7094 computer and an IBM 7960 Special Image Processing System. The Image Processing System was designed and built by IBM to specifications provided by the General Motors Research Laboratories (GMR).<sup>2</sup>

The supporting software was developed by the GM Research Laboratories Computer Technology Department and includes a multiprogramming system, an algebraic compiler (NOMAD), a data channel command compiler (MAYBE), a dynamic storage assignment procedure, and extensive facilities for the storage, retrieval and editing of programs and data stored on a disk storage device.<sup>1</sup>

Each major portion of the DAC-I system—the 7960 Special Image Processing System, the computer with its attached disk memory, the multiprogramming trap control system, the DAC-I monitor system, the programming languages used for system development and the disk filing programs—have, via their design criteria, all contributed to the system's flexibility and ease of use for experimentation.

#### DESIGN OBJECTIVES OF THE IMAGE PROCESSING HARDWARE

The over-all objective of the image processing system was to achieve the equivalent of what is possible with graphical man-to-man communication while utilizing drawings. In establishing systems specifications, four types of man-machine communication were sought.

The *first* type of drawing communication desired was static. The machine should be able to produce a hard copy drawing for engineering use. Conversely, it should be able to accept a drawing and be able, under computer control, to read the drawing. Because of the nature of automobile design, it was necessary that the DAC-I system be able to accept free form curves, i.e., curves which are constructed without consideration of particular mathematical representations. Furthermore, to provide compatibility with existing design procedures, precision input and output of such curves was needed. These requirements ruled against a "sketchpad" type of operation.<sup>5</sup> The drawing input-output functions have been achieved in the image processor of the 7960 System by: (a) using a high resolution cathode ray tube (CRT) under computer control to record drawings onto 35 millimeter film and (b) using a second and similar CRT as a computer-controlled flying spot scanner to scan 35mm film images of drawings. The image processor has built into it the ability to photo-

seconds have the film ready for optical scanning. The output drawings are also ready for viewing 30 seconds after film exposure.

The *second* type of drawing communication desired was dynamic. The system should simulate the type of man-to-man communication where one man is drawing or pointing at a particular part of a drawing while another man is observing or discussing details of the design with the first man. This capability was provided in the graphic console of the 7960 through the combination of a 17-inch display tube and a device called a position-indicating pencil. When a designer touches the pencil to the glass plate in front of the display tube, the computer program can detect what position on the tube face is being pointed to and, hence, can react to any comments the man may wish to make about the indicated portion of the display. Thus, after the computer generates a picture on the display tube, either the man (by pointing at the display) or the computer (by placing an "x" on the displayed picture) can in effect say to the other, "Consider this portion of the picture."

The *third* drawing communication objective was simulation of the comparison function. The system should allow the overlay of two pictures to permit comparisons of the differences and similarities in the information.

This feature was provided by having the image processor designed such that pictures can be recorded on two separate film trains and then projected automatically onto a common view screen. This feature allows, for example, overlay of scanned data with the original film source for verification. By programming techniques, the graphic console can also be readily used to compare drawing information.

The *fourth* design objective was to achieve man-machine communication of non-graphic information. The system should provide, via the graphic console, a convenient means of communicating (a) alphabetic and numeric information to the computer, (b) multiple choice decision responses to the computer, and (c) permissible actions by the man. For alphanumeric information, a 36-position keyboard with upper and lower case and a slow-speed card reader is part of the graphic console. For communication of gross actions, the console has 36

program control keys and 36 message lights; the computer receives a signal when a program control key is depressed by the man, and inversely the man receives a visual signal when the computer turns on a message light.

A detailed description of the 7960 Special Image Processing System is contained in the paper by B. Hargreaves, J. D. Joyce and G. L. Cole, et al.<sup>2</sup>

#### OBJECTIVES OF THE COMPUTER HARDWARE COMPLEX

Studies at the GM Research Laboratories in 1959 and 1960 were made to estimate the computing facilities required to adequately support the DAC-I project. Considered in the studies were the number of instructions required to support the experiments, execution time for the required programs, and man-machine response rate. These studies indicated that approximately 200,000 to 500,000 instructions would be programmed for the graphic communication experiments. The computation required for these experiments was estimated in terms of central processing unit use per hour and amounted to 6 minutes of 7090 time for each hour of console use.

The response rate considerations were stated in terms of system objectives. We wanted the designer to be essentially working on-line and in "real time." The measure of real time was that the man and machine could carry on a meaningful conversation about a design at a rate satisfactory to the man. The response consideration then required a real-time approach to receiving and handling data arriving from the man. But the computer programs and hardware did not need to have a fail-safe time limit approach to sending a response to the man. For this reason the system is best described as on-line console system rather than a real-time system.

Another more independent consideration was the computing requirements of the GM Research Laboratories. In 1959 and 1960, a 704 was in use between two and three shifts per day, and it was forecast that a 7090 or equivalent computer would be required by 1961 to satisfy the continuing needs for an engineering graph 22 x 22 inch drawings and within 30

and scientific computing facility at GM Research.

The combination of the above requirements, we believed, could be met by a 7090 computer\*. The speed of the 7090 would adequately handle the computational load and, if properly multiprogrammed, the machine would effectively be able to give the response time desired for console communication and computational purposes without wasting the estimated 54 minutes per hour of non-console use. The requirement for multiprogramming implied that the computer would need to be modified such that two independent programs could reside in its core memory with a minimum risk of either program modifying the other program. For this purpose, a core memory protection system was designed which prevents instructions from storing into program-specified 4K blocks of the memory.

Multiprogramming also implied that for accounting purposes a clock be attached to the computer so that proper timekeeping could be performed during the switching from program to program. A clock was built by the Delco Radio Division of General Motors for GMR with a millisecond as its basic interval of time.

The requirements for  $0.5 \times 10^6$  words of program storage could be satisfied by having a disk memory on the computer. The original 7090 configuration had a 1405 disk connected via a 1401 and a direct data connection to the computer. The current facility uses a 1301 disk and three drum storage units for the program and data library. The computer complex resulting from the above set of specifications is shown in Figure 1.

#### OBJECTIVES OF PROGRAMMING SYSTEMS SUPPORT

The combination of the IBM 7090 and the IBM 7960 system as described above was to provide an experimental graphical communication hardware facility. To support this system from the software standpoint, it was decided that an investment would be made in programming techniques which would minimize the time

from the conception of a man-machine communication experiment until the required programs were operating.

Figure 1 shows the computer as a central processing unit with five attached data channels. That is how the machine appears to the hardware man. To the people responsible for programming graphical communication experiments, however, the machine was to have an entirely different appearance. These people were to be in a programming position in which a large library of procedures were at the "finger tips" of their programs.

The programs were to be able to conveniently display situations to the man. If the man was expected to require more than a millisecond to respond, the programs were to be able to say to a control program, "Control, I am in standby status now and when the man answers my question or takes other action, return control to me."

For programming convenience, the programmer should be able to do all his programming in a higher level language (higher than an assembly type language at least) including the programming of the data channel driving the 7960 System, the loading of programs by name from the disk, and the analysis of all data coming from the image processing or graphic console equipment. In short, he should be able to program all of his graphical communication experiments in a language similar to FORTRAN or ALGOL. An algebraic compiler (NOMAD) and a data channel compiler (MAYBE) were developed for this purpose.

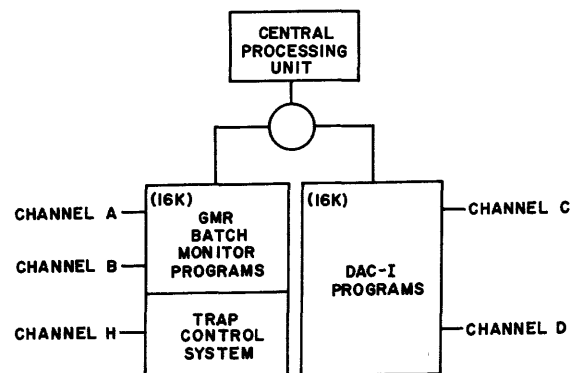


Figure 1. Computer Configuration.

\* In 1963, the originally installed 7090 was upgraded to a 7094.

The specifications of the programming system revolved around three broad statements of facility operational policy. First, for programming purposes, the 32K computer memory was to be considered as two 16K blocks of memory\*. This is represented symbolically in Figure 2, with 16K assigned to the DAC-I console support programs and 16K assigned to the standard batch monitor operation.

Second, all input/output programs in both the DAC-I operation and the batch monitor operation must use the trapping hardware built onto the 7090 and all trap program operations must be compatible with a trap control system (TCS) developed at GM Research.

The third policy statement indicated that the batch monitor's use of the computer was limited to channels A and B while the DAC-I console program's use was limited to channels C and D. This condition was imposed to prevent conflicts in hardware use and means, for instance, that tapes in use by the batch monitor could not be used by DAC-I during multiprogramming. One major exception to this rule was that the use of the disk was permitted by programs being executed under batch monitor control for purposes of compiling or checking out programs being developed as part of the DAC-I project.

TRAP CONTROL SYSTEM SPECIFICATIONS

Based on the above operation policies and on the programming system's objectives, specifica-

\* As of March, 1964, the 7094 was expanded to two 32K memories.

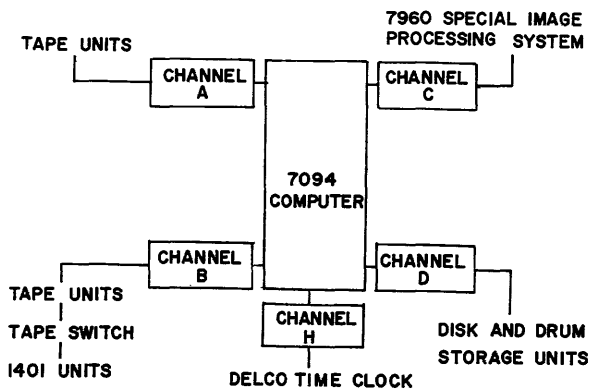


Figure 2. Split-Memory Operation.

tions for the trap control system's performance in multiprogramming and the programming techniques for DAC-I were developed.

The trap control system (TCS) was to meet the following broad specifications: (Refer to Fig 2)

1. DAC-I channel traps terminate monitor job central processing unit (CPU) operation. Machine status is saved by TCS prior to transfer of control to the appropriate DAC-I program.
2. If DAC-I has CPU control, monitor job channel traps are saved for later action. Monitor job traps are processed immediately upon return of CPU control from DAC-I.
3. TCS must switch memory protection regions as CPU control is switched.
4. TCS must honor DAC-I requests for additional memory space by a dump and memory protect release of all remaining core (except TCS). Restore and restart procedures for monitor jobs must be provided.
5. DAC-I channel traps should not be inhibited unless absolutely necessary.

MONITOR SYSTEM'S OBJECTIVES

With the above type of trap control system, the batch monitor and DAC-I program sequencing monitor each had distinct operating objectives.

1. The batch monitor should be a general purpose batch processing monitor and should be able to execute any program as long as the program was compatible with the trap control requirements and the 16K core limitations.
2. The DAC-I monitor was to accept from the graphic console card reader a single card containing job accounting information and a program name. The program name was to be the name of any subroutine stored on disk. The monitor loads the subroutine and turns control over to it.
3. Any program being executed on the DAC-I side should be able to enter a job into the batch monitor job stack. This permits conventional printed and punched

output from a DAC-I program without using channels A and B at the time the output is generated. The disk is used as the temporary storage space for the output while waiting for a between job break in the monitor operation.

4. The DAC-I programming system should be duplicated within the scope of the batch monitor. This permits debugging of programs independent of the 7960 System and prior to execution in the DAC-I environment. When operating in batch monitor mode a basic monitor facility must be provided for the compiling of DAC-I programs in the NOMAD language and the subsequent filing of the resulting object program in the disk file.

### THE DAC-I MONITOR

The DAC-I operating monitor was developed around the idea of a disk program library where all system functions and program execution sequences are built up from a) the basic operations of storage, retrieval, and updating of a library, and b) the allocation by the currently running program of memory core space for program and data.

Basic concepts of the system are:

- a) The basic unit of a program is a subroutine which has a name, an entry point name, and a disk file area name.
- b) Data for a subroutine may be either global or transmitted to it by a standard subroutine calling sequence. A global variable is declared at subroutine compile time, but no memory locations are assigned until the subroutine is loaded into core.
- c) Program execution involves the loading of subroutines from disk as they are needed. It is the function of the loaded subroutine to assign subsequent locations within memory for whatever additional subroutines and global variable assignments are required for the program's task.

Based on the above concepts, the DAC-I monitor was required to provide:

- a) A table which contains the location and size of each subroutine in memory.

- b) A table which contains the location and size of all global variables in memory.
- c) The basic codes required to retrieve programs and data stored in the auxiliary disk and drum memories.
- d) A relocation program which, when given data in the form of a subroutine in memory, will relocate the subroutine and assign memory addresses to its global variables based on the subroutine and global variable tables mentioned in a) and b).

The basic facility ground rule was that given essentially the above codes any program written in NOMAD could then at execution time do its own storage allocation. The paper by M. Phyllis Cole, Philip H. Dorn, and C. Richard Lewis<sup>1</sup> describes the procedure for storage allocation (itself written in NOMAD). The method is basically program subroutine selection at execution time and it allows a programmer to make decisions in his programs as to which is the best method for handling the storage assignment for a given data set at execution time. He may either keep a large block of data in memory and pass programs by the data, or keep all his programs in memory and pass the data by his programs. In practice, for small data sets, the programmer keeps all of his program in memory. As the data set becomes larger, initialization, computation, and post-processing subroutines are cycled by the data.

With the combined facility of disk program retrieval by subroutine name and memory storage allocation at execution time, a very useful feature develops—any alphabetic data can be viewed as a subroutine name. This permits convenient modular expansion of programs.

### SYSTEM PERFORMANCE

The system has been in operation eight hours per day since early 1963. In this time we have been utilizing extensively the hardware and software previously outlined.

The paper by F. Krull and J. Foote<sup>4</sup> illustrates the combination of computer-controlled image scanning and man-machine communication. Where the input film is high contrast and there are basically no uncertainties, a simple computer program rapidly solves the problem of conversion from graphics to binary data.



When uncertainties, such as arise when scanning low contrast film, become the dominant problem, then the man, as referee, can obtain control. One can argue that for each uncertainty a program can be written to analyze the situation and then the man is not needed to aid the process. The strong point of man-machine communication via graphic consoles is that for any given problem, one may now ask which parts of the problem are easily solved by the computer and which parts are best solved heuristically by man. This results in programs being written which have decision points in them at which the man at the console can be asked for advice. Many of the past discussions of man-machine communication have been based on the concept of "let the man get to the computer" so he can directly ask questions of the computer program. Experience at GMR to date has been that the payoff from consoles comes not from asking the computer a question but assigning the computer a task from which the response is one of the following: "What is my next job?" "Here is the answer; what next?" or, "I don't understand; and here is my analysis of the situation."

From the standpoint of a laboratory facility, the system is performing excellently. We are learning that man and machine can communicate readily via graphical means.

## SUMMARY

The software development for the graphical man-machine communication laboratory has incorporated three major departures from conventional higher level language programming: 1) multiprogramming, 2) source program storage allocation control, and 3) a disk library of programs available during program execution. Each of these programming techniques is essential to the concept of Design Augmented by Computers. Multiprogramming permits computer programs to be written such that, even though they work at the man's pace, they achieve efficient utilization of the computer's processing unit. Program storage allocation control allows each program to adjust storage assignment dynamically as a function of data needs.

A disk library available at execution time allows a control subroutine to view other subrou-

tines as black boxes which required certain inputs and produce outputs. The size and name of the black box does not need to be known at programming time and, in fact, are data at execution time, for the control subroutine. This feature allows continued growth of the design support programs with no change to control programs.

The above three software techniques combined with the flexibility of NOMAD, permitted a fourth major departure from conventional programming techniques. Ninety percent of the DAC-I programming system was written in NOMAD. The trap control system and the basic subroutine relocation programs were the major exceptions to the above. With the new laboratory facilities at GM Research, the process of man-machine communication for design can now be explored with both formal experiments (direct comparisons of methods with planned testing) and informal experiments (let's try something to see how it works).

## ACKNOWLEDGEMENTS

The DAC-I software system is the product of many people at the GM Research Laboratories. The four papers (1, 2, 3, 4) associated with this paper reflects the contributions of their authors. The work on the programming system was directed by Charles S. Gerrish, and the batch monitor system was developed under the guidance of George F. Ryckman. The trap control system was developed by Floyd Livermore, and the linkage system to the DAC-I operation was done by Theodore J. Theodoroff.

Throughout this project Professor Bernard A. Galler of the University of Michigan has served as a consultant to the Research Laboratories on the development of the DAC-I software support program.

## REFERENCES

1. COLE, M. P., DORN, P. H., LEWIS, C. R., "Operational Software in a Disc Oriented System," *Proceedings of the Fall Joint Computer Conference*, San Francisco, California, October 27-29, 1964, (this issue).
2. HARGREAVES, B., JOYCE, J. D., COLE, G. L., et al., "Image Processing Hardware for a Man-Machine Graphical Communication

- System", *Proceedings of the Fall Joint Computer Conference*, San Francisco, California, October 27-29, 1964, (this issue).
3. ALLEN, T. R., and FOOTE, J. E., "Input/Output Software Capability for a Man-Machine Communication and Image Processing System," *Proceedings of the Fall Joint Computer Conference*, San Francisco, California, October 27-29, 1964, (this issue).
  4. KRULL, F. N., and FOOTE, J. E., "A Line Scanning System Controlled from an On-Line Console," *Proceedings of the Fall Joint Computer Conference*, San Francisco, California, October 27-29, 1964, (this issue).
  5. SUTHERLAND, I. E., "Sketchpad, A Man-Machine Communication System," *Proceedings of the Spring Joint Computer Conference*, Detroit, Michigan, May 21-23, 1963.
  6. PREISS, R. J., "Design Automation Survey—A Report to the AIEE Membership," AIEE Winter General Meeting, New York, New York, January 29-February 3, 1961.

# OPERATIONAL SOFTWARE IN A DISK ORIENTED SYSTEM

*M. Phyllis Cole, Philip H. Dorn and C. Richard Lewis  
Research Laboratories, General Motors Corporation, Warren, Michigan*

## 1.0 INTRODUCTION

This is one of a series of papers which describes the General Motors Research Laboratories DAC-I (Design Augmented by Computers) System.<sup>(1, 2, 4, 5)</sup> For a summary of the overall system objectives and organization, the reader is referred in particular to the paper "A Laboratory for the Study of Graphical Man-Machine Communication" by Edwin L. Jacks.<sup>1</sup>

In using the DAC-I system, the man at the console wants to perform the following types of tasks in solving his problems—

1. Introduce data rapidly and accurately to the computer.
2. Operate on this data.
3. Observe the results of these operations and have the ability to modify them while still in the on-line environment.
4. File the original data and final results for future references.

To accomplish these tasks requires the interplay of the 7960 Special Image Processing System,\* the 7094 Computer Complex, and the man.

It is the purpose of this paper to discuss the systems software developed in support of this interplay.

---

\* The IBM 7960 Special Image Processing System, consisting of a graphic console, an image processor, and a modified data channel, was designed and built by IBM to specifications provided by GM.<sup>2</sup>

Requirements of this software system include:

1. Establishing efficient storage and retrieval methods for handling large numbers of data arrays and subroutines.
2. Creating an environment within the computer which would allow subroutines and associated data to be brought into memory, processed, overlaid, filed, etc. based on the operational demands made by the man at the console.

A disk oriented software system (hereafter referred to as the D-System) has been implemented in order to provide the basic software support. The objectives of the D-System are to provide compiler level accessibility to the new hardware devices and minimize the impact of disk usage on the general programmer. The D-System provides all of the necessary subroutines required in using the disk for data storage and retrieval, for scratch space during intermediate computation, and for the loading of subroutines based upon program needs. The D-System is accessed through a batch monitor system in order to generate subroutines and data for storage on disk. Program execution may be accomplished through the batch monitor or the on-line console.

Establishing an operational computer environment also required the development of compilers to satisfy needs for both the object and system codes which were to be written. The NOMAD compiler was used extensively

for central processing unit (CPU) codes and the MAYBE compiler was developed for 7960 data channel codes. Processors for disk book-keeping and a complete system for loading and overlaying subroutines in memory were also required.

This paper considers first the organization and maintenance of a disk file and then describes an operating system based on a disk with both system and execution time features detailed. A brief discussion of the on-line D-System operation is included. Finally, certain conclusions are drawn based on experience with a disk-oriented system.

## 2.0 THE ORGANIZATION AND MAINTENANCE OF A DISK FILE

### 2.1 Introduction

During 1961, a small disk file was attached to the computer permitting experiments in disk file usage. Although the file was both small and slow, its capacities were sufficient to allow meaningful simulation of the ultimate system. When the final D-System (a disk oriented software system) was being designed, certain design principles learned during simulation were applied.

It was ascertained that when planning storage allocation for a random access device, the following design criteria should be followed:

1. An absolute integrity must be maintained between the data to be stored and the programs which deal with the data.
2. The programming system shall not impose restrictions on the quantities of data retrieved from or stored on disk.
3. The system must be free of fixed locations on the disk file. The optimum is to provide one fixed track location and reference all other locations from this one.
4. The design of object time programs should not be concerned with the manner in which data is physically stored on disk.
5. The number and time duration of disk arm movements to reach a particular piece of information must be kept to a minimum even at the cost of inconveniences to the operating system.

6. Efficient disk back-up procedures are necessary.

In planning the storage allocation, four different types of storage areas were noted. These were: first, processor areas where subfunctions of the operating system reside; second, subroutine areas where object time subroutines are permanently stored; third, data areas; and fourth, scratch and temporary working areas. Each of these areas is handled in a unique manner and the techniques therein involved will be discussed below.

### 2.2 Storage of System Processors

A processor is a logical portion of the operating system which may be called upon to perform a unique function. Examples of processors are the compilers, assembler and a processor which files subroutines on disk.

Processors\* are stored in absolute form. Filing is handled by a separate non-system program which blocks relocatable subroutines into an absolute package. During the filing process, a dictionary is developed which contains the following information for each processor:

1. processor name
2. disk location
3. length of processor
4. processor entry point
5. processor loading location

Since both processor storage and retrieval is by absolute blocks, it may be seen that speed in loading is a feature of the processors. The system (using a cylinder reading technique) loads each processor as an absolute block into the location specified in the dictionary. Control is transferred to the entry point as specified by the processor dictionary. Since no system processor exceeds the capacity of a disk cylinder, no provision has been made for handling overflow to adjacent cylinders.

### 2.3 Storage of Subroutines

The disk area reserved for subroutine storage is divided into logical areas with each

\* All processors are stored on disk except for the basic compiler and assembler. Although there is no logical reason why they could not have been placed on the disk, as a convenience factor they were left on the System Master Tape.

uniquely named area assigned on a project basis. Each logical area consists of a block of physically contiguous tracks bounded for minimum access time.

A separate dictionary is maintained for each disk area. Each dictionary entry contains the following information for each subroutine within that area:

1. subroutine name
2. disk address of the subroutine
3. length of the subroutine
4. date the subroutine was filed on disk

Subroutines are stored on disk in a blocked, relocatable binary form. Output from compilers and the assembler is post-processed into blocked form and then filed in a two record per track format. Each record contains a pointer to the following record except the last record which has a null pointer. When subroutines are being retrieved, the pointer becomes a command to the loading routine to access the next record.

Allocation of a particular location to a subroutine is made from a subroutine space assignment table. This table contains the following information for each disk area:

1. area name
2. location of area dictionary
3. next available dictionary track
4. first and next available tracks in the area
5. number of available records in the area
6. next available record in the area.

In addition, the subroutine assignment table contains the location of a track "pool" to be used by an area whose basic space allocation is exhausted. Should this occur, subroutines enter the "pool". The area limits may be enlarged during edit time\*. Subroutine areas and their respective indexes are located by rereferencing this table; should it become necessary to redefine any or all areas, only the assignment table need be changed. The system refers to track assignments through the table; references to absolute disk locations are not permitted.

\* The edit function is explained in Section 2.6 entitled, "Disk File Maintenance Procedures".

#### 2.4 Permanent Data Storage

All permanent data is stored in uniquely numbered files assigned on a project basis. Each file may contain four different data types where a data type is defined as one of the four different record sizes maintained on the disk. The types are:

- type 1: 25 word record—stored 15 records per track
- type 2: 111 word record—stored 4 records per track
- type 3: 229 word record—stored 2 records per track
- type 4: 465 word record—stored 1 record per track

To establish a data file, the user must indicate how many records of each type his file will contain\*.

A directory table is maintained for each data file permanently stored on disk. The directory tables are stored on disk in numeric sequence beginning with file number 1. The table contains information as to whether this particular file number has been assigned, and if assigned, where the first track of each data type may be located. The location of the first file directory table is maintained in memory so the operating system may access the directories quickly. Given a file number and the number of directories per track, it is simple arithmetic to compute the disk address for a file directory which in turn points to the data file.

Disk space allocation for permanent data is made from a table stored on disk. This table has an entry for each group of uniquely formatted tracks.\*\* A group of tracks is called an area and the entry for each area contains:

1. the first track in the area
2. the currently available track
3. the next available record on the currently used track

\* Space allocations, if not sufficient, may be changed after the file has been assigned space. This process is not, however, performed automatically.

\*\* The IBM 1301 requires a strict formatting of any given cylinder for the size of the records to be maintained on the tracks within that cylinder. The file can be reformatted under program control but this procedure is not generally available to D-System users.

4. number of available tracks in the area
5. number of available records in the area

System subroutines have been provided to store and retrieve permanent data. In general, data files are referred to at execution time as record N of type M.

### 2.5 *Scratch Data Storage*

A scratch file is available to programmers for temporary storage of data during execution. To use the scratch area, a request is made for n tracks of 465 words to a system subroutine. Assignment is on a "last-in, first-out" basis. If the space is available, the system returns a key word containing the first and last track of the area assigned. Any subsequent storage and retrieval is by record number with the key being used as a base address for computation of the physical track referenced.

When the programmer is finished, he executes a "closing" subroutine and the tracks which were made available to him will be returned for future assignment.

### 2.6 *Disk File Maintenance Procedures*

Because the disk file may have occasional mechanical, hydraulic or electronic failures, a back-up procedure was developed to insure the preservation of the permanently resident information. Since both data and subroutines change from day to day, a working procedure was developed which consists of writing a disk save tape(s) daily. These areas are the only disk areas saved. Disk back-up tape(s) are retained for several days before being released.

Both subroutine and data areas are periodically edited by separate programs. Editing is a clean-up procedure; information is physically moved to designated areas and space made available by deletions is returned for reassignment.

Subroutine editing is basically the process of constructing a new area dictionary by omitting entries for deleted subroutines. The subroutines are taken off disk and placed on a scratch tape for temporary holding. At this point in the edit process any change to the boundaries of an area may be made. The subroutines are then assigned new locations

within the area, the area dictionary is updated and the subroutines and dictionary written back onto the disk. At completion of the edit, all subroutines within an area are sequentially assigned and remaining space is available for future changes.

The data editing process is basically similar to the subroutine edit although the data edit program has the additional attribute of being able to operate upon any one type of data (or all types if desired) in a given machine run.

Data and subroutine restore programs reverse the save procedure and reload the disk from the last save tape(s). This procedure normally is used only in the event of major machine failures. A "cold start" procedure in the event of complete catastrophe includes subroutine and data restoration as well as reformatting, rewriting of home addresses, reloading processors and rewriting certain system control information.

## 3.0 THE D-SYSTEM

### 3.1 *Introduction*

The D-System is a switchbox which examines the input stream of requests, loads the appropriate processor and links processors by passing parameters. A pointer is positioned by the D-System so that a processor may know the current position in the input stream. The D-System is also responsible for collecting timing information to be used for installation billing operations.

The basic functional breakdown of the D-System is outlined in Figure 1\*:

Each of these three areas—input, bookkeeping, and execution processing—is described in succeeding sections.

### 3.2 *Input Processing*

Input processing is that set of system functions which results in object programs being

\* The execution of the D-System as well as other systems operated at GM Research is controlled by the General Monitor Program. The General Monitor decides which system to call in, processes accounting information, signs tapes on and off the machine, and provides a means for operator communication to the programming systems.

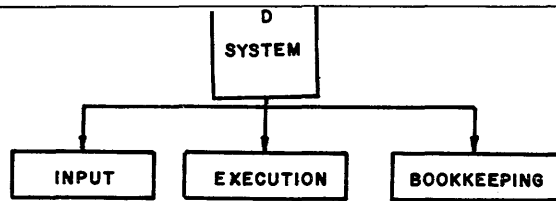


Figure 1. D-System Functions.

placed on the disk. Object programs are placed in one of two disk areas—either checkout or scratch, depending on the programmer's request. Subroutines filed in scratch disappear at the end of the job while those filed in checkout remain on disk for approximately one month.

The input processors are shown diagrammatically in Figure 2.

### 3.2.1 The NOMAD Compiler

NOMAD is an algebraic compiler adapted from the MAD\* language to meet the special needs of this installation. It is a high speed compiler which permits a wide latitude of generality in expressions. Since MAD was implemented in a highly modular fashion, there was little difficulty redefining code generation sequences, adding new operators and enlarging the statement repertoire. Although MAD is of the ALGOL '58 family of languages, it has been modified considerably since its original design.

The NOMAD dialect developed at GM Research Laboratories has four basic areas of difference from MAD:

1. additional operators
2. new variable types
3. new relocation scheme
4. real-time statements

Thirteen new operators were added to the language to permit a full set of logical operations. Of special note are three bit detection operators that seek the first, last, and number of "1" bits in a variable, and a set of address/

\* The Michigan Algorithmic Decoder (MAD) is an algebraic compiler based upon ALGOL '58. It was originally developed by Arden, Galler, and Graham of the University of Michigan Computing Center for use on the IBM 704. The language and compiler have been updated through a series of revisions for the 709 and 7090. The NOMAD compiler springs from the earliest 7090 version circa 1961.

decrement packing and unpacking operators. These operators were especially designed to permit coding of system subroutines in the NOMAD language.

A new class of variables, the *GLOBAL VARIABLE*, was introduced. A global variable is defined as a variable, single or array, to which storage is not assigned at compile time. Unlike the COMMON variables in a FORTRAN program, global variables not used in a program do not have to be declared merely for the information of the loader. No special ordering of global variable declarations is necessary.

For each global variable used in a NOMAD program, the compiler generates an entry into a list attached to the program card information \*\*. Each occurrence of a global variable results in special relocation bits being produced to indicate that one or more fields of this instruction are global. Additional bits indicate the slot in the list to which the global variable is assigned. If the global variable is subscripted, the numeric subscript is placed in the field normally assigned to the address.

The relocation scheme is based on a variable number of bits assigned to each instruction type (e.g., absolute, relocatable address with absolute decrement, etc.). The most frequently used class of instruction is described by one bit, the second most frequent by two bits, the next most frequent by three bits, etc. The first "0" bit reached acts as a delimiter. Comparisons made to other schemes have shown non-trivial operating efficiencies as well as considerable core and disk space savings.

The real time statements within the NOMAD compiler seek to acknowledge the presence of man in the program loop. Since a console for display of graphical information is part of the hardware configuration, system users output data onto the console display screen rather than the system output tape. Because the

\*\* A NOMAD subroutine contains on its program card(s) data relating to the global variable(s), the entry point(s), the program length and the number of program cards. When the subroutine is actually filed on disk, additional information is added during the blocking process such as the program's checksum and the length of the transfer vector.

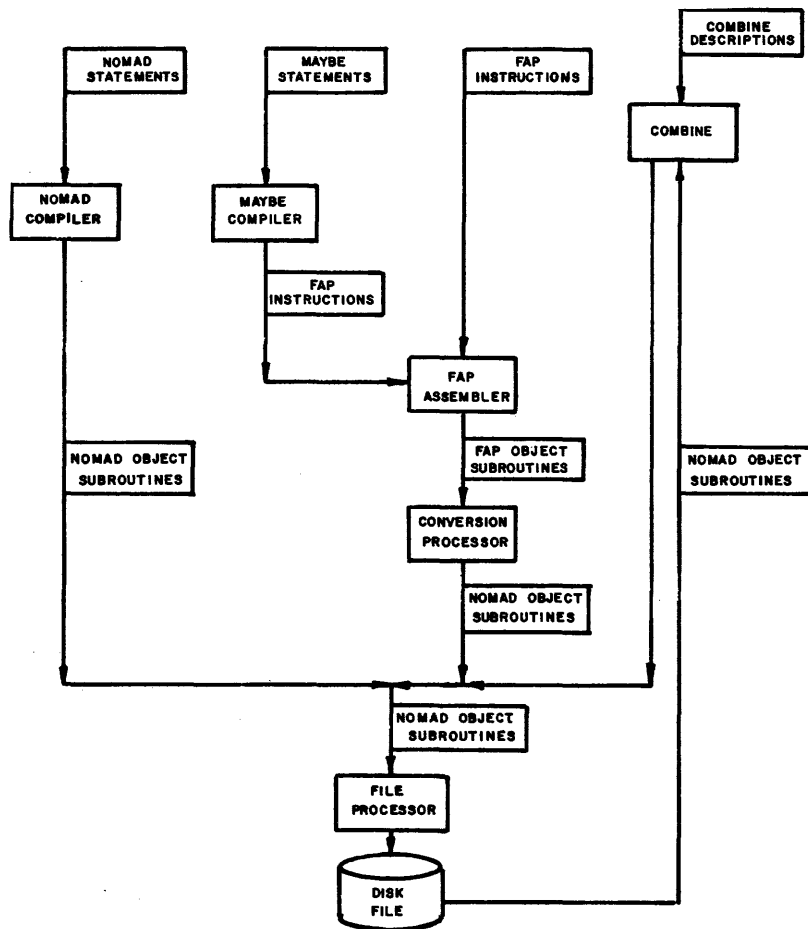


Figure 2. D-System Input Processors.

NOMAD statement for writing tape output has the form:

PRINT FORMAT F, List

where F is the name of a format and List is the data to be written, it is appropriate and logical to give the statement:

DISPLAY FORMAT F, List

for displaying information on a screen. A similar statement exists for production of hard copy output through the recording CRT:

RECORD FORMAT F, List

where F and List have the same meanings as above.

These statements represent steps toward development of a language which recognizes parallel processing in a large scale computer. Since the IBM 7094 has the capability to drive multiple channels in parallel, it is essential to

permit the direct use of the full capabilities of the machine at the source language level.

Other features of NOMAD which contribute to its selection for use in the D-System are more conventional but still important. NOMAD permits a completely general subscription expression, a generalized iteration statement, multiple entries and exits from subroutines, the manipulation of statement labels, the use of internal procedures, nested conditional statements and the use of certain elementary push down list facilities.

### 3.2.2 The MAYBE Compiler

The I/O devices of the 7960 Image Processing System are connected to the central computer through a modified 7909 data channel. The data channel is capable of performing simple iteration loops, full and partial word



substitution and byte testing as well as driving I/O devices. It cannot, however, add, subtract, shift or mask. The data channel can be viewed as a special purpose processor attached to the 7094 memory and capable of running in parallel with the 7094 central processing unit. It was necessary that means be provided to program this special purpose computer in a higher level language.

The MAYBE compiler was designed and implemented to provide the instructions, commands and orders for operation of the data channel and 7960 I/O devices. In addition, MAYBE automatically produces the necessary system linkages to process the data channel interrupts and central computer traps. Figure 3 shows the relationships among the computer, the data channel and the 7960 system.

Each subroutine generated by MAYBE consists of prologue instructions and a main body of data channel commands and orders. Stripped of its frills, the MAYBE compiler is a macro-generator which feeds symbolic input to the standard assembly program. MAYBE was coded in NOMAD and utilizes standard system I/O routines.

The MAYBE language includes approximately 75 declarations and statements divided into the following classes:

1. storage allocation declarations
2. replacement statements
3. iteration statements
4. control and linking statements
5. device manipulation statements

MAYBE declarations are essentially the same ones found in the NOMAD compiler and provide a means to utilize local and global storage for data variables. The replacement

statements allow substitution of data variables even when they are positioned in non-standard fields\*. Iteration statements permit loops within the data channel and testing of the loop control index. Control statements permit the transfer of control within MAYBE subroutines or externally to other MAYBE or NOMAD subroutines. Device manipulation statements permit the starting, continuance and stopping of I/O devices attached to the 7960 system.

Linkages generated by the MAYBE compiler permit MAYBE subroutines to interrupt operations and transfer control to NOMAD (main frame) subroutines. In this way, a nesting of alternate MAYBE and NOMAD subroutines is achieved. The maximum depth of this nesting operation is the programmer's ability to remember where he is; there is no system specified limit. At each step, the data channel and the main frame will be jointly interrupted and their respective status saved. Thus, no matter how deep the nesting, the machine status will be restored upon return to each higher level.

MAYBE is essentially a compiler for use of system programmers. Most users operate devices through NOMAD statements (such as DISPLAY or RECORD FORMAT). Subroutines coded in MAYBE provide the I/O commands to drive the requested device.

### 3.2.3 The Combine Processor

The COMBINE processor permits the D-System user to reduce a set of NOMAD object level subroutines into one physical subroutine. This installation's programming standards emphasize subroutinizing as a checkout technique. Advantages may be gained by using a combined package of subroutines since the number of disk accesses at load time is sharply reduced as are the bookkeeping tasks during subroutine execution.

In addition to producing one relocatable subroutine from many, the COMBINE processor has the following features:

1. Global variables used for communication between the set of subroutines to be com-

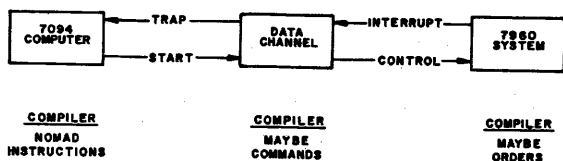


Figure 3. 7909 Relationships.

\* Instruction fields for 7960 commands and orders vary from the address and decrement fields normal to 7094 instructions. Provisions had to be made to handle fields as small as three bits.

bined may be drawn inside the package and assigned local storage or left as external global variables at the user's option.

2. A merged external transfer vector is produced for the combined package. Internal connections between the set of subroutines involved are handled through an internal transfer vector. References between one subroutine of the set and another within the set do not produce an external transfer vector entry.
3. Any entry point or set of entry points associated with the subroutines of the set may be retained, discarded or renamed at the user's option. A discarded entry point has no external meaning and does not exist outside the scope of the set.

Since all D-System processors have themselves been through COMBINE, there remains only the single merged transfer vector to be set during processor loading.

#### 3.2.4 *The Assembly Program*

A standard 7094 assembly program, FAP<sup>3</sup>, has been built into the D-System. Although it is available for general use, programmers are discouraged from using it except for highly special cases where extreme speed or efficiency is required, or for special purpose utility routines (such as number conversion and input/output). The bulk of the D-System and the processors, as well as over 95% of the applications programs, are coded in NOMAD.

#### 3.3 *The Bookkeeping Processors*

The bookkeeping processors exist for the purpose of allowing the programmers to add, replace, and delete subroutines within the disk area to which their project has been assigned. Since the system permits retention of many subroutines with the same name but allows only one version to be in any area at any time, users' are responsible for having the right version in the right place at the right time.

The following bookkeeping processors are available in the D-System:

1. Move
2. Delete

#### 3. Move and Delete

##### 4. List Areas

Subroutines are initially filed in either checkout or scratch. Programmers generally leave new subroutines in checkout until debugging is completed. A facility exists for overriding normal calls for this particular subroutine \* so that the new version may be debugged as either an isolated unit or within the total operating environment. Subroutines may remain in checkout for one month after they are compiled. After that date, if not moved from checkout, they will be discarded at edit time. MOVE is the programmer's means of sending a subroutine to a permanent area.

DELETE allows the programmer to take a version of a subroutine off the disk when it has outlived its usefulness.

MOVE AND DELETE performs the dual function of moving a subroutine to a new area and deleting it from the old area.

All these processing functions do not physically move the subroutine involved but merely make entries (or delete entries) in the subroutine dictionary for the area under consideration. Actual moving is done at edit time. Should a programmer move a subroutine to an area which already has a same named subroutine on file, the new version will automatically override the old version.

LIST AREAS permits a programmer to obtain a full listing of all subroutines in a specified area(s) along with their lengths and filing date. The physical disk address is also printed out at this time \*\*.

#### 3.4 *The Execution Phase*

D-System exception philosophy diverges from execution logic in most systems. Taking a standard FORTRAN batch processing moni-

\* This is the USE processor described in the section on subroutine execution. (Section 3.4.2)

\*\* While printing the actual location of a subroutine may seem to violate the system criteria of not allowing a user to know the physical location of anything on the disk, it actually is virtually useless information since the physical layout of an area will change at each edit pass. Those responsible for the correct operation of the hardware need this information occasionally after a machine failure.

tor for comparison, the following major differences may be pointed out:

<i>FORTRAN</i> <i>SYSTEM</i>	<i>D-SYSTEM</i>
Subroutine loaded from tape	Subroutines loaded from disk
Core loads	One subroutine loaded at a time
Ping-pong and overlay	Continuously changing core configuration
COMMON variables loader assigned	Global variables not assigned until needed

### 3.4.1 *Dynamic Loading*

While grouping subroutines in a core load is reasonably efficient when using magnetic tape as an input medium, this procedure becomes wasteful when a random access on-line device is available. Subroutines can be loaded individually from disk *as they are needed* without the burden of tape spacing and rewind time. Variables placed in COMMON in a FORTRAN system are assigned locations at compile time. D-System global variables are assigned locations at execute time when the actual core availability determines the location assigned. This floating quality of a global variable is essential to maintaining a flexible core arrangement.

Since multiple versions of any subroutine may exist, the D-System loader must receive a specification of the area in which the subroutine is located. When a global variable is first referenced by a subroutine, the dimension of the variable is needed to assign space. A D-System programmer defines the scope of his program by supplying area information for his subroutines and dimensions for his global variables. The D-System performs the function of an interface between the programmer and the loader to initiate execution. The system accepts the name and area of one subroutine and the dimensions of a set of global variables and passes this information to the loader as "starter" parameters. Execution commences with one "starter" subroutine which may be located anywhere on disk. The disk areas of other subroutines and the dimensions of other global variables are dynamically passed to the loader as execution proceeds.

When a program is in execution, the status of individual subroutines may vary as indicated in Figure 4. The status of a given subroutine is one of the following:

1. Undefined—This status is included for completeness and indicates the basic state of all subroutines on the disk.
2. Not In—The subroutine has been located on disk and may be loaded as needed.
3. Active—Whenever a subroutine is executed for the first time, it is loaded from disk and becomes "Active".
4. Inactive—A subroutine declared "Inactive" by the programmer remains in core in anticipation of later use. However, if additional core storage is required, it is returned to "Not In" status making the core location which it occupied available.

When subroutines have served their purpose, they are declared "Out" and return to undefined status.

The status of global variables may vary as shown in Figure 5. A global variable is assigned storage when the first subroutine which references it is loaded from disk. The storage is released when all subroutines referencing the global variables are declared "Out".

Availability of core space is maintained by the loader as subroutines and global variables change status. Therefore, the functions of the loader may be summarized as follows:

1. Change status of subroutines as directed by the executing program.

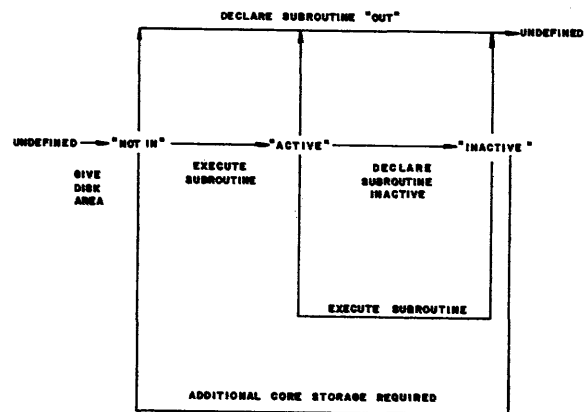


Figure 4. Subroutine Status.

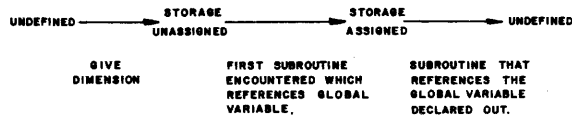


Figure 5. Global Variable Status.

2. Change status of global variables as directed by the executing program.
3. Maintain storage availability.

### 3.4.2 *Dynamic Subroutine Definition*

A D-System feature allows the checkout of a new version of a subroutine without disturbing the production execution of a previous version of that subroutine. This is done by placing the newer version in the scratch or checkout area. The production programs anticipate finding the subroutine in a permanent storage area. However, the loader may override their instructions and find the subroutine from scratch or checkout. This is accomplished by means of the USE processor which transmits a user's request to the loader. All requests to load this subroutine will be trapped and the version from scratch or checkout loaded instead of the production version.

### 3.4.3 *Error Procedures*

The D-System error package operates in one of two modes. The programmer may elect to monitor all software error conditions and take corrective action, or he may allow the error package to perform standard procedures. Four classes of error conditions are monitored in the error package:

1. arithmetic underflow/overflow in floating point operations
2. illegal subroutine parameters
3. I/O format and data errors
4. loading errors

The standard action of the error package is to reset and continue on underflow/overflow errors and to emit a diagnostic and a dump of the programmer's subroutine and data areas in all other situations\*.

\* The system is not normally dumped.

When the programmer is monitoring the error conditions, he may reenter the error package to obtain the following actions:

1. reset and continue on underflow/overflow
2. reset and continue on loading errors caused by insufficient core storage availability. Prior to re-entering the error package, the programmer must make additional core storage available by releasing space taken by either global variables or subroutines. If the same loading error occurs a second time, the execution will be terminated by a dump.
3. job termination with a memory dump of either the programmer's core area or all of core.

Since core is constantly changing during a D-System run, a full or selective core dump is always accompanied by a core map. The map contains the following information:

1. the name, location, length and status of all defined global variables.
2. the name, location, entry points, length and status of all defined subroutines.
3. the location and length of remaining available core space.

### 3.4.4 *Data Handling*

Data that is to be stored permanently is always in array form where word one of the array is the data name and word two contains the length of the array.

The data name is encoded to contain a file number, a key to which record within that file is being referenced and a record revision number. Upon the first reference to a file, the directory table\*\* is brought into memory and is used with the data name to compute the track and record address for the data request.

Should a data array exceed the record length of the indicated data type, an additional record is assigned and chained to the base record indicated by the data name.

At the beginning of each record is a control word, CW, containing three flags defined as follows:

flag 1 = 1 if revision has been filed, = 0 otherwise

\*\* The directory table is described in section 2.4 dealing with data storage assignment.

flag 2 = 1 if data has been filed, = 0 otherwise  
 flag 3 = 1 if data has been deleted, = 0 otherwise

An example will best illustrate the control and chaining techniques used. Assume a data array, TORQ, 129 words long. Assume also the record revision number is 0. The first two words of TORQ are as follows:

Assume the directory table for this file to be:

The encoded name in TORQ(0) references the fifth record of data type 2 in file #19. Data type 2 (stored 111 words per record, 4 records per track) begins on track 1280 for this file. Therefore, TORQ(0) points to record 1 of track 1281. Since the length of the array is greater than 111 words, the first record will be chained, as shown on the next page, to an additional record taken from a pool of available records.

If it is desired to store a revised version of the array TORQ at the same time retaining the original data, a slightly different procedure is followed. First, the data stored on track 1281, record 1, is moved and the original record of 106 words on track 1281 is chained to the new location. The remaining 23 words on pool track 1682, record 2, do not move. The revised data is stored in a newly assigned record and another chain in record 1, track 1281 is set to point to the location of the revised data.

The name given in TORQ(0) identifies revision number 1 of the data. As before, record 1 of track 1281 is to be referenced.

Under this addressing scheme, the programmer need only give the name of a data array to retrieve it—the length of the array is stored in the array itself and will govern the number of words transmitted from the disk.

4.0 ON-LINE EXECUTION

In addition to execution within the batch monitor system, a D-system program can be executed on-line from the graphic console. In this operation the batch monitor is restricted to half core, and the other half of core is made available to on-line operation. A simplified

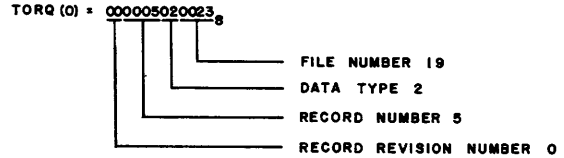


Figure 6. Data Name Encoding.

system exists for on-line execution to act as an interface between the man at the console and a slightly modified D-system loader. The man at the console indicates to the system the name and disk area of the subroutine which initiates execution.

While there is no difference in basic philosophy between batch monitor and on-line execution, implementation is quite different since tapes are not used directly when operating on-line. Graphical output is used whenever possible to replace normal output tape functions. When printed output is required, as in producing core dumps, the information is placed on the disk and inserted in the output stream of the batch monitor system between jobs. The on-line program has the ability to insert jobs into the batch monitor system via the disk. A circular file is used to pass data to these jobs. The program in execution must also use the disk rather than tape for scratch space.

5.0 CONCLUSIONS

Our experience indicates it is feasible to operate from a disk and gain rapid access to large amounts of information, thus attaining considerable on-line capability. To obtain this on-line capability, users must pay a penalty in several areas. Core memory space must be reserved for an in-memory loading and relocation routine. Machine time must be granted for disk bookkeeping and editing functions.

DISK TABLE FOR FILE - 19

DATA TYPE	BEGINNING TRACK	NUMBER OF RECORDS RESERVED
1	0080	25
2	1280	10
3	2880	12
4	6800	8

Figure 7. File Directory Table.

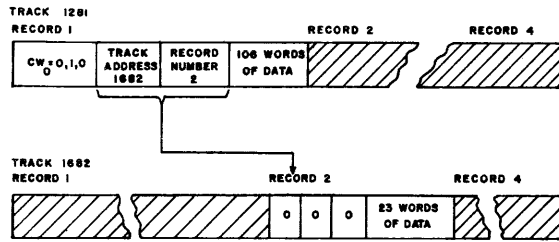


Figure 8. Data Track Layout.

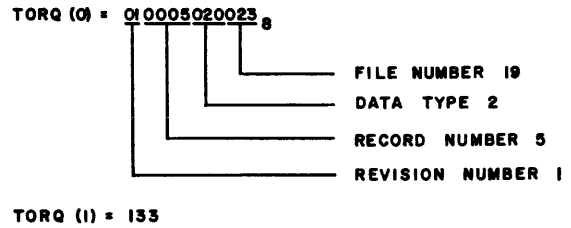


Figure 9. Encoded Revision Name.

Compatibility with other installations is completely lost.

In return for this investment, the system allows access to an enormous library of routines without having to deal with an object level deck. Large quantities of data are stored on-line and may be added to, modified, deleted or used with no difficulty. Both sub-routines and data are always available; run preparation time is sharply reduced.

REFERENCES

1. JACKS, E. L., "A Laboratory for the Study of Graphical Man-Machine Communication," *Proceedings Fall Joint Computer Conference*, San Francisco, California, Oct., 1964, (this volume).
2. HARGREAVES, B., JOYCE, J. D., COLE, G. L., FOSS, E. D., GRAY, R. G., SHARP, E. M., SIPPEL, R. J., SPELLMAN, T. M., THORPE,

R. A., "Image Processing Hardware for a Man-Machine Graphic Communication System," *Proceedings Fall Joint Computer Conference*, San Francisco, California, Oct., 1964, (this volume).

3. IBM 7090/7094 Programming Systems FORTRAN II Assembly Program (FAP) IBM Form C28-6235-2.
4. KRULL, F. N., FOOTE, J. E., "A Line Scanning System Controlled From an On-Line Console," *Proceedings Fall Joint Computer Conference*, San Francisco, California, Oct. 1964, (this volume).
5. ALLEN, T. R., FOOTE, J. E., "Input/Output Software Capability For a Man-Machine Communication and Image Processing System," *Proceedings Fall Joint Computer Conference*, San Francisco, California, Oct. 1964, (this volume).

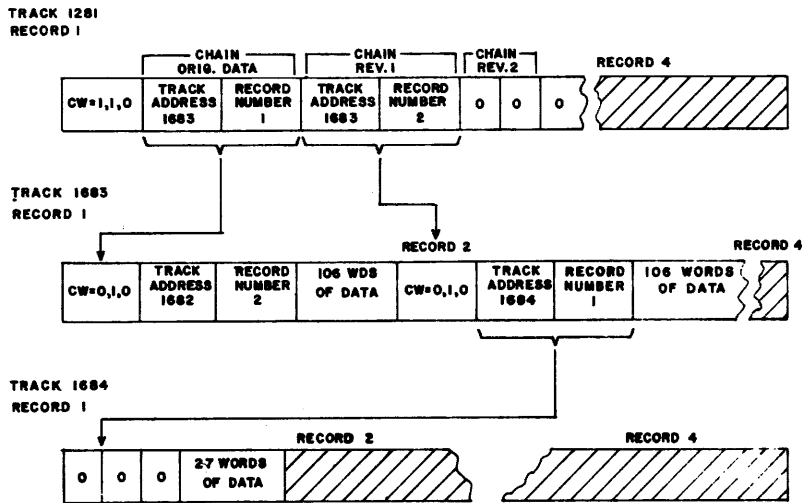


Figure 10. Revision Track Layout.

# IMAGE PROCESSING HARDWARE FOR A MAN-MACHINE GRAPHICAL COMMUNICATION SYSTEM

*Barrett Hargreaves, John D. Joyce and George L. Cole  
Research Laboratories, General Motors Corporation,  
Warren, Michigan*

*and*

*Ernest D. Foss, Richard G. Gray, Elmer M. Sharp, Robert J. Sippel, Thomas M. Spellman and  
Robert A. Thorpe  
Data Systems Division, International Business Machines Corporation,  
Kingston, New York*

## INTRODUCTION

The General Motors Research Laboratories (GMR) obtained the IBM 7960 Special Image Processing System in order to provide a laboratory for the study of graphic data processing and related man-machine communication problems. The IBM 7960, designed and built by the IBM Data Systems Division to specifications provided by GMR, consists of:

- a) A graphic console which includes a display tube, control buttons and lights, a card reader, an alphanumeric keyboard and a position indicating pencil.
- b) An image processor which permits computer-controlled scanning of film images and computer-controlled recording on 35mm film.

This paper is divided into two parts. Part I, written by the IBM authors, describes the design of the special image processing components and the integration of these components into the system. The main functional requirements for these components are computer compatible image generation speeds and high image quality. The design shows how the diverse technol-

ogies of analog circuits, cathode ray tubes, optics and film processing were successfully combined to provide a new type of image processing system.

Part II, written by the GM authors, is a review of GM's experience with the hardware as a component in the General Motors Research Laboratories' DAC-I (Design Augmented by Computers) System. Other papers in this series (1, 2, 3, 4) cover various aspects of this system.

The hardware is a working model of auxiliary computer equipment for designers. Previously, experiments have been conducted on individual components of equipment such as man-machine consoles or light pens, or image digitizers or image recorders or plotters. These experiments have pointed out possibilities for future developments in computer-aided design equipment. Now, all the necessary hardware components have been developed and put to use as a complete DAC-I hardware system.

In addition to having demonstrated the capabilities of this equipment for man-machine relationships in design experiments during the

past one and one-half years, GM Research has created and used extensively new types of programs that both improve the effectiveness of the hardware by calibration and evaluate and display the status of the hardware for the user or maintenance engineer. One test program is described briefly as an example.

## PART I—ENGINEERING DESIGN

### FUNCTIONAL DESCRIPTION

A block diagram of the 7960 Special Image Processing System is shown in Figure 1-1. The attachment of this system to the central processing complex is through data-channel logic. In relation to the central processing system, the Special Image Processing System appears almost identical with any of the other data channels which may be attached to the system. In fact, the special data channel is an IBM 7909 Data Channel, modified slightly to make it better suited to the particular tempo of data flow that exists with this system.

The 7960 system comprises three basic units. The display adapter unit performs such functions as control of the basic system, control unit selection, and digital-to-analog conversion.

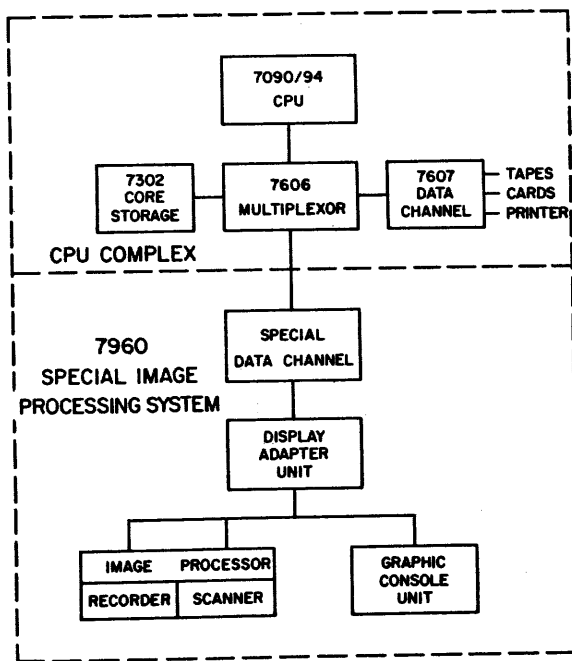


Figure 1-1. System Block Diagram.

The image processor provides for the input and output of data in graphic form. The unit contains a CRT photo recorder, projectors, CRT photo scanner, an input camera (for photographing drawings and documents), and rapid film-processing equipment. (Figures 1-2 and 1-3).

The graphic console is the primary system control point (Figure 1-4). It contains a CRT display, graphic pencil input, alphanumeric input from keys and punched cards, special function inputs from keys, and status and program status indicators. Information may be entered or modified in the system through the use of the graphic pencil, the program control keys, and alpha-numeric keys, or the card input. The results of calculations are displayed on the cathode ray tube or indicated on the status lights. Detailed descriptions of each of these units follow.

### Display Adapter Unit

The display adapter unit controls the transmission of data, unit control information, and unit status information, and the sequencing and synchronizing of the various units in the system. In addition, digital data received from

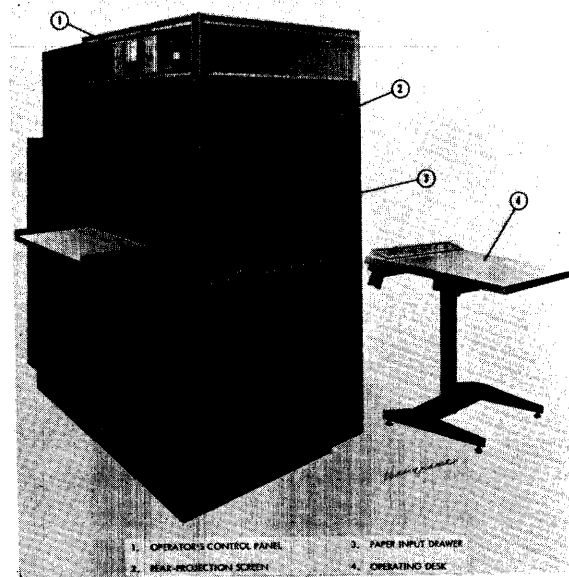


Figure 1-2. Image Processor Unit.



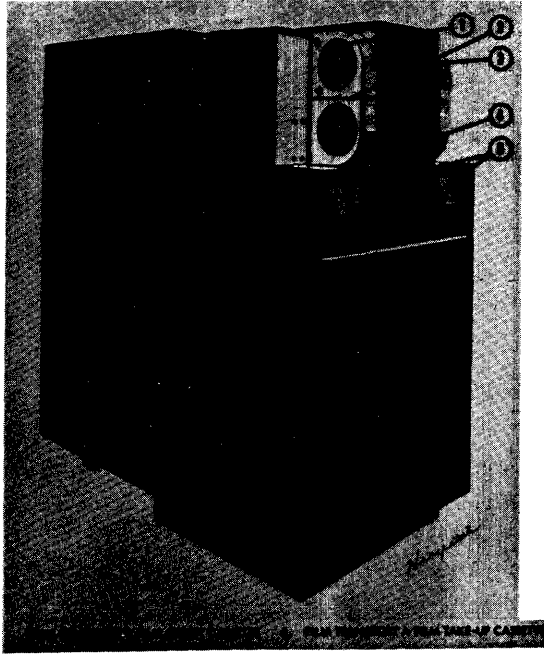


Figure 1-3. Image Processor Unit.

computer storage is formatted for deflection commands for the CRT devices.

The display adapter unit and the data channel recognize five states that may exist. Four standard commands (control, write, read, and sense), perform all data movements in the system. These commands serve to transfer data in 6-bit bytes, which contain the encoded information associated with the operation to be performed. The fifth state, an interrupt signal, is used to notify the data channel when operator action requires a branch in the program.

**Image Processor**

The image processor provides the input and output of data in graphic form. The unit contains two photographic-film transport units which are similar in operation but which differ in the functions that they perform. For convenience, they are designated transport A and transport B. Figure 1-5 is a simplified representation of the film transport and optical system.

**Transport A:**

- a. Exposes film from a high-resolution recording cathode-ray tube.

- b. Exposes film from a paper input station.
- c. Processes the film (develops, fixes, washes, and dries).
- d. Scans processed film for computer input at the read station using a high-resolution scanning CRT.
- e. Projects the processed film from the read station to a 20 x 22-inch rear-projection screen located at the front of the unit.

**Transport B:**

- a. Exposes film from the record CRT.
- b. Processes the exposed film.
- c. Projects processed film.

Both transports can be operated independently and simultaneously, within the limits imposed by the optical and shared data paths for the CRT's. For example, exposing film from the record CRT involves a mirror which directs the image to the selected film transport; therefore, only one film may be exposed at a time.

**Image Input**

The source document for an image-processing design system is normally graphic information on paper. The paper documents will include engineering drawings, sketches, or graphs

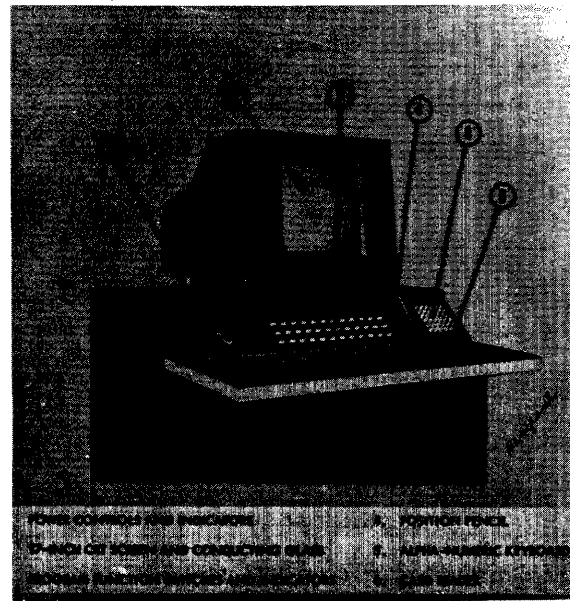


Figure 1-4. Graphic Console Unit.

which must be converted by a digitizing process into a form acceptable to the computer. In the 7960 System, paper documents are photographed on 35mm film and the film is used as the media for the image input process, as shown in Figure 1-5.

Film was chosen as the media to be scanned because of the need of being capable of processing a range of paper sizes and image qualities. The reduction of various document sizes to a standard image size for scanning permits higher scanning speeds and the ability to control image quality.

Paper documents up to 22 inches square are reduced to a 1.2-inch square on sprocketless 35mm film by the paper input camera. To expose the paper it is positioned in the paper-input drawer and held flat by means of a vacuum. An array of eight flood lamps illuminates the paper. The intensity of the illumination is under operator control and exposure can be varied to adjust for differences in image density and contrast on the paper. A paper-input shutter provides a timed exposure.

The use of 35mm silver film as the image input media requires an on-line-computer-con-

trolled, rapid-film processor so that the exposed film can be developed for immediate scanning. A three-station cup application process was chosen to provide an image with uniform density, high-image stability and a resolution comparable to that obtained by hand-processing methods.

The processed film image is digitized under computer control by a flying-spot, CRT scanner (Figure 1-5). A CRT scanner was chosen for the reduction of graphic data to digital computer data because of the CRT scanner's computer-compatible speeds and flexibility. The CRT beam can be scanned over the 1.2-inch square film image area under program control. Light from the CRT passes through the film and is intensity-modulated by the dark and light areas of the image. The modulated light is detected by a photo-multiplier and the amplitude-modulated signal is converted to digital information.

The primary considerations in the design of the scanner were high scanning speed and accuracy. Accuracy includes both the reliability of detected data and the high relative positional stability over the period that the image is being scanned. The CRT beam is moved from point to point over the image by vectors composed of straight-line segments of varying length. This method of beam positioning is called an end-point vector method because, regardless of the length or direction of a vector, only the new end-point must be specified. This results in minimum computer data to control the scanning vector.

The data required to draw one vector is given by the computer in 12 bits for X position and 12 bits for Y position. These digital values are converted to analog voltages which determine the deflection current applied to the deflection yoke of the CRT. The scanning beam can be positioned over 4096 x 4096 addressable positions. An effective increase in positional resolution of the scanner is obtained by the use of a constant-time-scan vector system. In the course of a vector, the light passing through the film intercepts a line or lines of the image contained in the frame. Each line interception is sensed by the PMT and is called a strike. With a constant time vector system, it is possible to divide a vector into time segments which can be related to position. A strike occurring during a par-

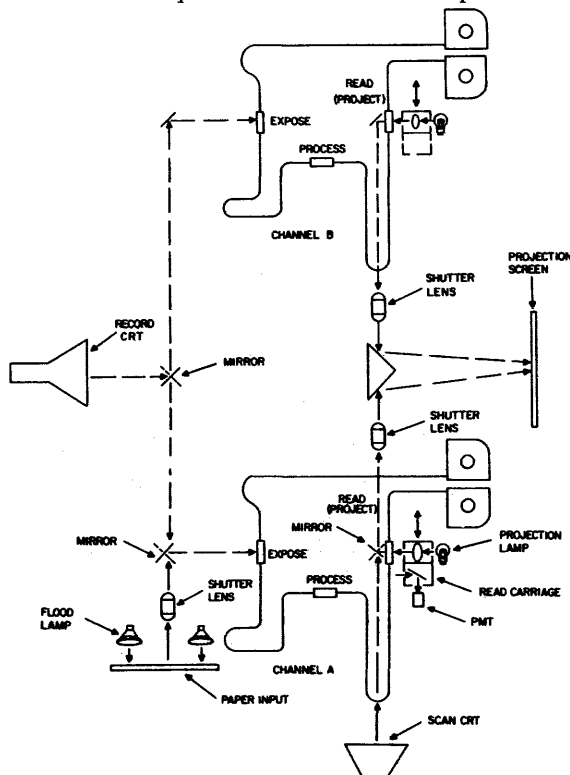


Figure 1-5. Image Processor Schematic.

ticular time segment can be related to the position on the image corresponding to the position of the scanning vector at that time. Scanning response resolution to a fractional part of the vector length can be obtained by this method.

In the scanner, the time required to draw a vector is either 32 or 256 usec. The 32-usec-scan-vector time is used for short vectors; the 256-usec-time for long vectors. To accommodate a wide range of images with varying image density, contrast and line widths, it is necessary to provide program controlled scanner detection sensitivity. The detection threshold of the strikes or hits can be varied by commands so that optimum detection sensitivity can be selected by the program.

#### Image Output

After an image is processed by the computer program and modified and verified by the operator at the graphic console, it is frequently necessary to produce a permanent output document of the processed image. Film is used for the output image media, as for the input media, because of its compatibility with computer speeds, flexibility of use, and high image quality.

CRT recording on silver film permits images to be generated by computer control at computer-data-channel speeds. The CRT used for output image recording is similar to the type used for input image scanning. As in the scanner, a CRT image is formed by vectors drawn on the CRT screen. The vector trace is visible when the beam is unblanked, in order that the beam trace can be recorded on the film. When the beam is blanked, the trace is not visible and the beam can be positioned over the image without exposing the film. An image—for example, a drawing—is divided into straight line segments. Core storage contains the X and Y coordinates of the end point of each vector. Vectors may vary in length from the very short traces required to form a smooth arc to longer traces which are used for straight line segments of the drawing.

The deflection circuits used for CRT recording are the same as those used for CRT scanning and graphic console display. Therefore, two fixed vector times are available for image recording: 32-usec vector time, generally used

for drawing short vectors, and 256 usec generally used for long vectors. Four vector line widths are selectable: basic, 2x basic, 4x basic and 6x basic. A constant vector time system for recording results in a beam velocity that varies with vector length. If not compensated, this would result in varying film exposure and therefore, varying density on the recorded image. To provide even exposure for all vector lengths and vector widths, dynamic intensity control is used. This analog circuit provides continuous compensation for beam speed and line width.

Other analog circuit corrections must be used to obtain a high quality, high resolution, linear image on film. Continuous beam focus compensation is required to maintain a perfectly focused beam over the CRT face. The flat screens of the record CRT and scan CRT require focus compensation that is a maximum when the beam is deflected to the edges of the screen, and follows a parabolic function which gradually diminishes to zero compensation when the beam is at the center of the screen.

The X and Y signals, which cause beam deflection at the record and scan CRT's must be corrected to prevent "pin cushion" effect. This effect is inherent in flat CRT screens and causes the sides of a square to become concave arcs and the overall area to be enlarged. An analog pin cushion corrector is used to modify the deflection current to make the beam position a linear function of the angular position of the beam.

The output image is exposed to film at the expose station of either Transport A or B of the CRT recorder. The film is pulled through the expose station in one-frame increments, and is exposed to an image from the record CRT as for the paper input. The exposed film is accumulated in a storage loop until a sufficient quantity of film is available for rapid processing. A loop of film is maintained in front of the expose station so that the frame can be quickly pulled down by the drive mechanism. The processing of the exposed film is identical to that which occurs after paper input exposure.

After the output image is processed, it may be immediately viewed by the operator by pro-

jecting the film image onto a 20-inch by 22-inch rear projection screen (See Figure 1-5). At the projection station the film may be advanced or backspaced one frame at a time under computer control or advanced or backspaced incrementally under operator control. By utilizing both of the film transports, each with a projection station capable of projecting a film image onto the common projection screen, it is possible to compare two images or to produce 3-dimensional effects on the screen.

The large screen projector permits the operator to study the output image off line from the computer. The image is larger and of higher quality than can be obtained on the graphic console and the image can be studied and compared with drawings or other graphic console images.

#### Graphic Console

The graphic console (See Figure 1-4) provides primary system control. The man-machine-communication components of the graphic console are:

1. A 10-inch-square CRT display surface, and a position-indicating pencil.
2. Thirty-six program-status lights (with a message overlay) and 36 program control keys.
3. An alphanumeric keyboard.
4. A card reader.

The 10-inch-square display surface is a CRT display with a phosphor coating designed to control flicker and improve viewing comfort. As with the scanner and recorder, the display is created by having the computer specify the end points of the vector to be drawn. The dynamic CRT display utilizes a transparent conductive screen with an impressed voltage gradient and a voltage-pickup position pencil to aid in operator modification of the displayed image. Basically, the data read under program control notifies the tracking routine whether the pencil position is to the left or right and above or below a particular (X,Y) position.

Thirty-six program control keys are provided for use in addition to the input pencil. The function of a particular program control key is assigned by the program and can be

changed from program to program. Replaceable overlays (See Figure 1-6) are used to identify the function of each key for each application. The descriptive labels on the overlay can be illuminated by the program-status indicators which are also under program control. An alphanumeric keyboard, consisting of 36 keys arranged in a 6-by-6 key matrix pattern enables the operator to enter data at the console. When the keyboard is operated in conjunction with an upper and lower case switch, alphabetic, numeric and special character codes can be generated. There is, in addition, a manually-fed, card-reader input for the entry of limited amounts of data.

The code generated by either the keyboard or the card reader can be interpreted by the program as desired, giving additional flexibility to these devices.

#### ENGINEERING DESCRIPTION

##### Film Transport Control

A film transport consists of the following units (Figure 1-7):

1. A film supply cassette and takeup cassette.
2. Drive motors, clutches, film guides and other controls that move the film from the supply cassette.
3. An expose station.
4. A process station with its associated processor - applicator - elevating mechanism.
5. A read station for projection or scanning.

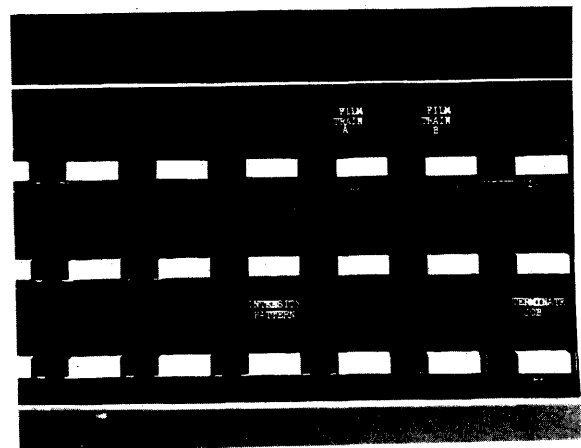


Figure 1-6. View of Graphic Console Overlay.

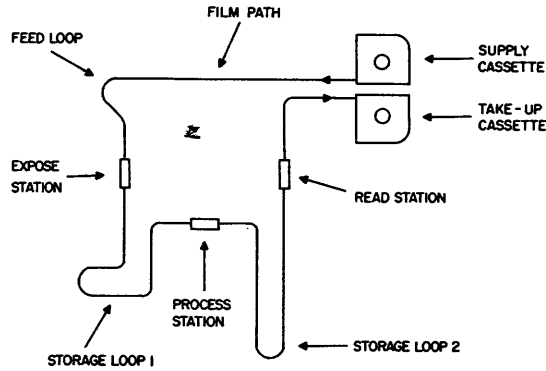


Figure 1-7. Film Transport.

Film is fed from a supply cassette capable of holding 400 feet of 35mm unsprocketed film. The film is moved through the channel to a takeup cassette by means of a friction drive.

Initially, the film motion controls thread the film through the film transport. Independent motions at each of the three stations (expose, process and read) are permitted within the limits imposed by film storage loops between stations.

A small loop of film (Loop Feed) is maintained prior to the expose station so that the film can be quickly pulled down by the drive mechanism without encountering excessive drag. The film is pulled through the expose station in one-frame increments at a rate of approximately one frame every 210 ms by the expose station drive motor. The exposed film is accumulated in Storage Loop 1 until processing is desired. The maximum capacity of Storage Loop 1 is 20 frames before processing must be performed. When the computer signals the channel to process the exposed film, the film is moved through the process station at a rate of 31 inches per minute by the processor drive motor.

Storage Loop 2 (between the process and read stations) accumulates the processed film until the computer advances the film to the read station. If Storage Loop 2 becomes full and processing is still going on, film is forced to advance through the read station. In this way, processing is not interrupted to prevent film from being ruined through over-development or under-development. Film can be backspaced at the read station into Storage

Loop 2; its maximum capacity is 20 frames. At the read station, film may be advanced or backspaced one frame at a time under computer control at a rate of approximately one frame every 170 ms or advanced or backspaced under operator control at one of two speeds:  $\frac{1}{4}$  inch or one inch per second as seen on the screen.

### Photographic System

#### Optical Elements

There are five essentially independent optical systems in the image processor:

1. Input camera system
2. Scanner system
3. Recorder system
4. Projection system
5. Alignment system

The first four systems are used in the operation of the image processor while the fifth is a maintenance aid. All of the optical paths (except alignment) are shown schematically in Figure 1-5.

The input camera system consists of a 22-inch-square, paper-input drawer (See Figure 1-2) with a vacuum-actuated platen to keep documents flat; a series of tungsten, line-filament, light sources located above the drawer; a series of reflecting mirrors; a 4-inch, f/4.0 lens with an electrically-actuated shutter and an expose station on channel A.

In operation the drawer would be extended outside the image processor, a document placed in the drawer, and the hold-platen actuated. Upon depression of a control switch, the drawer with the document on it will automatically return to its normal position, the light sources will be turned on, the shutter will be opened, and an exposure made on 35mm film at a reduction of 18.3X. Both the intensity of the light sources and the shutter timing are variable to provide for flexibility of exposure. The resolution of the optical system at the film plane is approximately 150 lines/mm.

Film images at the channel A read station can be projected for visual examination or can be scanned electronically by means of the flying spot scanner. Optical switching is used to

obtain the functional selection (See Figure 1-5). A 9-inch, f/4.5 CRT lens was designed to reduce the CRT presentation 2.5X to scan the 35mm film. The lens, corrected for the P16 phosphor emission band, yields a resolution at the image plane of 160 lines/mm with a distortion of less than 0.1%.

A collector lens is placed behind the film image which images the exit pupil of the scanner lens on the sensitive cathode of a photomultiplier tube (PMT) detector. This lens serves to uniformly distribute the light passing through the film over the PMT cathode.

Dual-channel recorder optics are provided to expose film on either channel A or channel B (See Figure 1-5) using a common CRT source with optical switching. Optical switching is also used to select the exposure source from either the paper input or the recording CRT on channel A. The recording operation utilizes a lens similar to that used in the scanner optics except that it is corrected for the P11 phosphor emission band.

The simultaneous projection of film images in the read stations of the two channels to a common viewing screen is provided (See Figure 1-8). The same lens as that used in the paper input camera was selected for projection. Off-axis projection, using the displaced image plane technique, gives maximum screen illumination with minimum distortion. Superposition of the two projected images is achieved by moving the film along the vertical axis of the screen and by racking the channel B projector lens along the horizontal axis.

Color filters are provided in the projector lamp housings to aid in differentiating the two images at the screen while projecting simultaneously into the projectors to provide stereoscopic or 3-D viewing. When these filters are introduced by electrical command into the projector light path, the projected images are selectively polarized. Complementary polarizing glasses must be worn by the viewer. The capability of advancing and backspacing the film in the read station is provided. A magnification of 18.3X is used in projecting the film on the 20 by 22 inch projection screen.

The CRT's internal alignment optics serve as a reference to which the recorder and scanner

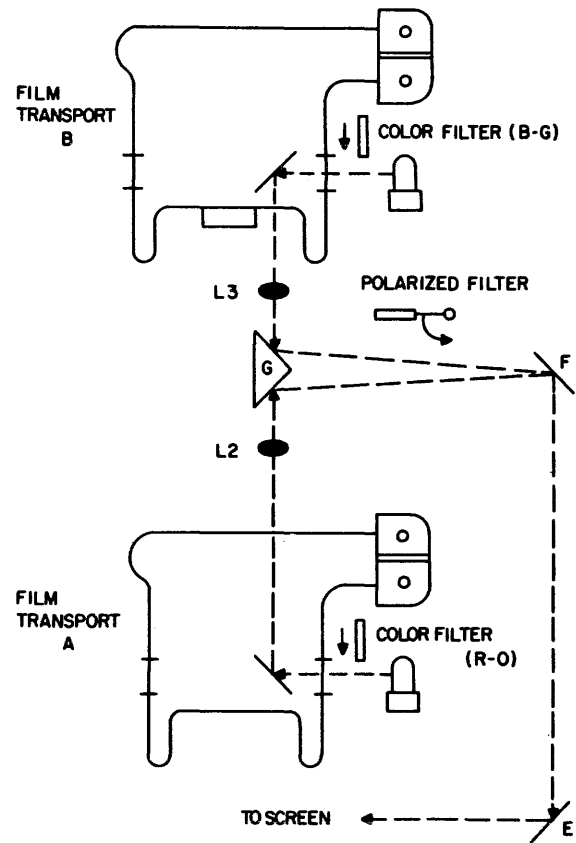


Figure 1-8. Simultaneous A-B Projection, Simplified Diagram.

cathode ray tubes are aligned. An illuminated reticle is brought into visual superposition with either CRT display. Switching from the recorder CRT display to the scanner CRT display is accomplished by manual rotation of a beam splitter. Variable magnification of the alignment system allows for both gross and detailed inspection of the CRT displays.

#### Recording Media

The selection of photosensitive material required the weighing of several desired characteristics. For document-recording, a film in the microfilm resolution class was desired. For CRT recording, a film with an extremely high sensitivity was desired. An additional design criterion was that of elevated temperature, short-time film processing. The silver halide film emulsion designed for this unit has a high blue sensitivity, a reasonably high resolution—130 lines/mm, medium contrast and capability of withstanding the rigors of a high-temperature process.

### Rapid Film Processing

Upon a command, the exposed film is automatically processed in the rapid-film processor, which brings chemicals for processing at high temperatures in contact with the film emulsion. This high temperature increases the chemical activity of the solution such that the film can be processed with a total contact time in the developer, fixer and rinse of only five seconds.

A negative-pressure, cup-application method was selected for the processing (See Figure 1-9). The compliant rubber lips on the solution-applicator cavities form a seal with the film emulsion. Pumps situated in the heated-fluid container draw the solution from the container through the hose to the applicator cavities and back to the pump. The negative-pressure fluid system greatly reduces liquid spillage hazards since any leakage in the liquid circuit results in air being drawn into the circuit. This is particularly important when a process such as this one is integrated into a complex electronic device. The rubber lips on the applicator act as squeegees between the sequential processing steps to minimize contamination of solutions. The exit lip of the applicator removes the surface moisture from

the film to minimize drying time. The film dryer directs high-velocity heated air against the film emulsion which is thus dried in approximately one second. Vent valves located in the return line to the pump are used when the processor is moved away from the film. When these valves are actuated, air enters the fluid line allowing for gravity drain of the applicator cavities. The applicator can then be lowered away from the film, without spilling any chemicals.

The processor processes film at a 31 inches/minute rate. The compartmented solution tank has a volume to accommodate the processing of 400 feet of film. The solutions in the tank are maintained at 130° F through the use of a blanket heater and a temperature controller. The one solution tank and two sets of solution pumps supply the solution to the two rapid film processors, one for each of the two channels.

### Analog System

#### General Description

The analog system, which controls the scan, display, and record CRT's, scan detection, and position-pencil operation, is shown in Figure 1-10. As can be seen, a single set of analog circuits is used to control all three CRTs. Switching between tubes is done with relays and is under computer control.

The control circuits relating to the CRTs perform three basic functions. The deflection-control system precisely controls the position of the electron beam on the face of a given CRT as a result of a sequence of digital X, Y addresses supplied by the computer through the control unit. The focus control provides a uniform (in size), round CRT beam over the entire usable area of the flat-face record and scan CRTs. Without this control, the CRT beam would increase in size and become astigmatic (oval) as the beam position moved off-axis. A farther requirement of this control is to provide for four program-selectable line widths (CRT beam sizes) for film recording. The intensity control is required to maintain constant beam brightness in the system CRTs, independently of beam size (line width) and beam velocity.

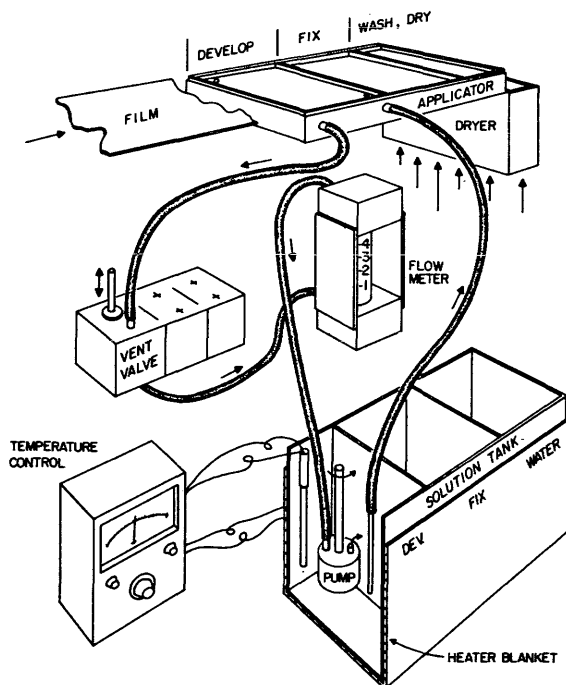


Figure 1-9. Film Processing System.

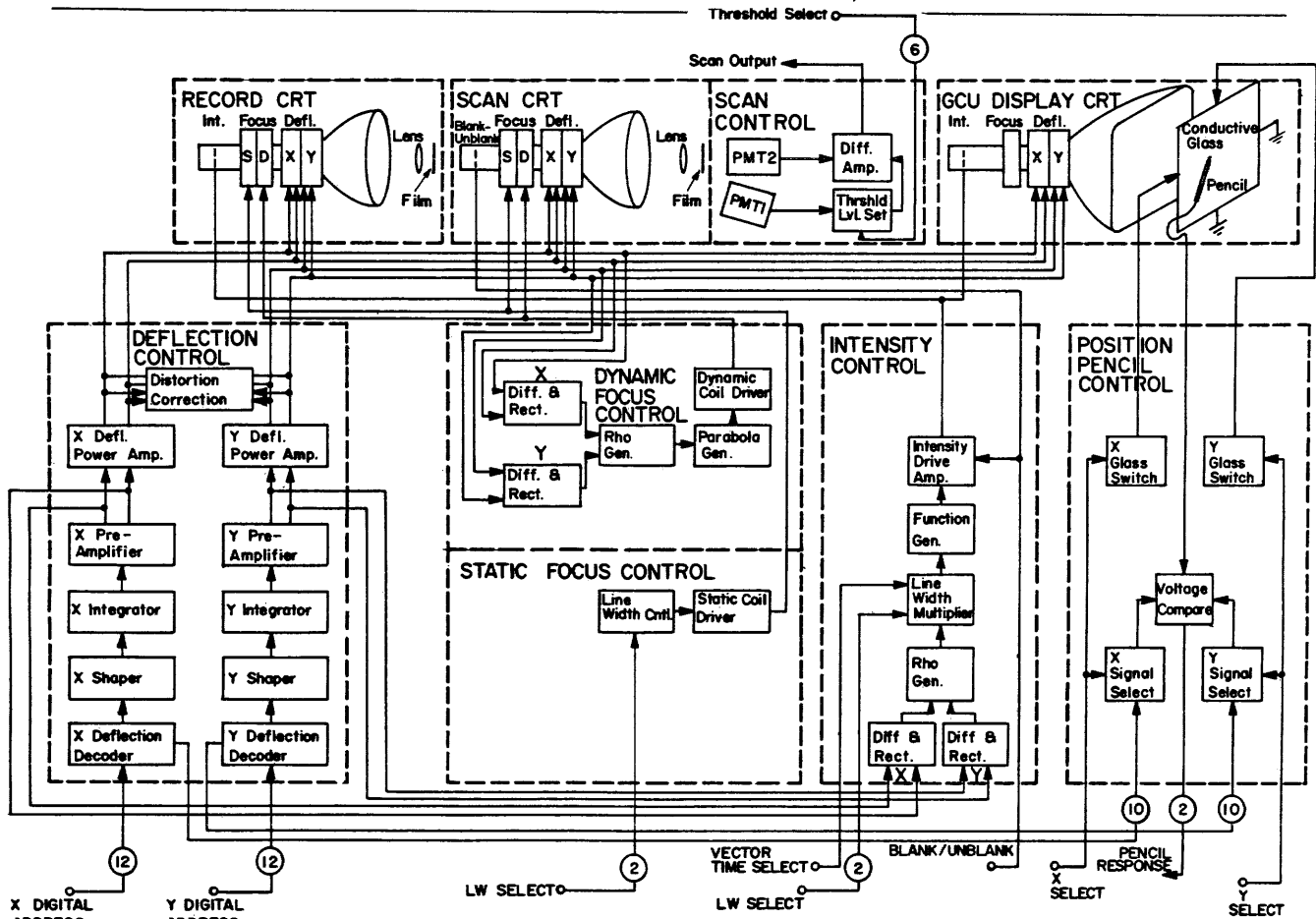


Figure 1-10. Analog System Block Diagram.

The position-pencil control system allows the pencil, when in contact with the conductive glass screen of the GCU, to be located by the computer such that the CRT beam (either blanked or unblanked) appears at the pencil location. The scan detection system senses the light output of the scanner CRT which is modulated by the film image being scanned and correlates the amount of light received at a particular time to a position of the film image. There are 64 program selectable threshold levels representing image transmissivities from 0 to 100%.

The following sections describe each of these major control systems.

**Deflection Control**

The deflection control circuits utilized in the 7960 System are shown in block diagram form in Figure 1-10. Note that the circuit configuration is identical for both X- and Y- deflection channels. The main elements of the X-

deflection Y- deflection control circuits are the 12-bit, digital-to-analog converter or decoder, the waveform shaper, the integrating network, the pre-amplifier and the deflection yoke current amplifier. Another circuit which is common to both the X and Y channels is the distortion correction system which provides deflection yoke current compensation to minimize pin-cushioning effects on the flat-face record and scan CRTs.

The decoders convert the digital addresses received from the computer into an analog signal proportional to the 12 binary-weighted bits. The output of the decoder is a current level which remains constant until a new address is received from the computer. The output then changes in a step-like manner to the new current level where it remains until still another address is received.

The decoder output is then fed into the waveform shaper network which converts the



current steps into a voltage waveform. As may be recalled, the 7960 System operational characteristics require a so called constant-time, end-point, vector-generation mode in which the CRT beam is deflected from a previous end-point address to a new end point address in a straight line and in a constant time, T, regardless of the distance between the points. A further requirement imposed by the scanning system was that the beam move between the points at a rate linear with time. In other words, the beam would move one quarter of the distance between points in a time T/4, half the distance in T/2, etc. In order to achieve these two objectives, it is necessary to generate a deflection waveform in which the change in current or voltage from one level to the next takes place in a constant time T and at a linear rate. The shaper output, when integrated, provides such a waveform. The time period, T, is program selectable to be either 32 or 256 microseconds. Restrictions on dynamic ranges of the circuits limit the maximum positional change in any one cycle to  $\frac{1}{8}$  of the total X, Y positions in the 32-microsecond mode, and  $\frac{1}{4}$  of the total X, Y positions in the 256-microsecond mode. The output of the integrator is then fed into a preamplifier which provides an impedance match with the deflection-yoke power amplifier and also converts the single-ended input signal into a push-pull output signal.

The deflection-yoke power amplifier provides the current into a high-performance push-pull deflection yoke for driving the 5-inch record and scan CRTs. The yoke was selected for maximum perpendicularity and linearity, and minimum residual magnetism (or hysteresis). The power amplifier drives a lower performance, push-pull deflection yoke when connected to the 17-inch display CRT. The maximum display area utilized is 10 inches square.

The record and scan CRTs are provided with optically-flat faceplates for utilization with an optics system. Because of the flat faceplate and the fact that a change in deflection current produces a proportional change in the sine of the deflection angle, an optical distortion known as pin-cushion is observed at the faceplate. This distortion can be explained best with the

aid of Figure 1-11. If the face of a CRT had a radius of curvature equal to the distance from the center of the deflection coil to the screen, the deflection distance A would be proportional to the sine of the deflection angle and thus to the deflection current. The image thus produced by independent X, Y deflection would appear, when viewed from a distance, to be undistorted as indicated by the inside box in the figure.

With flat-faced CRTs, however, the deflection distance,  $A^1$ , is proportional to the tangent of the deflection angle and thus proportional, non-linearly, with deflection current. The effect of this is that deflection distance increases somewhat faster than the current. Under these conditions, the X and Y deflection components interact, producing the pin-cushion pattern shown in the diagram. For the maximum angles of deflection utilized in the record and scan CRTs, the maximum displacement error at a corner of the image would correspond to approximately +6% (proportionate distance between B and B<sup>1</sup>). To correct for this error and meet the requirements for positional accuracy, a distortion correction circuit is utilized to provide correction which can be expressed mathematically as follows:

$$\Delta X = -KX(X^2 + Y^2)$$

$$\Delta Y = -KY(X^2 + Y^2)$$

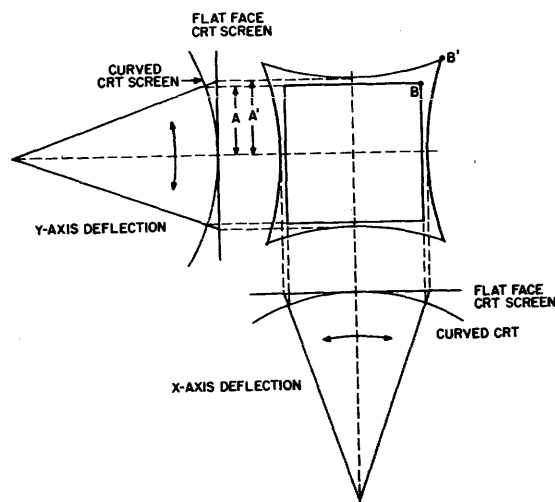


Figure 1-11. CRT Pin-Cushion Distortion.

where  $\Delta X$  and  $\Delta Y$  are the deflection correction currents,  $K$  is a proportionality constant which is a function of the system geometry, and  $X$  and  $Y$  are the deflection currents.

#### Focus Control

The focus control system provides two separate and concurrent functions:

1. A dynamic control to compensate for beam defocussing as a function of beam position.
2. A static control to generate the correct size of the CRT beam as determined by the programmed line-width selection.

The requirement for dynamic focus compensation is, like pin-cushion, caused by the optically flat faceplate of the CRT. As the beam is deflected off-axis, the distance to the screen increases and, for a constant value of focus current, the beam defocusses. By changing the current in the focus coil, and thus the magnetic field, the beam can be refocussed at any point on the CRT screen.

Because the CRT geometry is radially symmetrical, the focus error function is also radially symmetrical and mathematically approximates a parabolic function. The block diagram of the focus control is shown in Figure 1-10.

The rho generator produces a signal which is the approximate vectorial addition of the  $X$  and  $Y$  deflection components referenced to the electrical geometric center of the CRT. The output of the rho generator is then fed into a parabola generator which provides an increasingly large amount of compensating signal through the dynamic focus coil driver as rho increases (or as the beam moves toward the edge of the CRT screen).

The static focus circuit produces the current required to provide four different spot sizes. It is static in the sense that for a given spot size, the current in the static focus coil remains constant regardless of beam position. The smallest beam spot is the true focus condition; the larger spots are obtained by defocussing. Each spot size provides different line widths for image recording of vectors. The line widths are provided in the ratio of 1:2:4:6 where the minimum spot size or line

width relative to the 1.2-inch square film image is less than 0.001 inch. Line-width control is provided by relay selection under computer control.

#### Intensity Control

Intensity compensation is required to maintain constant beam brightness in the CRT's. Two operating conditions account for this requirement.

1. Beam sweep speed. Because the vector time,  $T$ , is constant (either 32 or 256 microseconds) regardless of vector length, the beam sweep speed or velocity varies. In order to maintain constant brightness, the CRT beam current must be increased proportionately with vector length.
2. Line width. As line width is increased, the beam current is effectively spread out over a larger area. In order to maintain equal brightness over all line widths, beam current must be increased proportionately to the increased line width.

The requirement for intensity compensation applies only to the record and display CRT's: the record CRT to provide an even exposure of the film; the display CRT to provide an evenly illuminated display. The beam velocity range of the scan CRT is much more restricted and only the basic line width is utilized. Thus, no compensation is required for the scan CRT.

Dynamic beam intensity compensation is accomplished by determining the status of the three variable quantities:

1. Vector length (continually variable from zero to  $\frac{1}{4}$  full image size).
2. Line width (basic, 2X basic, 4X basic, 6X basic).
3. Vector time (32 or 256 microseconds).

A block diagram of the intensity control circuits is shown in Figure 1-10.

The length of each vector is determined by sampling the  $X$ ,  $Y$  deflection signals, differentiating and rectifying these signals, and then feeding them into a rho generator. The rho generator produces an output signal which is the approximate vectorial addition of the change in the  $X$  and  $Y$  deflection components.

The output of the rho generator is thus a signal proportional to vector length or velocity. This signal is fed into a circuit called a line-width multiplier. This circuit is controlled by the status of two digital lines from the computer which relate to the particular line width selected at any given time. The signal out of the line width multiplier is proportioned in the same ratio as the line width options (1:2:4:6). For example, if the 4X basic line width is selected, the output is four times as great as when the basic line width is selected for a vector of a given length. Another input to this circuit is the vector time selection. The signal level, as described above, is further modified as a function of the vector time, 32 or 256 microseconds.

The CRT beam intensity is, for all practical purposes, linearly proportional to the beam current. The intensity control circuit changes grid voltage to control this beam current. However, the CRT grid voltage-to-cathode (beam) current transfer characteristic is not linear. Therefore, it is necessary to provide a non-linear function generator circuit which compensates for the CRT characteristic such that the intensity compensating signal, as derived from the output of the line width multiplier, provides a non-linear grid voltage signal in such a way as to provide the proper level of beam current. The final block is the intensity drive amplifier which is a gated amplifier controlled by the blank/unblank line from the computer.

The intensity control is capable of providing an intensity level for film recording that permits lines of varying lengths and thicknesses to maintain a density tolerance of  $\pm 0.1D$  about a nominal level of  $0.7D$  after rapid processing.

#### Position Pencil Control

The position-pencil control system allows the pencil, when in contact with the conductive glass screen of the graphic console, to be located by the computer and a CRT beam (blanked or unblanked) to appear at the pencil location. By sampling at a rate high in comparison with the motion of the pencil, the position-pencil control system can maintain the position of the beam under the pencil, making it appear that the pencil traces an image on the CRT screen.

The block diagram of this system is shown in Figure 1-10. The conductive screen is a piece of glass 14 inches square, coated with a thin transparent layer of tin oxide and placed directly in front of the display CRT. A voltage is alternately applied to the screen through the X, Y glass switches causing a voltage gradient to develop on the screen, which is oriented left to right (X) or top to bottom (Y). The pencil when in contact with the screen will thus alternately detect a voltage which is proportional to the distance that the pencil is from the left side of the screen or the top of the screen. This voltage is fed into one side of the voltage comparator. The opposite side of the comparator is supplied with a signal alternately from the X and Y deflection decoder. A comparison is thus made at a given time of the Y position of the pencil and the Y position of the CRT beam or the X position of the pencil and the X position of the CRT beam. The output of the comparator is two digital lines to the computer which indicate that the pencil coincides with the CRT beam, is to the right or left (for X) or above or below (for Y) the CRT beam, or that the pencil is not touching the conductive screen.

Programs have been written which use the interpretation of the digital output of the comparator to direct the CRT beam to coincide with the location of the pencil to accomplish pencil tracking.

#### Scan Control

The scan operation is initiated by an unblanked (visible) beam deflection, or scan vector, written on the scan CRT screen under program control. Light produced by the sweeping beam is sensed by photomultiplier tubes PMT 1 and PMT 2 (see Figure 1-10). PMT 1 receives light directly from the CRT screen. Consequently, PMT 1 receives light whenever the beam is sweeping. PMT 2 receives light from the CRT screen through the film image. Thus, PMT 2 receives light only when the beam sweeps through the clear areas of the film.

The object of writing a scan vector is to intercept the lines on the film image as the beam sweeps. Thus, assuming that the film image is composed of black lines on a clear background (positive mode), the beam light sensed

by PMT 2 will be momentarily interrupted when the sweeping beam intercepts each line. Assuming that the film image is composed of clear lines on a black background (negative mode), PMT 2 will sense light momentarily when the sweeping beam intercepts the clear lines.

However, the PMT 2 output depends on the relationships of beam speed, beam spot diameter, and width of the intercepted line (Figure 1-12). Section A shows positive-mode scanning, in which the line width is greater than the beam-spot diameter. In this case, when the beam intercepts the line, PMT 2 senses total darkness and, accordingly, produces a maximum amplitude level. Section B shows the same beam spot intercepting a line whose width is smaller than the beam diameter. In this case, PMT 2 does not sense total darkness but a degree of light dimming called *gray level*. The PMT 2 output then reflects this gray level with a signal of proportional amplitude. Sections C and D illustrate negative mode scanning where line width is greater and smaller than beam spot diameter.

In practical operation, the most common occurrence is that interceptions with lines of average width produce different gray levels. The PMT 2 signal indicates a gradual transition from light to dark and dark to light. This corresponds to the gradual dimming of the beam in either transition.

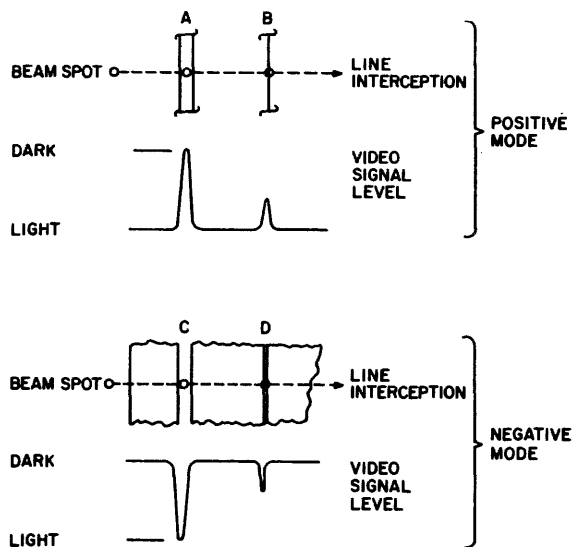


Figure 1-12. Scan Signal Generation.

Since line width is sensed by PMT 2 in terms of varying gray levels, a search is required to make sure that all interceptions, or meaningful levels, are accounted for in the scan process. This search is done by comparing the signal level with a reference level which can be varied by predetermined values. Figure 1-13 illustrates this comparison with examples of typical vector scan operations. In this figure it is assumed that the positive and negative versions of the same image are being scanned and that the sweeping beam makes two line interceptions.

Section A of the figure shows that the reference level, which lasts for the length of the scan vector, can be set to any one of 64 threshold levels by computer program.

Section B shows (1) the superimposed signal and reference levels as they are placed for comparison at the differential amplifier (Figure 1-10) and (2) the threshold levels which are lowered until they cross the higher level signal. Thereafter, the search process requires an additional scan operation with a lower reference threshold to cross the lower-level signal.

Section C indicates the output signal transferred to the control unit to indicate a strike in positive-mode scanning.

Sections D and E indicate the search for signal levels in the negative mode in which the program raises the threshold until it crosses

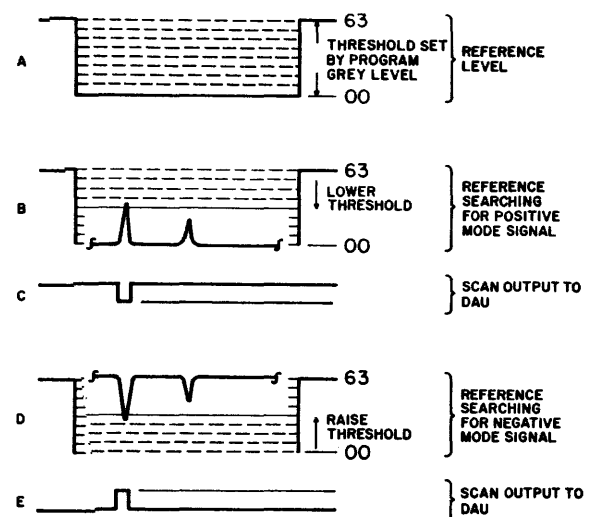


Figure 1-13. Scan Output Generation.

the higher-level signal; an additional scan operation with a higher reference threshold is required to cross the lower-level signal.

In the section on deflection control, it was stated that a requirement of the deflection signal was that it be linear with time. The reason for this is that the strike or output data from the scanner is sampled at 16 equal intervals during one vector time. Thus, in the 32 microsecond mode, the scan output is sampled and stored every 2 microseconds; in the 256 microsecond mode, the output is sampled every 16 microseconds. By making the deflection signal, and therefore the scan vector, linear with time, the time segment during which an output was sensed from the scanner can be correlated to beam position. This is extremely useful in either one of two ways. It provides the equivalent of 16 separate vector scans in one scan vector time period thereby increasing the effective scan rate by 16, or it can be considered to increase the resolution capability of the scanner for a scan vector of a given length by a factor of 16. The example shown in Figure 1-14 illustrates this point. A scan vector is generated which traverses from point A to point B, intercepting four lines on the film image. If the scan output was sampled only once per scan vector, the only information which would be retrieved would be that a line or lines were located somewhere between points A and B. By sampling 16 times during the generation of the single scan vector, not only was it determined that there were four lines but the locations of the lines relative to points A and B can be computed to within one part in sixteen of the distance between the points.

The maximum length of the scan vector (and therefore the CRT beam velocity) is restricted by the band width of the scan deflection circuits which are in turn restricted in order to minimize noise. The resolution of the scanner is primarily a function of CRT spot size. The scanner can resolve lines 0.0005 of an inch thick separated by 0.0015 of an inch relative to the 1.2-inch-square film image.

#### Cathode-Ray and Photomultiplier Tubes

The CRT's utilized for scanning and recording are identical except for their phosphors. Each is a 5-inch, round, high-resolution CRT

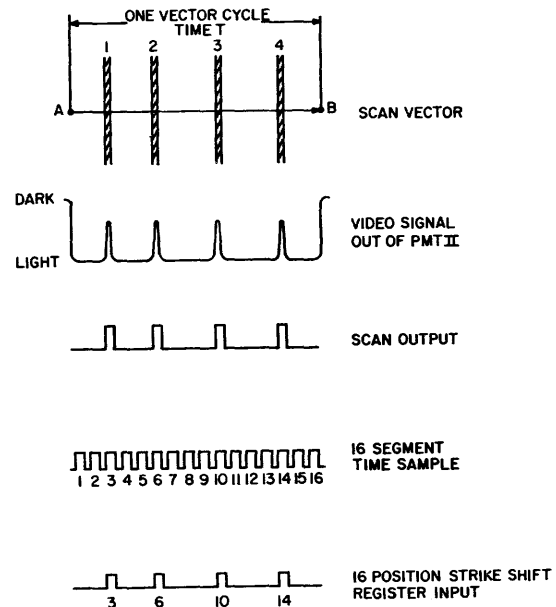


Figure 1-14. Scan Output Time Sampling.

with an optically flat faceplate. Both the deflection and focus is magnetic. Nominal spot size at the CRT screen is 0.001 inch.

The scan CRT employs a P16 phosphor which was selected because of its ultra-fast light-decay characteristic. It has a spectral energy distribution which peaks around 3800 Å (violet and near UV). The scan lens is color-compensated for this spectrum and the scan detection photomultipliers employ an S11 photo cathode which is quite sensitive to the P16 spectrum.

The record CRT employs a P11 phosphor which was selected because of its high light-output capability, which is required for exposing film. It has a spectral distribution which peaks around 4600 Å (blue) and is considered medium fast relative to its decay characteristic.

The display CRT is a 17-inch, rectangular CRT with a curved faceplate and requires magnetic focus and deflection. The nominal spot size is 0.020 inch and the P19 phosphor peaks around 5900 Å (orange). This phosphor was selected for its long persistence characteristic. The usable display area is 10 inches square. The focus control is not connected to the display CRT. Because of the larger spot size and the curved faceplate, a permanent magnet focus was determined to be adequate.

## PART II OPERATIONAL EXPERIENCE

### INTRODUCTION TO PART II

The specifications developed by General Motors Research for new man-machine graphical communication hardware evolved from experience with early GMR Laboratory equipment and from projected requirements of a designer in a computerized design process. The designer, unconcerned with the hardware specifications outlined in part I of this paper, enters the computer room with a drawing or sketch of standard drawing quality as input to the design process. While in the computer room, he uses the graphic console and the computer as tools for rapidly developing, modifying, and reviewing his design. When he leaves, he expects to take with him (without delay) a roll of paper documenting his computer-aided design.

The remainder of this paper evaluates the degree to which the hardware has approached the requirements of a designer in the process of design. In addition, some programming techniques for improving the effectiveness of the hardware by testing and calibration are described.

### THE GRAPHIC CONSOLE AS A MAN-MACHINE INTERFACE

Continued use of the graphic console by a large number of people has led to some conclusions regarding the equipment as a tool in a design environment.

The CRT and pencil have proven to be a highly successful man-machine interface in spite of the small screen size (10" x 10"). Except for a period of two months when a lower persistence phosphor tube was tested in the unit, there have been no serious operator complaints directed toward use of the display tube. The lower persistence phosphor caused a highly objectionable flicker when large amounts of data were displayed. The operator antagonism toward this flicker was sufficiently severe in some cases to keep the individual from the equipment.

The position-indicating pencil has been found to be functionally very suitable. One reason for this is the positive action-reaction; that is,

the program is alerted whenever the pencil touches the screen. Contrast this action with a light pen which must be placed close to a specific area of interest and requires in addition some type of manual switch to alert the program. Another advantage of the pencil compared to a light pen is that pencil position can be determined in milliseconds when the pencil is pointing at a void portion of the image area. This quick position location is essential in an on-line problem solving environment. Users of the pencil must be able to quickly locate in real dimensions any point of the display area. With a light pen it is necessary for the user either to wait for a searching operation or to point at part of a display and then track the pen to an area of interest.

The speed and positive action characteristics make the pencil a comfortable tool in an alphanumeric display-and-correct mode in which the user points to a character of the display and uses the alphanumeric keyboard for correction. The user of the pencil need only touch the screen in the general vicinity of a particular character and the program can determine which character is to be replaced. Users have adapted very rapidly to using the pencil as a pointer.

One disadvantage of a position-indicating pencil when compared to a light pen is that the pencil does not establish a positive reference to a particular line of the displayed composite line image. It is necessary to compare pencil position with the position of each of the displayed lines to determine at which line the pencil is pointing.

From these experiences with the use of the pencil, we have learned that the functional characteristics of a device for pen-pencil man-machine communication should include modes such that:

1. The control program is alerted each and every time the pencil touches the display surface.
2. The control program is able to determine within milliseconds the position of the pencil on the screen independently of the presence or absence of any display.

3. The control program is able to establish a positive identification by the pencil of a particular line of a displayed image.

The use of a man-computer pen or pencil for the input of graphic data is an interesting topic in itself. When analyzing the tracking mode, however, it is difficult to separate the characteristics of the hardware, the programming and the application. While the pencil or a light pen could be used for entering images consisting of connected straight lines, a highly sophisticated program would be required to track and accurately digitize drawings of curves which are connected to form an image. In the GMR DAC-I System, precision graphic data input is entered through the paper input facility of the image processor.

The program status lights and control keys have also proven to be excellent both for the programmers and for the man at the console. The alphanumeric keyboard, on the other hand, has been an object of much discussion from which the following controversial points have arisen:

1. The keyboard should be a standard typewriter keyboard to take advantage of the speed of those who know how to type.
2. The typist-designer intersection is small; therefore, use a keyboard that is arranged in some order to minimize learning.
3. The typewriter should be installed immediately in front of the display tube to maximize display feedback of a typed message.
4. The space in front of the display tube should be reserved as a work area for listings, drawings, note pads, etc.
5. The typewriter would be of great value for long message input.
6. No long messages should be entered at the graphic console.

This type of discussion can be and has been carried out to great lengths by human factors people. Our experience has shown that an alphanumeric keyboard is difficult to cope with and not a natural device for man-machine communication. Until a pencil entry device accompanied by character recognition is available, however, the keyboard must remain an important part of the man-machine interface.

## EVALUATION OF THE IMAGE PROCESSOR

The graphic console provides the user with the facility for a dynamic display of comparatively low accuracy and repeatability. The voltage pencil is also dynamic in its functional capabilities and not intended to be used for the entry of precision graphic data. The image processor on the other hand is expected to provide the more static man-machine graphic communication with higher precision input and output.

When establishing criteria for the evaluation of image processing equipment, it is essential that the criteria be defined in terms that give an overall evaluation of the unit, encompassing electronic circuits, optics and photography. The resolution of the CRT as measured by the shrunken raster method, for instance, is of little interest to the user of an image recorder. For Design Augmented by Computers the user wants to know, first of all, the line width and resolution as measured on the paper of the hard copy output from the recorder. If he must supply his own hard copy machine, the user wants the specifications as measured on the processed photographic film. From the user's standpoint, all specifications must be based upon the response of the unit in the final output state and the specifications must be measurable at that final output state.

What follows is an important subset of image processing evaluation criteria and programs resulting from daily use of the image processor since early 1963. The criteria are specified in a manner that allows rapid evaluation whenever possible. The programs are intended to provide versatility for the evaluation of the equipment over a wide range of conditions and requirements.

### *Accuracy*

The user of image processing equipment for Design Augmented by Computers wants to know the accuracy of the positioning of a point on the output image when referred to any other point on the image. He wishes to measure with a rule the distance between two points and know his confidence limits. Accuracy to the user is then defined as the maximum error in the distance between any two points on an

image. It is measured by recording a pattern of vectors with known spacings. The distance between vector endpoints is measured on hard copy if provided, on the film image with an optical comparator, or on the film image using the image scanner. The maximum error encountered in measuring the distance between any two points is the accuracy characteristic of the unit. With a little experience, the user learns the particular pattern and vector endpoints that demonstrate the maximum error.

Figure 2-6 shows the accuracy errors of various points (referred to the center point) on the scanner image magnified by a constant factor. Note that although the accuracy figure is stated in terms of per cent of image size, the error measured is an absolute value that includes pincushion error (distortion corrected) and all other electronic, photographic and optical errors. Note also that this method of measuring accuracy results in a figure nominally twice that of those techniques that measure accuracy as the error of one point referred to an origin. This method does, however, yield a value of accuracy such as a designer would measure with his rule.

Users of the equipment are not completely satisfied with the accuracy (approximately 1%) resulting from uncalibrated scanning input and output recordings in spite of the fact that the equipment represents an advanced state of the art. Calibrated input and output accurate to 0.2% barely meets the requirements of static man-machine communication.

#### *Stability and Repeatability*

To the user of DAC-I image processing equipment, stability means freedom from drift of analog components including power supplies and deflection circuitry. This specification is important since it is essential in maintaining constant raster shape and size on a recorder and scanner CRT. Accuracy can be improved by calibration procedures, if and only if the hardware stability is such that the calibration runs can be spaced at practical intervals of time. A practical interval of time might be approximately one hour unless some form of automatic calibration is provided.

Repeatability is the degree of capability of the hardware to exactly duplicate an output

condition after an intervening number of random input conditions. Other definitions of repeatability which are dependent on repeating a sequence of inputs are of little or no value to the users of image processing equipment.

Imperfect vector endpoint repeatability shows up dramatically in circles (see Figure 2-1) which do not close properly and at intersections of a number of lines which are all supposed to pass through the same point. Another example of repeatability error is the case where two or more lines are spaced close together. The lines may touch or even cross if the repeatability is poor. Although this type of error may be aesthetically less troublesome than poor repeatability at intersections and at closing points of circles, it would be important if a user were trying to evaluate positions of edges of parts which are to fit together.

It has been observed that it is the natural tendency of the human eye to notice the worst case of repeatability on recorded output even if all the remainder of the drawing represents high-quality output in terms of repeatability and accuracy. The width of the lines affects the aesthetic appearance of the output. The observer immediately spots repeatability errors at the junction of thin lines while the same errors with thicker lines are overlooked. Since thin lines are normally more desirable on recorded output, some compromise is frequently required.

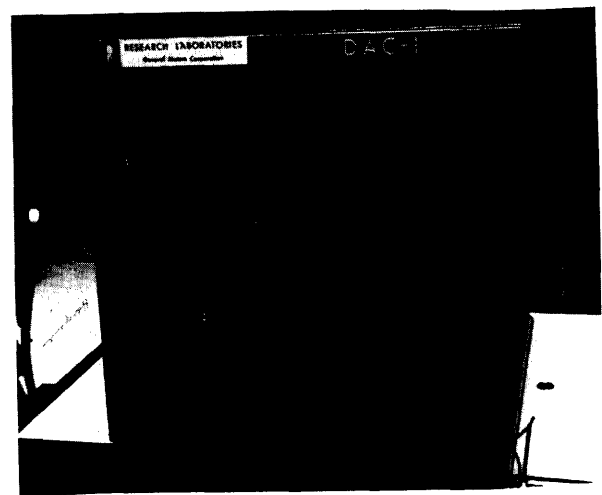


Figure 2-1. Circle Pattern for Demonstrating Repeatability.



The pattern in Figure 2-1 has proven to be excellent in showing the limit of the recorder repeatability for quick approximations. The errors in the closing of the circles can be observed in terms of line widths and then converted to per cent of image size. The maximum repeatability error for the circles, for instance, might be approximated at one line width.

Repeatability is also an important factor in scanning operations. In line tracking, repeatability is particularly important when input lines have small changes in curvature and the positions of inflection points are important. Repeatability errors also put a limit on the effectiveness of calibrating for improved accuracy.

Experience has shown that the effective repeatability of the scanner can be improved by repeated scanning at a given region and then averaging the results of these several scans. However, this does not eliminate repeatability errors, but merely reduces the magnitude of the errors when using the scanner.

#### *Line Width and Intensity*

Recorder line width must be defined in terms of measurements on micro-densitometer recordings from the output film. The width of the line is arbitrarily defined as the width of the line at the mean light transmission level where the mean is the average of the film background light transmission and the transmission at the peak of the line. Use of the microdensitometer gives a numerical measurement of the line width independent of human observation. Line intensity of recorder output film is defined as the optical density of a line at the center point.

Variations in the optical density and line width are very noticeable to users of the image processor recorder. Line density and thickness variations with vector length frequently result in short vectors having a greater density than the longer vectors. This is particularly objectionable in areas of a drawing where information in the form of sharply curved lines is concentrated. In this case, sharply curved lines require many short vectors with a corresponding high density and loss of fine detail. In addition, small alphabetic characters, normally readable, may become illegible when line width and intensity are out of adjustment for short vectors. Variations in line density and width with vec-

tor direction and position has also proven to be very noticeable to the equipment users. Even if drawings are made accurately with only small repeatability errors, poorly controlled line widths and peak densities will make the recorded output look poor.

The pattern of Figure 2-2 was generated by a program that has proven excellent in testing for the conditions mentioned above. The position, direction, and lengths of the vectors were selected at the graphic console to show how uniform the line widths and intensities are over a wide range of these parameters.

#### *Film Processing Time*

It has been observed that any time an operator has nothing to do, or no drawings or displays to review, a forced delay of one minute or even half a minute becomes annoying. Even though the rapid film processor develops, fixes, rinses and dries film at a rate of approximately one frame every three seconds, a minimum of 25 seconds is required to process the first frame plus three seconds for each additional frame.

Whenever practical, the graphic console operator should be given a choice of other functions to perform while film is being processed for the best man-machine interaction, at least until film processing speeds increase by another order of magnitude or some other more rapid

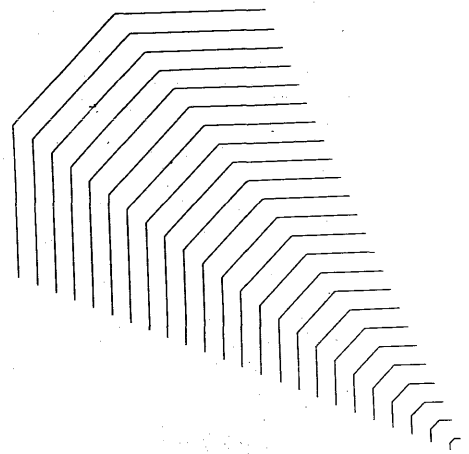


Figure 2-2. Intensity Control Pattern for Demonstrating Line Intensity and Width Conditions.

form of static image documentation is available.

### *Scanner Sensitivity*

The sensitivity of the scanner is defined by the thinnest line the scanner can consistently detect and the range of thresholds for this detection. Typically, the scanner, using the paper input facility, can consistently detect lines .01" wide (on the 22 x 22 inch paper) over a range of three thresholds.

A considerably more complex algorithm can be used to define the sensitivity of the scanner over a range of line and background densities. This is essential for determining which photographic images can be expected to be scanned. A simple definition and measuring technique is necessary, however, for determining the hardware sensitivity.

Figure 2-3 shows what is displayed on the graphic console within two minutes after the start of the sensitivity test. This display demonstrates to the user that lines nominally 9.8 raster units wide (referred to the 0-4095 raster unit image) can be detected over a range of 17 thresholds. It further points out the line thickness that one may expect at each of the threshold levels. It is now standard practice to test the equipment daily using test programs which are highly application oriented and

which can analyze the equipment and display the results for immediate review by the maintenance engineers or by users of the equipment.

### TEST PROGRAMS

It is no small task to define the exact performance status of electronic systems which are comprised only of digital components. It is still harder to define the status of electromechanical units. In the first instance, it is the complexity of the system that causes the difficulty even though the system operates at a discrete level for performance evaluation. In the second case, the mechanical positioning involved may result in a series of discrete levels that must be evaluated. Defining the performance level of an analog-digital-mechanical system such as the image processor is still more difficult because of the continuous range of the operational status which is biased by the discrete digital-mechanical levels. Test programs which are understood and used by both the maintenance people and the operators of the equipment have proven essential in obtaining the maximum performance from the equipment for the operator and computer time involved.

Hardware test programs for the DAC-I system have been written primarily for the scanner and the recorder. The recorder programs basically provide a variety of test patterns to

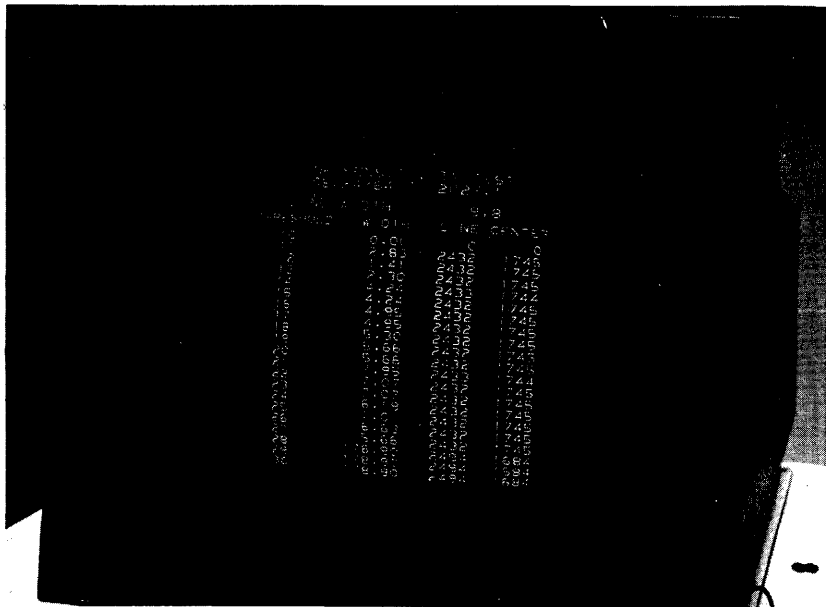


Figure 2-3. Scan Sensitivity Test Program Display.

show recorder repeatability, accuracy, line intensity, width and smoothness. The patterns are selected to demonstrate dramatically each characteristic to be tested. This frequently allows preliminary evaluation at the projection station. Detailed evaluation is made with test instruments when necessary. Test programs for the scanner include the program for line sensitivity described earlier, line resolution, threshold characteristics and scanner and recorder accuracy and repeatability.

With a man-machine console, test programs should include the following features:

1. rapid measurements
2. rapid editing of the measured data so that only a small amount of important information will be displayed
3. rapid display of results
4. options to record results on film
5. pictorial rather than numeric display of measurements when requested
6. options for conventional printout
7. facilities to easily change and display program parameters.

Permanent records of test program results are kept to monitor and signal when a long-term decrease in hardware performance should be corrected. These records also aid in the evaluation of changes in the hardware.

Sometimes adjustments or changes are made that result in a temporary improvement in the quality of the hardware. The test program results, when analyzed over a period of time, show if the change corrected a problem or merely adjusted around it temporarily. To illustrate some of these desirable characteristics of test programs, we will outline as an example the main features of the calibration test program for scanner and recorder accuracy.

The calibration test program uses the scanner to measure the nonlinearities of both the scanner and recorder caused by electronic and optical effects. The magnitudes and positions of these non-linearities are displayed for immediate review and stored for later calibration of both scanner input data and output recordings.

The reference for linearity and size measurements is a metal plate with grid lines inked

on its surface. The locations of all the intersections have been previously stored in the computer. The grid is exposed onto film through the paper input station, processed, and moved to the scan station.

Before executing a complete scan of the grid, a preliminary scan for the four-corner crosses and the center cross is made. (See Figure 2-4) This allows the operator to check the image size and the optical alignment of the system and insures that the entire image to be scanned is aligned optically and electronically within the scan raster.

Figure 2-5 shows the type of alphanumeric information displayed after scanning every grid intersection and comparing the scanned intersection locations to the reference values. Here a large amount of data has been condensed to a few numbers giving the most pertinent information about the raster size and linearity of the raster. This reduces the question of acceptability of the machine to a matter of comparing the numbers on the graphic console to figures which are previously defined to be acceptable.

For diagnostic purposes, the information shown in Figure 2-6 is most useful. The errors associated with each point are magnified by a factor and added to the reference points to produce an exaggerated representation of the non-linearity and size of the raster (see Figure 2-6). This graphic presentation shows the non-linearity of the raster size and shape. A numerical figure for scanner repeatability is displayed by the calibration program if repeated scans of the grid pattern are executed.

Experience has shown that test programs such as the calibration program have been able to define scanning status before actual digitizing begins and have maximized the usefulness of the scanner operation. Programs like the calibration program are also valuable in separating hardware deficiencies from errors in new programs; that is, if new programs fail but the test programs run normally, then it is likely that the new programs are either in error or expecting too much from the analog hardware.

The calibration program illustrated above has resulted in improved scanning and record-

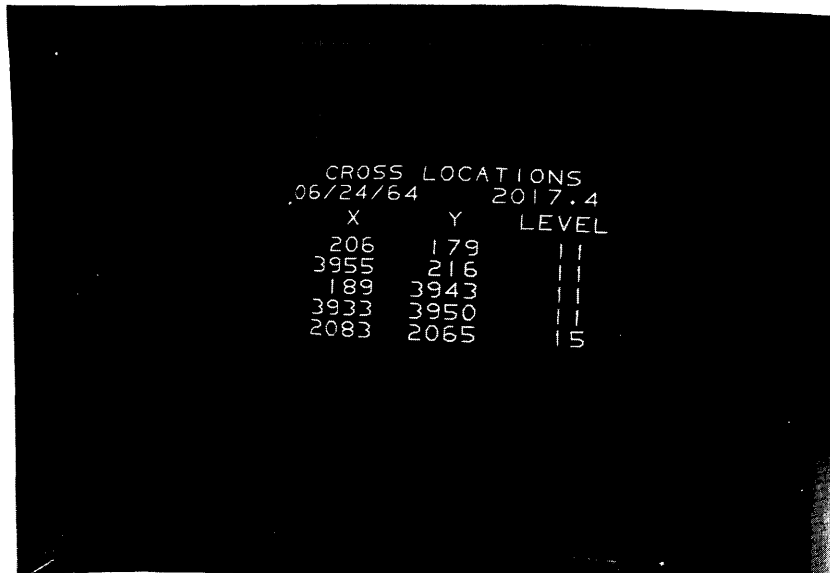


Figure 2-4. Corner and Center Crosses on Scan Program.

ing to the extent of nearly one order of magnitude. The hardware has proven itself sufficiently stable to allow a significant improvement in scanner accuracy by a calibration procedure which uses the error data measured by the test program.

The recorded output can also be calibrated to show a significant improvement in accuracy. The scanner is used to scan recorded output and the errors of the recorder raster relative to the scanner raster are then modified by the

scanner errors and stored for calibration by recording programs. Two of the most important values of these GM test programs are that they provide a good overall test of the system and they are oriented to evaluating the user's requirements.

#### SUMMARY

The IBM 7960 Special Image Processing System was designed and built by IBM to specifications provided by the General Motors Re-

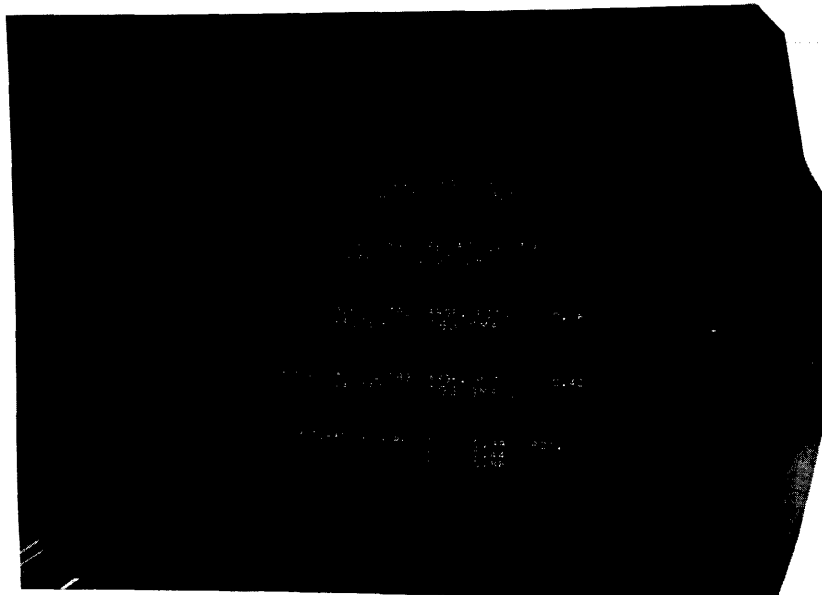


Figure 2-5. Scanner Accuracy and Raster Size.

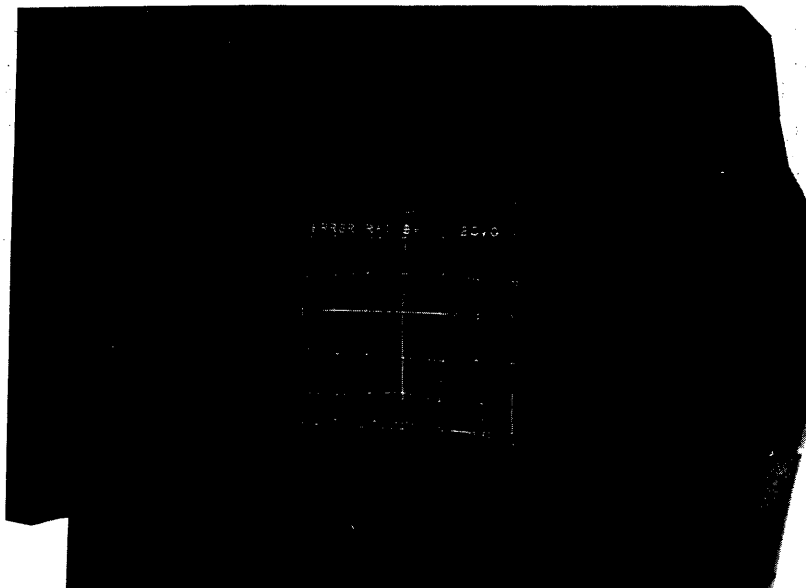


Figure 2-6. Accuracy Errors Multiplied by a Constant.

search Laboratories. The system is the man-machine and image processing hardware for the GM Research DAC-I system.

The design shows how the functional requirements for an image processing system were implemented to achieve a new type of computer input-output system. The components were chosen for characteristics that were compatible with digital computer speed and accuracy. Technologies normally foreign to computer technology were successfully integrated by careful consideration of the interface between components and the effect of each component on the total system performance. Some of the features of the new hardware system such as the high resolution, the excellent accuracy and the rapidly processed film for pictorial input and output have been extensions of the state of their respective arts. These features have also indicated the necessary quality and speed for a graphical machine to interact with a man in the iterations of a design cycle.

The hardware has proven to be valuable as a laboratory tool for the analysis of equipment required in an online computer aided design facility.

The use of image processing equipment in a computing facility has also pointed out some of its interesting operational characteristics. The

digital program-analog response characteristic of the hardware makes system testing and status documentation a necessary part of operational procedures. In addition, programming techniques that improve the apparent performance of the hardware must be used. Man-machine programs such as the calibration program described in this paper have proven to be a solution to these requirements.

#### REFERENCES

1. JACKS, E. L., "A Laboratory For The Study of Graphical Man-Machine Communication," *Proceedings Fall Joint Computer Conference*, San Francisco, California, Oct. 1964, (this volume).
2. COLE, M. P., DORN, P. H., LEWIS, C. R., "Operational Software In a Disc Oriented System," *Proceedings Fall Joint Computer Conference*, San Francisco, California, Oct. 1964, (this volume).
3. ALLEN, T. R., FOOTE, J. E., "Input/Output Software Capability For a Man-Machine Communication and Image Processing System," *Proceedings Fall Joint Computer Conference*, San Francisco, California, Oct. 1964, (this volume).
4. KRULL, F. N., FOOTE, J. E., "A Line Scanning System Controlled From an On-Line

- Console," *Proceedings Fall Joint Computer Conference*, San Francisco, California, Oct. 1964, (this volume).
5. LAZOVICK, P. B., TROST, J. C., REICKORD, A. W., GREEN, R. S., "A Versatile Man-Machine Communication Console," *Proceedings (1961) Eastern Joint Computer Conference*, Washington, D.C., Volume 20.
  6. STOTZ, R., "Man-Machine Console Facilities For Computer Aided Design," *Proceedings (1963) Spring Joint Computer Conference*, Detroit, Michigan, Volume 23.
  7. LICKLIDER, J. C. R., "Man Computer Symbiosis," *IRE Transactions on Human Factors in Electronics*, Volume HFE, March 1960.

# INPUT/OUTPUT SOFTWARE CAPABILITY FOR A MAN-MACHINE COMMUNICATION AND IMAGE PROCESSING SYSTEM

*Thomas R. Allen and James E. Foote  
Research Laboratories, General Motors Corporation, Warren, Michigan*

## INTRODUCTION

Consistent with the design objectives of the General Motors Research Laboratories DAC-I (Design Augmented by Computers) project,<sup>1</sup> the IBM 7960 Special Image Processing System has extensive input/output (I/O) capabilities.<sup>3,4</sup> In order to facilitate the programmed control of this new hardware, the NOMAD and MAYBE programming languages were developed.<sup>2</sup> However, while these languages provided the programmer with an effective means of controlling the new hardware, they offered little assistance in meeting the hardware data format requirements. The programmer still would have had to convert his output information from his own internal format to the output format required by the hardware. Similarly, all input from the hardware would have had to be converted by the programmer back to a form suitable for his own use. Moreover, these conversion processes are typically very involved and complicated. In short, the programmer was still not in a position to make easy, efficient, and flexible use of the I/O capabilities of the new hardware system. Thus, the need to provide a layer of general purpose I/O software between the programmer and the hardware was very apparent.

The effort to develop a general purpose I/O software capability resulted in a hierarchy of utility routines. At the lowest level of this

hierarchy is a set of very basic utility codes. These include numeric-to-BCD (binary coded alphanumeric information) conversion routines, BCD-to-vectors character generation routines,<sup>5</sup> simple display and recording routines, and basic film train operation codes, all of which permit the programmer, if he so desires, to operate very close to the basic hardware without having to understand all of the intricacies of the hardware's operation. Each of these basic codes represents an implementation of only a very small facet of the total I/O capability but, as such, serves as a building block for subsequent level codes in the hierarchy. These higher level I/O utility codes are progressively more inclusive in their total I/O capabilities and, at the same time, relieve the user of the task of dealing with the hardware on its own terms.

This effort to develop an I/O software capability was not designed to culminate in one single all-inclusive routine. Rather, the aim was to produce a set of sophisticated, general purpose, problem oriented subroutines and source language statements, each of which would serve as a powerful tool in the utilization of the various I/O capabilities of the new hardware. The body of this paper describes five representative utility subroutines, three source language I/O statements, and some typical examples of their application.

## UTILITY SUBROUTINES

This section describes five different tasks and the I/O utility subroutines which were developed to meet their requirements. The basic problem associated with all of these tasks is that of requesting various types of information from the graphic console operator. Hence, the emphasis is on the implementation of the various input devices associated with the graphic console. The graphic console display CRT (cathode ray tube) is used to indicate what type of information is required.

### *Alphanumeric Input*

Subroutines names, variable names, titles for recorded film output, etc. constitute one type of information frequently required from the

graphic console operator. This information is most conveniently handled by the programmer in the form of BCD character strings. Thus, the task here was to provide a facility for requesting and accepting these strings. Two input devices are appropriate: the alphanumeric keyboard and the card reader (see Figure 1).

In addition to the primary requirement that the I/O subroutine for this task be simple for the programmer to use, two other requirements were felt to be important. First, there was the need to give the graphic console operator immediate feedback which would permit him to verify that each character has been entered correctly. Concomitant with this was the need to provide the operator with a means of correcting erroneously entered characters.

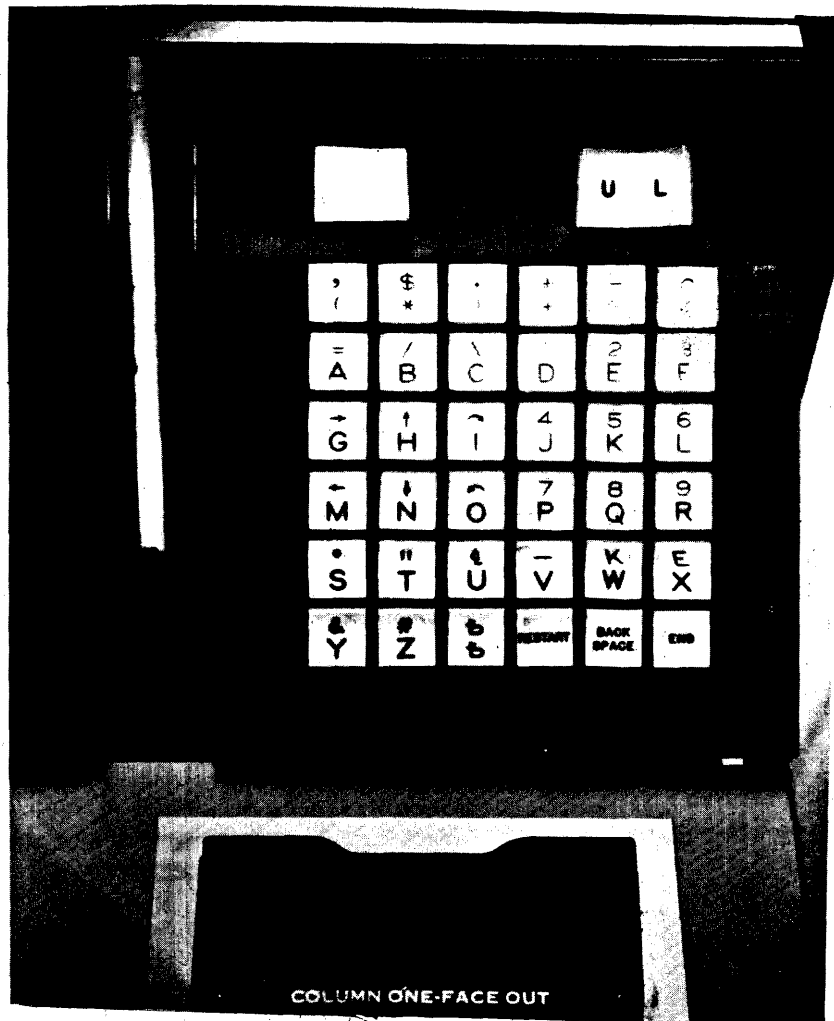


Figure 1. The 7960 Alphanumeric Keyboard and Card Reader.



The subroutine RINSE (*Request for INformation Subroutine*) was developed for this task. To use this subroutine, the programmer specifies three items of information: (1) the symbolic name (i.e., location in core) of an array which contains the request message in BCD format, (2) the maximum number of BCD characters which the operator is allowed to enter, and (3) the symbolic name of an array in which to store these characters. Figure 2 shows a typical set of source language statements for utilizing the RINSE subroutine. The DIMENSION declaration is used to reserve space for an array at compile time. The VECTOR VALUES declaration is used to preset an array (i.e., vector) at compile time. The EXECUTIVE statement generates a call to a subroutine.

In Figure 3 the request message in the example above is shown as it would appear on the display CRT upon the execution of RINSE.

Assume that the graphic console operator wishes to enter the subroutine name: ABC. At this point, the operator can either type in the subroutine name at the alphanumeric keyboard or insert a card in the card reader. If he elects to type in the name, each character will be added, one at a time, to the display. If he inserts a card in the card reader, characters will be added to the display sequentially as they are read from the card. The console operator can intermix these two modes of input. Figure 4 shows the display as it would appear after the subroutine name has been entered.

The operator can delete the last character in the string by depressing the BACKSPACE key or delete the entire string by depressing the RESTART key (see Figure 1). He may then enter more characters. In this manner, erroneous characters may be corrected. Once the input is satisfactory, the operator depresses the END key, the display is terminated and the BCD character string is passed back to the calling program.

```
DIMENSION REPLY(1)
VECTOR VALUES REQUEST = 5, $ENTER NAME OF SUBROUTINE$
EXECUTE RINSE.(REQUEST,6,REPLY)
```

Figure 2. Source Language Statements for the RINSE Subroutine.

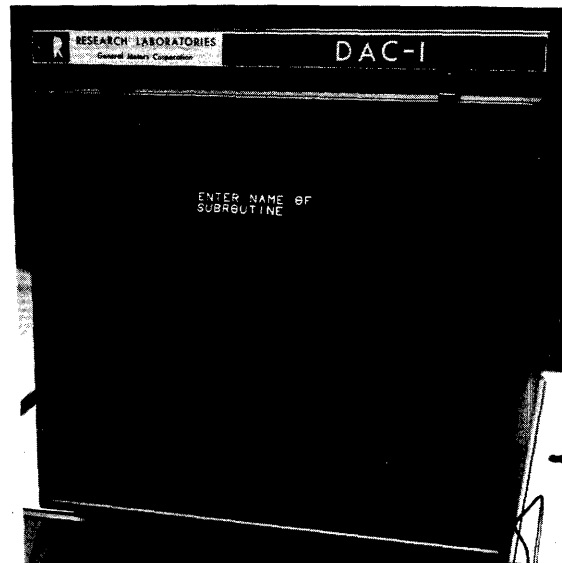


Figure 3. RINSE Subroutine Display.

### Numeric Input

Another type of information which the programmer may require from the graphic console operator is numeric data. The task here is one of providing a facility for initializing and/or modifying data variables. Two subroutines were developed to meet this requirement. The difference between these two routines stems from the two basically different ways in which data variables can be defined: as elements of an array or as a set of distinct variables which may be widely scattered throughout memory. In the first case, each variable is referenced by giving the name of the data array and the subscript of the particular item. In the second case, each variable has its own unique name.

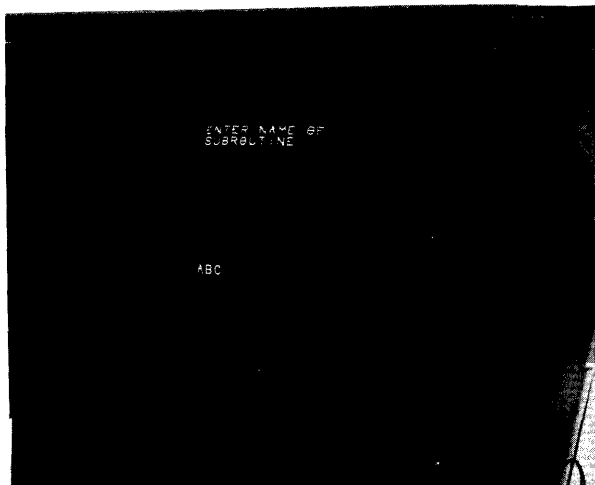


Figure 4. RINSE Subroutine Display.

In addition to the three task requirements mentioned in connection with the RINSE subroutine (i.e., ease of use, visual feedback to permit verification, and an error correction procedure) there was the additional requirement that the operator be able to enter numeric data in either floating point, integer or octal mode. For this task, the only input device which has been implemented is the alphanumeric keyboard.

The subroutine SETDA (*SET* up Data Array) permits the inspection and modification of items in a data array. To use SETDA, the programmer specifies four items of information (see Figure 5): (1) the BCD name of the data array, (2) the length of the data array, (3) the symbolic name (i.e., location in core) of a second array which defines the mode (integer, floating point or octal) of each item in the data array, and (4) the symbolic name of the data array. If each item in the data array has the same mode, the mode need only be specified once for the whole array.

Figure 6 shows the display which will be generated upon execution of SETDA, as indicated in Figure 5.

At this point, the operator can modify the value of the current item, step the display to the next item in the data array, define the subscript of any item in the data array which he wishes to see displayed next, or terminate the subroutine.

The operator enters a new value for the current data item by using the alphanumeric keyboard. The visual feedback and error correction procedures are identical to those described for the RINSE subroutine. Figure 7 shows how the display would appear during this process. The operator can cause the new value of 27 to be stored in DATA(0) by depressing the END key.

```
VECTOR VALUES NAME = DATA14
VECTOR VALUES DATA1 = 2,3,7,77K
VECTOR VALUES MODE = 1,2,3,4
EXECUTE SETDA.(NAME,3,MODE,DATA1)
```

Figure 5. Source Language Statements for SETDA Subroutine.

```
THE CURRENT ITEM IS
DATA1 ( 0)
ITS CURRENT VALUE IS
2
ITS MODE IS
TO DEFINE NEW SUBSCRIPT,
HIT "=" KEY
```

Figure 6. SETDA Subroutine Display.

If the operator tries to store a number of the wrong mode, the display changes to that shown in Figure 8.

If the operator wishes to step the display (Figure 6) to the next item in the data array, he depresses the alphanumeric key labeled "=", and the upper portion of the display changes appropriately. If, on the other hand, the operator wishes to "move" the display directly to any item in the data array, he first depresses the "=" key. The display then changes to that shown in Figure 9 and the operator enters the subscript of the desired item.

If the operator tries to enter a subscript which is too large, the display changes to that shown in Figure 10.

Control is returned to the calling program when the operator depresses the END key from the display indicated by Figure 6.

The second subroutine (SETPAR) developed to facilitate numeric input permits the inspection and modification of a set of distinct data variables. The programmer uses this subroutine in essentially the same manner as SETDA. However, the format in which the current status of the data variables is displayed was altered in an attempt to gain insight and experience in the presentation of information. The information display technique used by the SETDA subroutine can be characterized as the "player-piano" approach: the data items appear as if they were listed sequentially on a long strip of paper which is being moved back and forth past a one-item viewer under the opera-

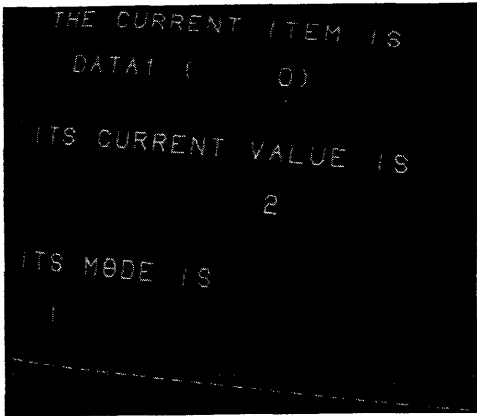


Figure 7. SETDA Subroutine Display.

tor's control. SETPAR uses a "page" approach as the information display technique. This technique treats the data variables as if they were listed, up to 16 to a page, in a loose-leaf notebook. As each "page" is presented to the graphic console operator, he has the options of modifying the values of any data items listed on that page, turning to the next "page", or "closing the book" (i.e., terminating the subroutine).

Figure 11 shows a typical "page" display.

If the operator wishes to change the value of any data variable in the displayed list, he uses

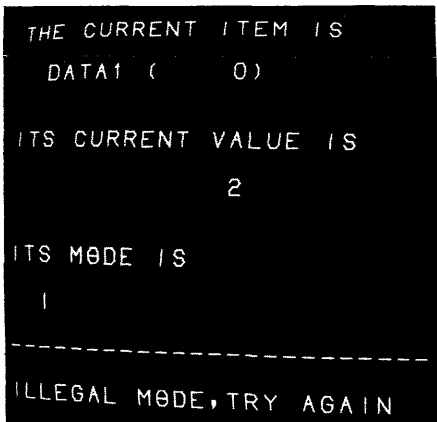


Figure 8. SETDA Subroutine Display.



Figure 9. SETDA Subroutine Display.

the alphanumeric keys labeled "↑" and "↓" to move the pointer to the desired data item and then enters the new value of this item using the alphanumeric keys exactly as in the SETDA subroutine. To "turn the page" the operator depresses the "," key and to terminate the subroutine he hits the END key.

*Positional Data Input*

Since the GM Research Laboratories DAC-I system was developed specifically to investigate the field of graphic data processing, it was necessary to provide a capability for presenting the graphic console operator with a display of graphic information and receiving positional input from him relative to this display. The position indicating pencil provided the basic hardware capability necessary to meet this need. The requirements for this task were: (1) to display the graphic information specified by the calling program, (2) to determine the position of the pencil relative to this display, and (3) to return this position to the calling program when the pencil was removed from the screen.

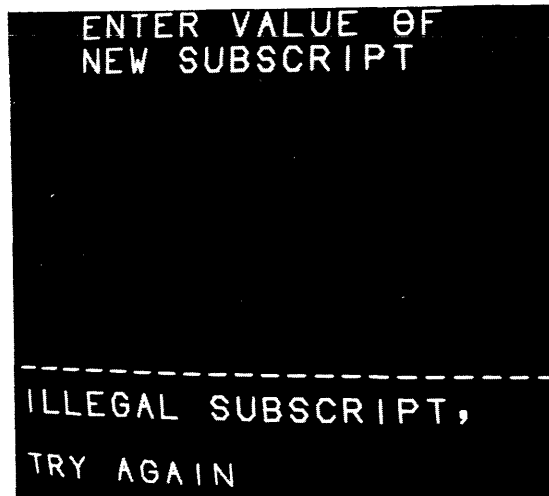


Figure 10. SETDA Subroutine Display.

```

DATA1 =          12      I
DATA2 =           1      F
DATA3 =          3.14159 F
DATA4 =           1      F
DATA5 = 212223242526  K
DATA6 = 000000010000  K
DATA7 =           41      I
DATA8 =           0      F
DATA9 = 000000000000  K
DATA10= 000000000000  K
DATA11=           0      F
DATA12=           0      I
DATA13=           0      F
DATA14=           0      I
DATA15= 000000000000  K
DATA16= 000000000000  K
-----
HIT . . . KEY TO TURN PAGE.
HIT END KEY WHEN DONE.

```

Figure 11. SETPAR Subroutine Display.

Because of the very general nature of graphic information, no attempt was made to assist the programmer in formatting his display. To increase the flexibility of the I/O utility subroutine designed for this task (PEN2) an option was added which allows the graphic console operator to terminate the display by depressing either an alphanumeric key or a program control button. In any case, however, the calling program is fully informed as to the condition which terminated the display.

Figure 12 shows the source language statement necessary to execute PEN2. The 2nd and 3rd arguments define the location and size of the array containing the graphical information. The 4th argument describes the condition which terminated the display. The 1st argument provides information pertaining to the pencil.

Upon execution of PEN2, a display of the specified information begins. At this point, the graphic console operator can depress an alpha-

```
EXECUTE PEN2.(PEN,ARRAY,SIZE,RETURN)
```

Figure 12. Source Language Statements for PEN2 Subroutine.

numeric key, a program control button, or bring the position indicating pencil into contact with the display CRT. If a key or button is depressed, the display is immediately terminated and control passes back to the calling program. If the pencil is touched to the screen, a small cross appears superimposed on the display. This cross defines the position of the pencil relative to the display and provides the operator with visual feedback. The visual feedback is particularly important here because of severe parallax problems. If the pencil is moved across the screen, the cross will follow. When the pencil leaves the screen, the display is terminated and the last location of the cross is returned to the calling program. Figure 13 shows the pencil being used to point to a region on a grid.

This subroutine has proven to be very flexible and easy to use, resulting in a wide variety of applications. One such application of PEN2 was in the development of a higher level I/O utility subroutine permitting the entry of decision type information.

#### Decision Information Input

The task associated with the entry of decision information is to present the graphic console operator with a set of possible actions available to the controlling program and to allow him to select an ordered subset of these actions.

The I/O subroutine CHOICE was developed for this task and requires three items of information from the calling program: (1) the number of alternatives to be presented, (2) the symbolic name (i.e., location in core) of an

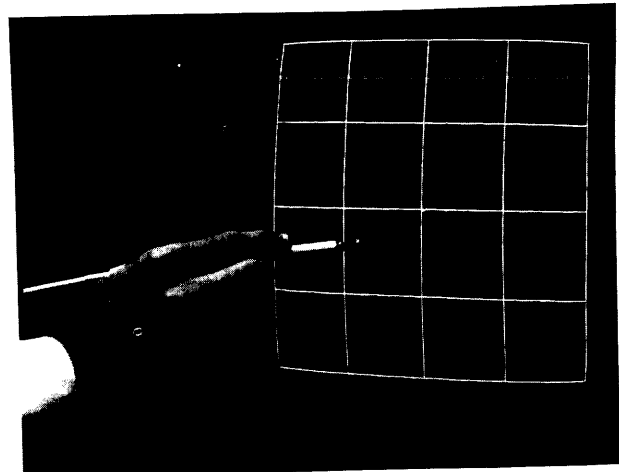


Figure 13. Positional Input Via the PEN2 Subroutine.

array containing the BCD messages describing the alternatives, and (3) the symbolic name of an array provided for the list of selected alternatives (see Figure 14).

The execution of this subroutine begins with a display of the alternatives, as shown in Figure 15.

The operator may now select his first alternative by pointing to the appropriate field with the voltage pencil and then removing the pencil from the screen. Immediate visual verification of the choice is provided by the digit 1 at the right hand side of the selected field as shown in Figure 16.

The operator proceeds in this manner to complete his selection of the desired alternatives. When he is done, he selects the field labeled "DONE" and control is returned to the calling program. If the operator makes a mistake or changes his mind, he can "erase" his last selection by selecting the field labeled BACKSPACE.

SOURCE LANGUAGE STATEMENTS

In the first section of this paper, five subroutines were described which are representative of a class of subroutines whose primary objective is to facilitate the on-line input of various types of information. In the following section, we describe three source language statements which provide the programmer with the capability of generating various types of on-line output in an easy, efficient, and flexible manner. In the development of these statements, an effort was made to conform (insofar as possible) to the precedents established by the normal I/O source statements appearing in the NOMAD language. It was felt that, in so doing, the resulting statements would be much easier for the programmer to use.

Display Format Statement

"DISPLAY FORMAT" was designed to permit the programmer to use the graphic console

EXECUTE CHOICE.(ALTERS,5,SELECT)

Figure 14. Source Language Statement for CHOICE Subroutine.

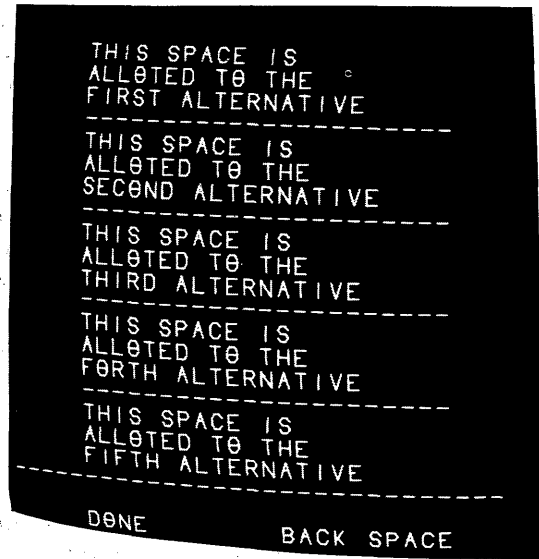


Figure 15. CHOICE Subroutine Display.

display CRT in much the same way as the on-line printer was used in "the good old days." The implementation of this "on-line print" capability provides the programmer with a very convenient way of presenting a wide variety of information to the graphic console operator.

The maximum amount of information which can be displayed by DISPLAY FORMAT (i.e., its unit record) is 480 characters. These 480 available character positions are in the form of 20 lines on the CRT with 24 character positions per line.

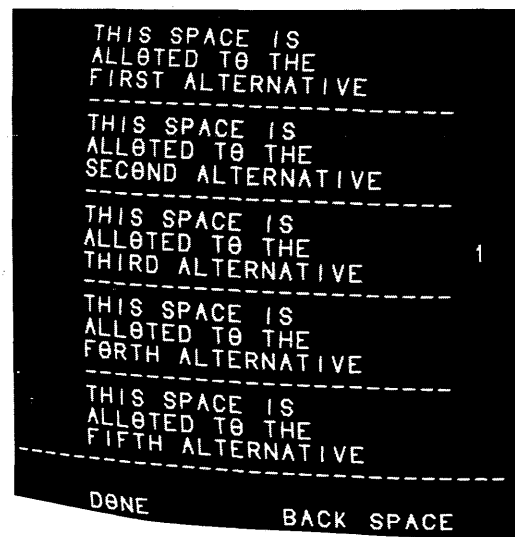


Figure 16. CHOICE Subroutine Display.

The DISPLAY FORMAT statement has a list and a format associated with it. The list gives the values and items to be displayed and the format gives the display pattern to be used. The format is composed of a sequence of format specifications of the form  $Cw$ . The character  $C$  is the control character and defines the type of operation or conversion to be performed. In general,  $w$  refers to the number of character positions associated with the operation defined by  $C$ . If, in any format specification, the field width  $w$  is zero, that format specification and its associated list item (if any) are ignored.

There are three types of format specifications available for the conversion of numeric data. Octal and decimal integers are generated by using the  $Kw$  and  $Iw$  specifications respectively. The  $Fw$  specification provides for floating point variables which will be displayed as either floating point decimals (e.g.,  $.2E-3$ ) or fixed point decimals (e.g.,  $.0002$ ) depending on which form produces the fewest characters for display.

There are two types of format specifications available for the conversion of BCD information. The  $Aw$  specification is identical with that used in other NOMAD I/O statements. In particular, one word (six characters) will be picked up from the list and right justified or truncated on the right, depending on whether  $w \geq 6$  or  $w < 6$ . The  $Hw$  specification differs from ordinary I/O statement usage in that the Hollerith information is specified in the list rather than in the format.

The specification  $Xw$  provides  $w$  blanks in the display. The specification  $/w$  results in a downspace of  $w$  lines. An automatic downspace of one line results when the number of characters on a line reaches 24. The specification  $Pw$  ( $w$  has no particular significance in this case unless it is 0) resets the current character position to the first character position of the first line. This specification has proven very useful when putting out columns of information which comes from several different arrays in memory.

The specification  $Rw$  causes the succeeding portion of the format to be repeated, if necessary, until the list is exhausted ( $w$ , again, has no particular significance unless it is 0). If the specification  $Rw$  does not appear anywhere in

the format, the entire format will be repeated until the list is exhausted.

The form of the "DISPLAY FORMAT" statement and a sample format is shown in Figure 17.

Perhaps the most radical departure of the DISPLAY FORMAT statement from standard NOMAD I/O statements involves the use of its second "argument" (i.e., the item called INOUT in Figure 17).

INOUT will, in general, be the first cell of a short array in which the program using DISPLAY FORMAT can:

- a) specify the status light configuration during and upon termination of the display,
- b) specify the input devices which the graphic console operator will be permitted to use in terminating the display.
- c) receive a description of the condition which caused the termination of the display.

The graphic console display shown in Figure 18 corresponds to the DISPLAY FORMAT statement of Figure 17.

Because of its flexibility and ease of use, the DISPLAY FORMAT statement has been a valuable aid in the development of a cooperative man-machine problem-solving system.

#### Record Format Statement

The RECORD FORMAT statement provides a means of producing hard copy alphanumeric output through the use of the recording feature of the image processor. The form of the RECORD FORMAT statement is identical to that of DISPLAY FORMAT (i.e., RECORD FORMAT FMT, INOUT, L1, L2, L3, . . .). RECORD FORMAT recognizes all of the format specifica-

```
VECTOR VALUES L1 = $CROSS LOCATIONS$
VECTOR VALUES X = $X$
VECTOR VALUES Y = $Y$
VECTOR VALUES LEVEL = $LEVEL$
VECTOR VALUES FMT = $(X5,H15,/1,X3,H8,X4,F6,/1,
1 X7,H1,X5,H1,X3,H5,R5,/1,X5,I4,X2,I4,X4,I2)$
DISPLAY FORMAT FMT,INOUT,L1,DATE,TIME,X,Y,LEVEL,
1 DATA(1)....DATA(15)
```

Figure 17. Source Language Statements for DISPLAY FORMAT.

```

CROSS LOCATIONS
06/24/64      2017.4
  X      Y      LEVEL
 206    179      | |
3955    216      | |
 189    3943      | |
3933    3950      | |
2083    2065      | 5
    
```

Figure 18. Application of DISPLAY FORMAT.

tions and conventions used by DISPLAY FORMAT.

The unit record for RECORD FORMAT is 12,000 characters per film frame. These 12,000 available character positions are in the form of 100 lines of 120 character positions each.

With RECORD FORMAT, INOUT is a single cell and is used by the calling program to specify the desired film train operations associated with the recording. Appropriate bytes in INOUT define which film train is to be used, how many frames are to be advanced before and after the recording is performed, and when developing of the film frame is to occur (if at all).

RECORD FORMAT enables the programmer to use the image processor as he would use an off line printer with the added advantage that his turnaround time is measured in minutes rather than hours. The output can be developed immediately and moved to the project station of the image processor for on-the-spot review and then placed on an aperture card for hard copy reproduction. The hard copy output is very similar to that produced by an IBM 1403 printer.

*Generate Format Statement*

The DISPLAY FORMAT and RECORD FORMAT statements enable the programmer to easily produce alphanumeric output. However, it is frequently necessary to produce either displays or recordings which contain a combination of graphic data and alphanumeric characters of different sizes (see Figure 19). In addition, it is desirable to have the capability of altering a small part of a display without having to regenerate the whole display. The value of such a capability is best illustrated in

the operation of the CHOICE subroutine (see Figures 15 and 16). The GENERATE FORMAT statement was specifically designed to meet these needs.

With respect to formats, input lists, and statement form (i.e., GENERATE FORMAT FMT, INOUT, L1, L2, L3, . . .), GENERATE FORMAT is identical to the previous two statements. However, instead of producing output directly (as is the case with DISPLAY FORMAT and RECORD FORMAT), GENERATE FORMAT provides the programmer with the array of CRT coordinates needed to produce a display of the alphanumeric information. In addition, GENERATE FORMAT enables the programmer to specify character size and line spacing (i.e., the user can define his own unit record).

By executing GENERATE FORMAT several times with different unit record specifications and combining the resultant arrays of CRT coordinates with an array of coordinates representing graphic data, the programmer can, through the use of a basic display or recording subroutine, produce output of the type shown in Figure 19.

The special features of GENERATE FORMAT were utilized in the I/O utility subroutines SETDA, SETPAR, and CHOICE to quickly and efficiently produce sequences of displays in which only portions of the basic display are altered. For instance, Figure 16 can

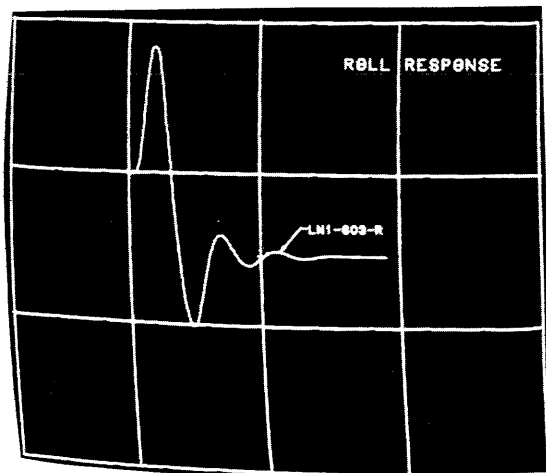


Figure 19. Application of GENERATE FORMAT.

be produced from Figure 15 simply by adding the three coordinate pairs necessary to generate the character 1 to the large coordinate array for the basic display.

The DISPLAY FORMAT and RECORD FORMAT statements provide the programmer with an easy and efficient method of utilizing the output capabilities of the 7960 system. The GENERATE FORMAT statement rounds out the output software package by providing the additional flexibility needed to handle the situations described above.

#### SUMMARY

Any attempt to implement a system which stresses cooperative man-machine interaction must concern itself primarily with the problem of man-machine communication. Furthermore, the man-machine communication will typically take place in a wide variety of situations and at many levels of problem discourse. Attempts to define all possible communication situations and develop specific programs for each will lead to a tremendous amount of redundancy and duplication of effort. It is possible to avoid this problem by realizing that all man-machine communication must pass through the I/O hardware interface. Thus, by providing an extensive, flexible, and powerful I/O software capability, the presence of the hardware interface need not concern the programmer and he can devote his whole attention to the more significant aspects of man-machine communication. The software described in the body of this paper has proven to be an immensely powerful tool which has enabled programmers to rapidly design and evaluate the wide variety of communication techniques necessary to any man-machine problem-solving system.

#### ACKNOWLEDGEMENTS

The material described in this paper is not solely the work of the authors. Significant contributions were made by other members of the programming staff. In particular, we would like to acknowledge the contributions of C. Richard Lewis of the Computer Technology Department, GM Research Laboratories, and Martin Weinrich.\*

#### REFERENCES

1. JACKS, E. L., "A Laboratory for the Study of Graphical Man-Machine Communication," *Proceedings Fall Joint Computer Conference*, San Francisco, California, Oct. 1964, (this volume).
2. COLE, M. P., DORN, P. H., LEWIS, C. R., "Operational Software in a Disk Oriented System," *Proceedings Fall Joint Computer Conference*, San Francisco, California, Oct. 1964, (this volume).
3. HARGREAVES, B., JOYCE, J. D., COLE, G. L., FOSS, E. D., GRAY, R. G., SHARP, E. M., SIPPEL, R. J., SPELLMAN, T. M., THORPE, R. A., "Image Processing Hardware for a Man-Machine Graphic Communication System," *Proceedings Fall Joint Computer Conference*, San Francisco, California, Oct. 1964, (this volume).
4. KRULL, F. N., FOOTE, J. E., "A Line Scanning System Controlled from an On-Line Console," *Proceedings Fall Joint Computer Conference*, San Francisco, California, Oct. 1964, (this volume).
5. FREEMAN, H., "On the Encoding of Arbitrary Geometric Configurations," *IRE Transactions on Electronic Computers*, June, 1961.

\* Now at the University of Michigan.



# A LINE SCANNING SYSTEM CONTROLLED FROM AN ON-LINE CONSOLE

*Fred N. Krull and James E. Foote*

*Research Laboratories, General Motors Corporation, Warren, Michigan*

## INTRODUCTION

Direct graphical input is one of the newest and most exciting sources of digital computer input. Programming techniques and hardware are beginning to appear which are designed to automatically process graphic information.<sup>1, 2, 3, 4</sup> This paper describes an experimental system which has been designed to facilitate the digitizing of line images. The equipment which is used for this purpose is an IBM 7960 Special Image Processing System, consisting of graphic console, image processor, and a modified data channel.<sup>7</sup> The image processor (Figure 1) contains a programmable Cathode Ray Tube (CRT) scanner, a CRT recorder, a 35mm camera, and film processing equipment.

The graphic console (Figure 2) contains devices to communicate with the operator and to control the image processor.

Both of these devices are tied directly to a computer via a modified data channel.

The principal objective of this portion of DAC-I (Design Augmented by Computers) project<sup>5, 6, 7, 8</sup> was to utilize the full capabilities of the image processor, graphic console, and digital computer to digitize a variety of line images to a high degree of accuracy. Program system objectives dictated that line widths .05% of image size (.01 inches on 20x20 inch document) must be detectable to an over-all system accuracy of  $\pm .05\%$  of image size. Thus, this system was designed to provide a rapid,

versatile, and accurate method for converting graphical data to digital form. While it provides only for the digitizing of lines, a natural outgrowth of the work will be a library of pattern and character processors to be used by programmers in much the same manner as card and tape input/output (I/O) routines are used now.

The application requirements of the DAC-I project dictated that one of two approaches be used for the analysis and processing of a two dimensional graphical form:

- a) Algorithms are specified which prescribe the rules for analyzing a graphical image. This approach assumes a structure for each characteristic image, and relies upon the availability of individual pattern processors to detect image components.
- b) No fixed structure is assumed for the input document. Minimal restrictions are placed on image quality only. Emphasis is placed upon providing elementary functions which an operator may call upon and combine in order to process a complex form. All decision capability and selection of functions is left to the operator who is controlling the scanning device from an on-line console.

The second approach was selected since at this point in the project it was not possible to specify the structure of the large variety of input documents anticipated. We felt, for instance, that the system would be used to digitize

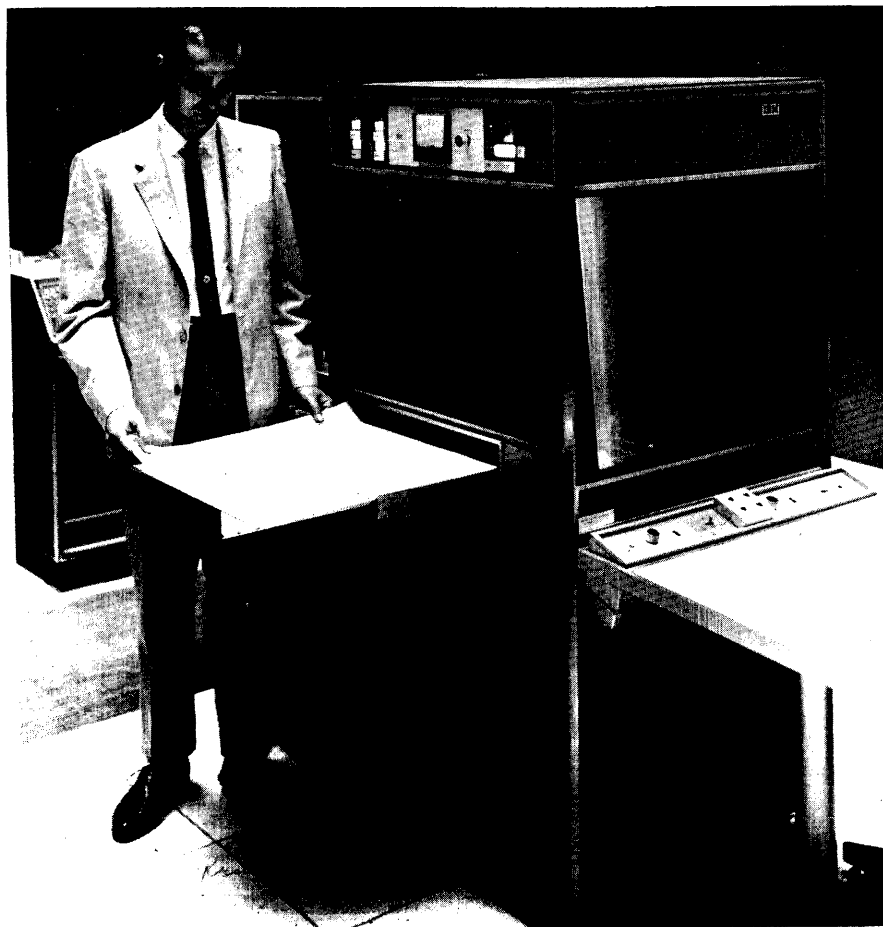


Figure 1. Image Processor.

documents ranging from mathematical graphs to engineering drawings. Therefore, it seemed advisable to concentrate our efforts on providing a reliable set of basic pattern analyzers. These elementary functions would then provide the basis for processing much more complex forms.

Since the console operator now becomes an essential part of the scanning sequence, it was necessary to devote considerable effort to the problem of man-machine communication. Development of the system has permitted us to carry out a series of experiments, designed to discover the best interface between man and computer. Scanning functions have been made automatic when practical, but the decision capability of the human operator is still used to best advantage. Thus, flow diagrams for scanning functions in many instances contain a block in which the operator is controlling the selection

of a branch or setting the value of a parameter. Each situation has warranted an investigation to answer the question: "Who can perform the job better—man or machine?"

An essential part of this process was to economically match the speed of the computer with the speed of man. The solution was to multiprogram a 32K digital computer being used to process regular compilations and executions under batch monitor control.<sup>5, 6</sup> A 16K-16K logical core split was made available for use by the on-line operation and the batch monitor programs\*. The on-line console operation has millisecond access to the CPU of the computer on a demand basis. When use of the CPU is no longer required, control is returned to the batch monitor program.

\* This configuration has been updated to a 64K memory with a 32K-32K logical core split.

A dynamic storage allocation system and execution processor<sup>6</sup> occupy almost half of the 16K memory cells devoted to on-line console operation. The remainder of the core space (approximately 8K) is available for use by the scanning system programs and data tables. The scanning programs total more than 40K cells, exclusive of memory data tables. Hence, the dynamic storage allocation scheme is utilized to overlay subroutines as they are required. The entire scanning system was written in an algebraic language called NOMAD and a channel language called MAYBE.<sup>6</sup>

All functions are made available to the operator via program control buttons on the graphic console. There is almost a one-to-one correspondence between operational capability and program control buttons. The organization of this type of system is illustrated in schematic form in Figure 3.

This concept allows the operator to depress a button and execute a wide range of functions. Upon completion, the system may return to standby and wait for the next selection, or each function may automatically call on other functions. For example, an operator may depress the DISPLAY button to obtain a graphical im-

age on the graphic console CRT. Similarly, a scanning function may display intermediate results by *logically* depressing the DISPLAY button. Thus, one function may logically depress other program control buttons. In addition, while one function is being performed, it may be advantageous to make other functions optionally available. Thus, function (B) can allow program control buttons (A) or (C) to be depressed, and subsequently signal the control program which option has been selected.

The functions themselves logically fall into four distinct areas: Film Operations, Scanning Operations, Display and Review Operations, and Modify and Store Operations. Each area presented the same problem; namely, how to best communicate with the man in order to perform a specific operation. The variety of solutions that have been employed are discussed in the remainder of this paper.

## FILM OPERATIONS

A line scanning problem will, in general, require more than one film frame (i.e., more than one input document) and the user may wish to use either the exposure and rapid processing facilities of the image processor or off-line film



Figure 2. Graphic Console.

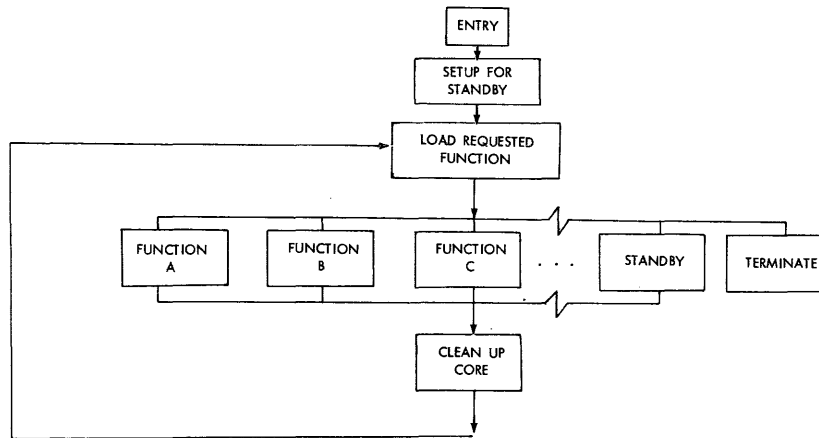


Figure 3. System Organization.

which has been pre-exposed and may or may not have been processed. During the course of the scanning operation, the user may at appropriate times wish to record current results on film either for purposes of verification or to form a permanent record of his work.

It was therefore necessary to include in this line scanning system a facility for the console control of both of the film trains<sup>7</sup> in the image processor which is shown schematically in Figure 4.

One film train (train B) may be used only for recording and reviewing output. The other film train (train A) may be used either for recording or for exposing input documents onto film and scanning the resulting film images. Both film trains can process an exposed film frame in approximately 30 seconds and project the image onto the 22x20 inch viewing screen.

The 22x22 inch paper input station can accept documents for exposure onto raw film in train A. Alternatively, pre-exposed microfilm may be inserted in the supply cassette. Film transport commands allow the film to be advanced into buffer 1, developed, and advanced or backspaced into buffer 2 under program control.

The console operator is provided with several film functions which make use of these facilities. An exposure processing function readies the paper input station to accept documents. The operator may then insert documents and make exposures by depressing an EXPOSE button on the side of the image processor. Each exposure is automatically advanced into buffer 1. A count of the number of exposures is displayed on the graphic console screen. When all exposures have been completed, the operator

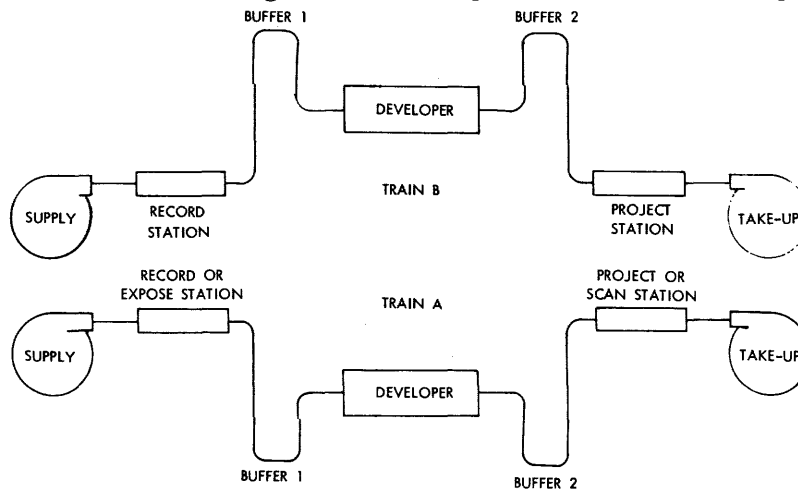


Figure 4. Film Train Configuration.

may initiate the processing cycle. The film transport is programmed to develop all of the exposed film in buffer 1 after adding a trailer to the exposures, and position the first exposure at the scan-project station. The operator may utilize manual controls on the image processor to advance or backspace frames that are in buffer 2 or on the take-up reel. Operational experience has shown the value of providing the console operator with a rapid means of preparing a film image suitable for scanning. A certain amount of flexibility and film quality is lost through the utilization of rapid processing, but this seems to be more than offset by the convenience of short turn-around time.

An auxiliary film processing function is available if off-line film is to be used. Utilization of pre-exposed film allows for a wider variety of input quality and document sizes. This film is inserted in the supply cassette and spliced to the film in train A. When the processing is initiated, both film buffers are emptied (i.e., the film is pulled tight). They remain empty as the film passes from the supply cassette, through the developer and then past the scan-project station. The graphic console operator can monitor the operation on the projection screen. Depressing the END key on the alphanumeric keyboard will halt the developing. In this manner rapid processing film which has been pre-exposed can be developed. Standard microfilm or rapid processing film that has been preprocessed is not significantly affected by passing through the developer during this operation. The use of preprocessed high contrast microfilm permits the scanning of material which is of much poorer quality than can be accepted at the paper input station.

During the course of various scan operations, it is frequently necessary to make a permanent record of the results. Selection of a recording function causes all current scan results to be recorded on film train B. The material recorded on film is the same as that which is shown on the graphic console by the DISPLAY function discussed later in the paper. After recording is completed, the frame will be automatically advanced into the first film buffer of train B. The developing process will be initiated automatically after six recordings but can be initiated at any time via a FILM CONTROL func-

tion. By means of manual controls on the image processor, a processed frame can be centered on the viewing screen. By appropriate manipulation of both film train projection lamp rheostats and positional controls (2 dimensional motion is possible on train B), both the scan results on train B and the original document on train A can be superimposed on the viewing screen for purposes of comparison.

The user also has at his disposal the ability to develop or clear film which has been moved into any of the film buffers. Clearing the film moves all exposures onto the take-up reel from which they can be removed and mounted in aperture cards. One of the accessory pieces of equipment available to the user is an aperture card printer. Thus, hard copy output is available to the user in a matter of minutes after he leaves the console. We find that this is particularly valuable in evaluating results and maintaining a record of work accomplished.

## SCAN OPERATIONS

The unique feature of this line scanning system is that only those elements of an image which are selected by the console operator will be scanned and digitized. The basic element of an image is a line segment defined as a continuous curve terminated by two ends, two junctions or a combination of both. Since complicated images may contain many intersecting curves, a line will, in general, consist of several segments which must be added together logically. Experience has shown the junction to be a far more accurate delimiter than the end of a curve. The normal procedure of this system has been, therefore, to define the end points of lines precisely by means of perpendicular slash marks. Figure 5 shows a typical document from which line AD is to be digitized. Line AD consists of segments AB, BC, and CD. A line may, of course, consist of only one segment.

The requirement of digitizing selected lines which represent only a small fraction of the entire image suggested a line tracking technique. Analysis of a raster scan of the entire image area was deemed impractical because of the volume of data involved.

The automatic line tracking procedure which forms the heart of this line scanning system is described briefly below. Figure 6 shows a typi-

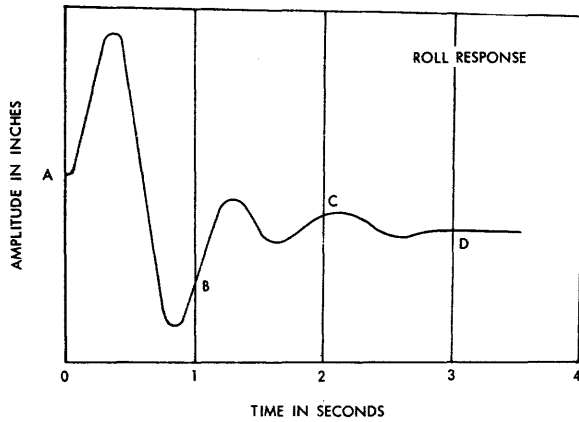


Figure 5. Sample Document.

cal line with two points ( $P_A$  and  $P_B$ ) that have been sampled. The next point is estimated to be  $P'_C$ : a distance ( $\Delta H$ ) away from  $P_B$  on the line defined by points  $P_A$  and  $P_B$ . Two "scan feeler vectors" ( $P_1-P_2$ ) and ( $P_3-P_4$ ) are then plotted on the CRT parallel to and a distance ( $\Delta V$ ) away from the line ( $P_B-P'_C$ ).  $\Delta H$  and  $\Delta V$  are specified by the console operator.

If nothing is encountered by either feeler vector, a scan vector ( $P_2-P_4$ ) is plotted and the next point  $P_C$  is determined by the intersection of ( $P_2-P_4$ ) with the line as shown in Figure 7.

The procedure is repeated until the line ends or one of the feeler scan vectors gets a "hit" indicating that either a junction has been encountered, or that the tracking step size ( $\Delta H$ ) and the feeler vector spacing ( $\Delta V$ ) are not compatible with the curvature of the line. When this occurs, a block of code will be called upon to analyze the situation, and action will be requested from the operator if necessary.

At each point, the threshold level (detection sensitivity of the scanner) is adjusted within rigid bounds, based on results at the last point,

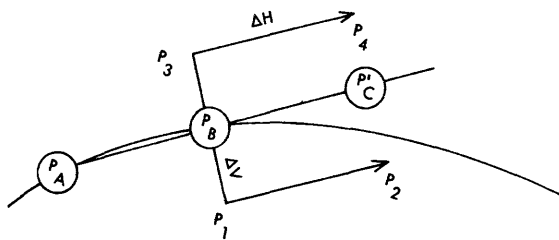


Figure 6. Line Tracking Procedure.

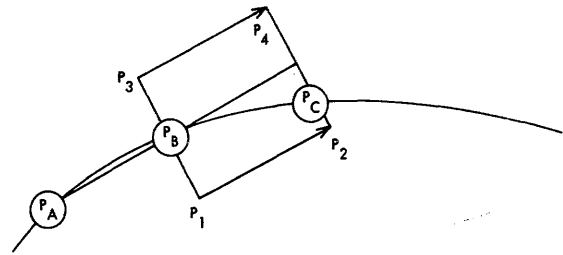


Figure 7. Line Tracking Procedure.

to the minimum necessary for line detection. When a new point is obtained, the threshold level bounds are modified if necessary. In this manner, large variations in density across a film frame can be accommodated while still preventing erroneous scanner responses due to improper threshold level. The threshold level bounds, along with tracking step size, feeler vector spacing and several other scanning parameters can be modified if necessary by the user at the console via a CHANGE PARAMETERS function.

The automatic method of line tracking is utilized by the user whenever possible. If areas of difficulty occur, the user may request or the scan program will automatically generate a call for a raster sweep (TV scan) of the area of difficulty. The user may then utilize the sweep display on the graphic console screen for diagnostic purposes.

The graphic console operator is required to combine those functions which will enable him to process a complicated image. It is assumed that a film frame has been centered in the scan gate before any scan functions are selected. A RESTART function readies the system to accept results from a new image. All previous scan results are deleted.

A REGISTRATION function may be used to search for a border around the image corresponding to a 20x20 inch square border on the input document. The main function of this border is to provide an easy means of supplying the scanning system with coordinate and scaling data. The coordinates of the border are assumed to be (0,0), (0,20), (20,20), (20,0) but can be modified by the console operator by using a CHANGE PARAMETERS function. If registration is successful, the border, plus lines dividing each side of the border into quarters, will

be displayed on the graphic console CRT. This grid aids the console operator in using the position indicating pencil to select lines for scanning. It is not necessary to register, however, in order to proceed with scanning since there are alternate methods for the system to obtain coordinate and scaling data.

A SCAN A LINE function readies the system to begin scanning the first segment of a new line. The user will immediately be requested (via appropriate comment on the graphic console screen) to select by means of a position-indicating pencil the approximate area for the scanning to begin. The user may then point directly to an area on the screen (using the display of previous results and the registration grid as a guide, if available). Alternatively, the console operator may request a gross raster scan and display of the entire image. He may then select a line with the pencil as shown in Figure 8.

Once a starting area has been supplied, a search will begin for two points on the first segment. These points will then be used to initiate the automatic line tracking procedure which will track to both ends of the segment in two steps. If any difficulty is encountered during this operation, the results up to that point

along with an appropriate comment and a box indicating the region of trouble will be displayed. The operator must then take the proper action which usually begins with a TV sweep of the problem area. After a portion of a line has been scanned, an ADD A SEGMENT function may be used to add segments to the given line. Selecting this function (active only after the first segment of the current line has been scanned) readies the system to scan a segment *adjacent* to one end of the current line and add it to the current line.

The user is immediately requested to indicate with the position-indicating pencil the approximate location and initial slope of the next segment. A search will then be initiated for a point on the next segment which is separated from the endpoint of the current line by a distance equal to the tracking step size ( $\Delta H$ ). This point and the endpoint will then be used to initiate line tracking which will proceed to the end of the segment. If any difficulty is encountered during this operation, all previous scan results, plus an appropriate comment and a box indicating the problem area, will be displayed. The user must then take appropriate action. If this entire operation is successful, the segment will be added logically to the current line to form the new current line.

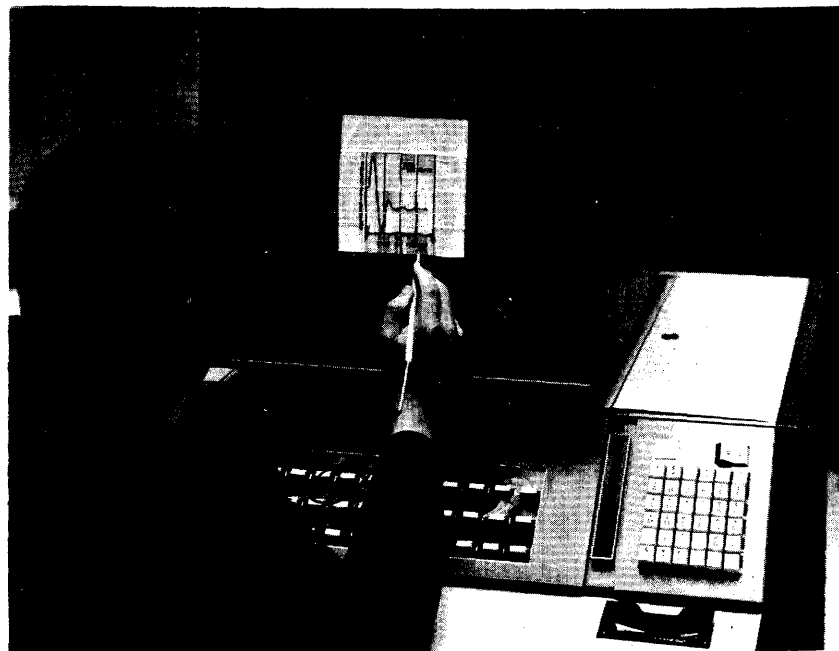


Figure 8. Line Selection.

A TV SWEEP function allows the user to examine a localized area for diagnostic or corrective purposes. This function will be initiated automatically whenever the tracking procedure encounters difficulty. The console operator may also use this function to investigate any data point along a line by selecting the point of interest with the position-indicating pencil. The scan responses are analyzed and a display is generated which nominally fills the entire graphic console screen. This enables the user to observe a magnified view of a localized area on the image (as it appears to the scanner). Figure 9 shows a TV sweep display of a typical junction. The size of the sweep area is approximately .5 x .5 inches on a 20x20 inch document.

The "X" indicates the center of the scan raster. The user may adjust the threshold level or change the location of the scan raster via appropriate alphanumeric keys. The raster may also be moved by indicating a desired location with the pencil. A new raster scan and display will then be generated. The system understands that the position-indicating pencil position superimposed over the TV sweep display refers to the corresponding position in the actual scan raster. Other appropriate system operations may be initiated from this mode at any time.

If a diagnostic study reveals that automatic line tracking is impractical for some reason, coordinate points may be stored in conjunction

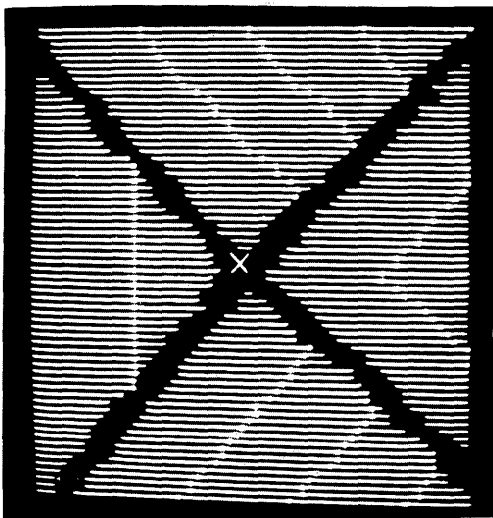


Figure 9. Junction of Two Lines.

with the TV sweep facility. This function allows the user to add the current center of the scan raster as a data point and thereby move manually over an area of difficulty before resuming automatic tracking. Under extraordinary circumstances (e.g., very poor film, poor document quality, or an extremely congested image), an entire line can be digitized in this manner.

The CHANGE PARAMETERS function allows the user, through a combination of multiple choice and alphanumeric keyboard responses, to change many of the scan parameters. The value of each parameter is displayed on the graphic console CRT along with suitable descriptions (Figure 10). Through the use of a multiple choice sequence employing the position-indicating pencil, the user may select a parameter and type in a new value for that parameter. This function allows the user to materially change the performance of a subroutine from the console. These parameters are used within the programs both as numeric constants and as switches for branching operations.

#### DISPLAY AND REVIEW OPERATIONS

The graphic console 10x10 inch CRT is the principal medium by which the computer communicates with the user. Throughout the sys-

```

ENTER NEW VALUES
HØRZ. ØRIGIN=      0 ←
VERT. ØRIGIN=      0
BØRDER SIZE =    20.
SEGMENT AREA =    50.
LINE AREA   =   200.
MINIMUM STEP=    25.
MAXIMUM STEP=   400.
EPSILON     =      2.
RESØLUTION  =    25.
SCAN RADIUS =    50.
CURRENT TL  =    35.
MINIMUM TL  =    30.
MAXIMUM TL  =    40.
START SWITCH=      1.

```

Figure 10. Parameter List.



tem, messages are continually being displayed on the CRT to which the user must respond. Although this display is low in accuracy with respect to the scanner or recorder, it is also utilized for the purpose of displaying scanned results. A display of the scan results may be used to check for completeness and to monitor the progress of the scanner. To some degree, the accuracy of the scanned results may also be checked at the graphic console.

As a film image is scanned, the digitized coordinate points are stored in an in-memory table. The structure of this table is quite simple. Each element of scanned data is stored as a linear array, with suitable identifiers and pointers to the next element of data. The scan operations may at any point in their logical sequence call upon the display operation to generate a pictorial representation of the in-memory scan data. A typical display is shown in Figure 11.

If an error condition has occurred during some previous function, the display operation may be so signalled. This will result in a small box being superimposed over the location of the error, plus addition of a suitable error comment (Figure 12).

After the scan results appear on the graphic console CRT, the following review operations become available.

- a) Increase Scale
- b) Change Mode
- c) Identify Dimensions
- d) Identify Coordinates

By selecting the INCREASE SCALE function, the size of the display will be doubled.

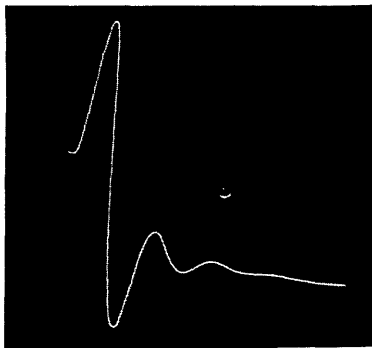


Figure 11. Display of Scan Results.

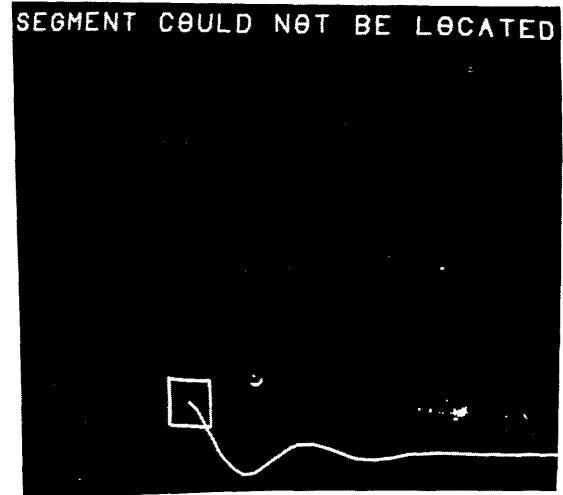


Figure 12. Error in Scanning.

Any scan data which falls outside of the field of the CRT is deleted by appropriate programming. Reselection of a display returns the scan results to their original scale. The CHANGE MODE function may be used to display all lines either as a continuous curve or as a series of crosses. The latter mode allows the user to ascertain the number and location of each single digitized point. The ability to view individual scanned points is essential when the scanner is being used to capture a minute feature of a line.

If some portion of the display outside the field of view is desired, the pencil may be used to point to the edge of the field. Removing the pencil from the screen will cause the scanned data to be redisplayed with the selected point relocated at the center of the field of view. Figure 13 illustrates a display in which the user has reviewed his scanned results by centering the field on the end of a line and magnified the scale four times to examine an end condition in detail.

Within the operation of the scaling and field centering mechanism can be found a lesson in man-machine interaction. A variety of functions can be selected subsequent to a display. Typical of these is the LINE SCAN function. After this function is selected, the user is requested (by comment on the graphic console screen) to use the pencil to point to a location on the screen. Thus, the sequence of operations was to select a function (by depressing a

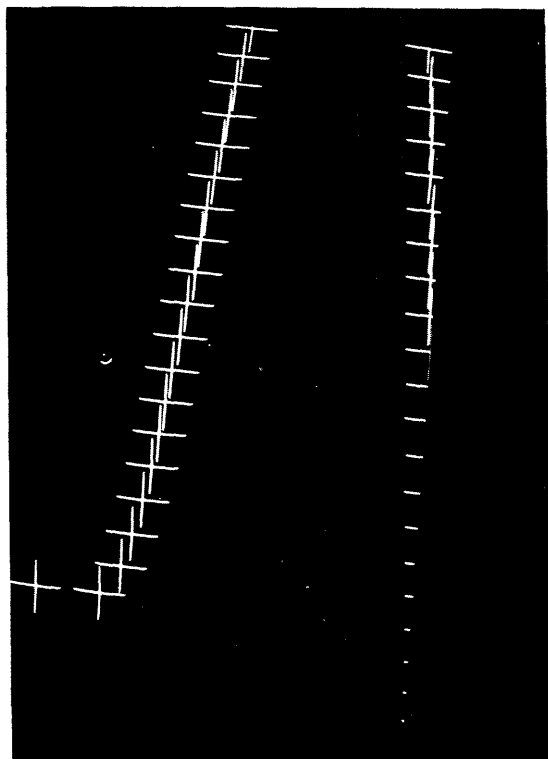


Figure 13. Magnified Display.

program control button) and *then* point. Our initial operational experience taught us that the more natural instinct was to point and then select an operation. Similar instances of man-machine interaction being uncomfortable were noticed throughout the development of the system. In most cases, the problem was rectified after user and programmer discussed alternative sequences of operation. Thus, many operations communicate with the man in more than one way, thereby anticipating the variety of ways in which a man may respond to a given situation.

While in review mode, the user is also frequently concerned with the accuracy of his results. The system provides two means for reviewing the accuracy of the scan data before storing the data on disk. If a square border has been registered from the film image (1.09x1.09 inches on film), all interior points are related to the coordinates of the corners of the border through the use of a four point perspective transformation. At the paper input station, the border would normally appear 20x20 inches square. The user through the use

of the CHANGE PARAMETERS function may define the coordinates of the corners of the border. Thus, the user may select any point interior to the border and receive the inch coordinates of that point. The point may be selected by pointing with the position-indicating pencil or by using linear arrows on the alphanumeric keyboard to vernier a pointer.

If no border registration is available, the only accuracy test which can be made is to check the chordal distance between two points. For example, if the scanner were used to digitize a sine wave consisting of two cycles, the user could measure the chordal distance in inches between the ends of the cycles on the original document. An IDENTIFY DIMENSION function allows the user to use the pencil to point to *two* locations on a line and receive a display of the chordal distance between the two points. In this case, the program searches for the two points on the scanned line nearest to the two locations selected by the user. The program displays the distance assuming an 18.33 reduction ratio between document and film image.

#### MODIFY AND STORE OPERATIONS

The development of an elementary set of modify operations was a direct result of monitoring the use to which various users put the system. The most frequent request was for operations to delete erroneous data. Here again let us emphasize that the scanner and computer are utilized to perform the elementary operations while the user (through graphic console displays) retains judgment as to the correctness of the results.

A typical situation might be that the scanner began to digitize a scratch on the film rather than a line on the film image. The requirement for data deletion operations led to three methods for deleting data. The console operator may select a DELETE LAST SEGMENT function and cause the *last* element of scanned data to be deleted from in-memory storage. An entire line or a single point may be deleted by pointing to a location on the graphic console screen with the pencil and selecting the desired operation. In these two cases, the item of scanned data, be it a line or point, nearest to the selected location is deleted from in-memory storage.

Another frequently requested operation was the ability to add intermediate points to a display of scan results. Because of its finite sampling capability, the scanner may miss a particularly critical point on a line. In these cases, the user may point at the display with the pencil and cause a coordinate point to be inserted in the scanned results at that point. A high degree of accuracy may be obtained by adding coordinates in this manner, particularly if the scale of the image is enlarged before any operation is attempted.

After all scanning, review, and modify operations have been completed, the user can choose to save these results by storing them on a random access disk file. For our purposes, each digitized line is assigned a unique data name (e.g., LN5-1-537-R) which determines its storage location on disk.<sup>7</sup> Any other application-oriented program may have access to this data by referring to the same data name. Library I/O subroutines are available to the programmer for storing and retrieving data from the disk.

The function which stores data on disk provides the facility for transforming the digitized points to any desired coordinate system. The manner in which this is accomplished is for the user to provide the *desired* coordinates of two of the scanned points (usually the left and right end point). This data enables the STORE function to compute a linear transformation between the raster unit coordinate system of the scanner and the desired output coordinate system.

Since the STORE function requires alphanumeric input, we were interested in the variety of ways in which a man could communicate this data. The devices which could be made available for the transmission of alphanumeric data are as follows:

- a) Alphanumeric keyboard used as a typewriter
- b) On-line card reader
- c) Scan and recognition of characters in the field of the image
- d) Writing with the position-indicating pencil on the face of the graphic console CRT

- e) Multiple choice operations by pointing with the position-indicating pencil on the face of the graphic console CRT

The options which are currently available are (a), (b), and (e). Our experience leads us to believe that (d) is not practical, simply because of the ease of using a typewriter and monitoring the message on the face of a CRT. Scanning and recognition of character sets on the film image is obviously very desirable. This work is under current development but will not be reported on in this paper.

When a line or series of lines have been scanned, reviewed, and modified, the console operator may elect to save the results by choosing the STORE function. At this point, all scanned data is corrected according to a table of calibration data stored on disk each morning by maintenance personnel. The calibration data gives a measure of the distortions in the scan raster. This data is obtained by scanning a high accuracy metal target. The calibration data may then be used to correct subsequent scan results.

After correction, all extraneous coordinate points (those lying within an epsilon of a straight line) are removed from the line. Resultant lines are then displayed singly on the graphic console CRT. A typical display is shown in Figure 14.

The density of points will be a function of line curvature. The user then has a multiple choice option in which he may choose to utilize card data, typed data, or border data for supplying coordinate information relative to the line. The pencil must be used to point to one of the fields in order to select the appropriate mode of input. We have found that when processing a volume of information, the users will prepare data cards ahead of time and utilize this mode of alphanumeric input. If only one or two lines are to be processed, the typed input will more likely be selected. In this case, the STORE function leads the user through a series of questions and responses, in which he is either required to make a choice or type in a reply at the alphanumeric keyboard. In all cases, a message on the graphic console CRT instructs the user as to the next operation or required response. As an added

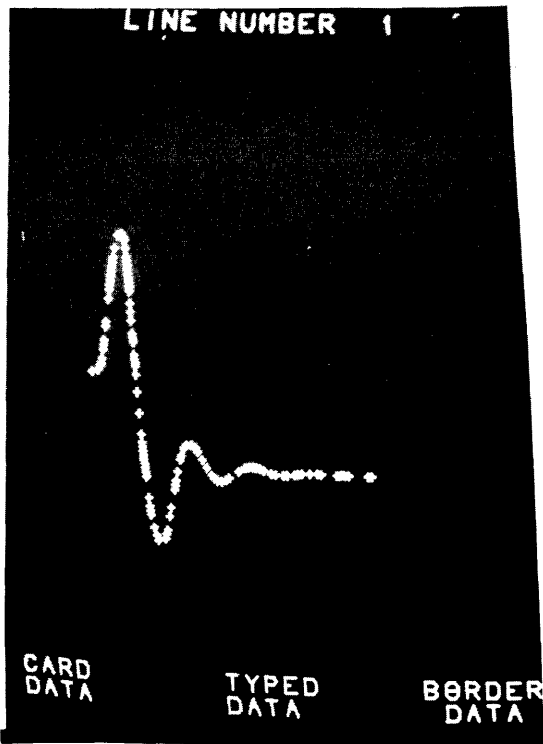


Figure 14. Display of Sampled Line.

feature, the user may select the BORDER DATA option, in which case the coordinates of all digitized points are related to the coordinates of the corners of a 20x20 inch border. The coordinate values of the corners of the border are assumed to have been preset by the CHANGE PARAMETERS function.

After all alphanumeric information has been entered, a summarizing message is displayed on the graphic console screen. An example is shown in Figure 15. The console operator is requested to pass judgment on the quality of the results before they are stored on disk.

## RESULTS

The actual performance of the equipment and system can best be measured by the accuracy and detection capability of the scanning operations. For these reasons, a series of experiments was conducted, aimed at determining the accuracy and detection capability of the device under various operating conditions. Figure 16 is a plot of the distribution of errors in scanning a vertical straight line located at the center of the image field.

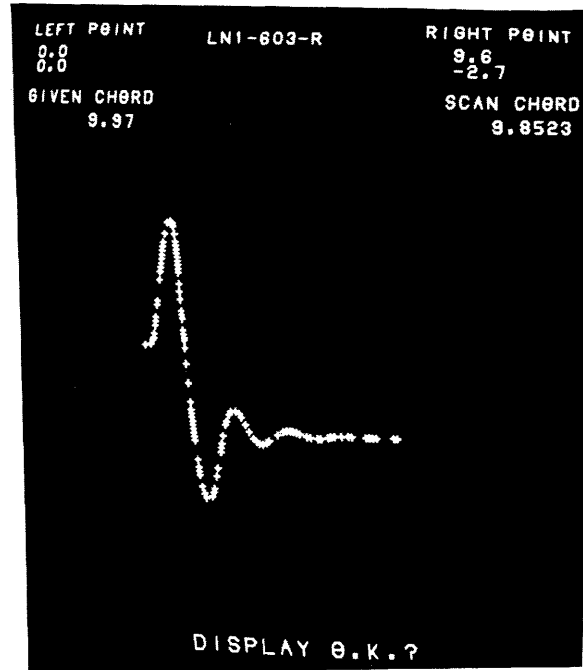


Figure 15. Summary Display.

It should be noted that this figure was generated via the CRT recorder which is a feature of the image processor. Also note that the unit squares along the chordal length of the line are measured in units of 5 inches, while the deviations are measured in units of .05 inches. As can be noted, the maximum deviation of the data from a straight line is  $\pm .02$  inches and there is a high frequency noise level of  $\pm .005$  inches. The original system objective was  $\pm .01$  inches average deviation on a 20x20 inch document. If this test is repeated with no utilization of calibration correction, the results are as shown in Figure 17.

Notice that while the high frequency noise level does not change, the maximum deviation or distortion is now .04 inches.

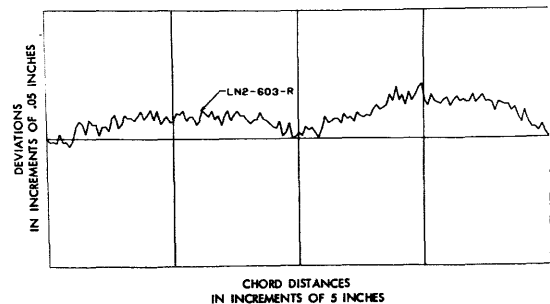


Figure 16. Error Distribution With Calibration.

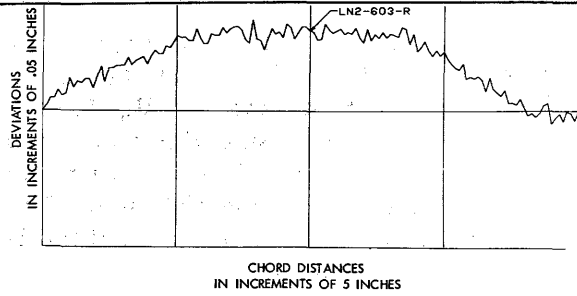


Figure 17. Error Distribution Without Calibration.

The second measure of system performance is the detection capability of the scanner. The range of threshold over which a line can be located is a measure of detection capability. Below a certain threshold level, the scanner can not detect any changes in the percentage of light transmission between background and lines. Above a certain threshold, the scanner will detect light over the entire image. This type of data may be plotted for a single film document for lines of various thickness. Figure 18 is a typical plot of minimum and maximum threshold level versus line width.

The maximum threshold level is the noise level, or the point at which the photomultiplier will always see light. This is a fairly constant value over the entire film image. The minimum threshold level is, of course, a function of line width, until such time as the line widths become appreciably greater than the effective CRT spot diameter. Various qualities of documents and various film exposures will move the wedge left or right and widen or close the wedge. For each particular digitizing application, wedge samples may be taken to determine the limits of detection. Optimum operating

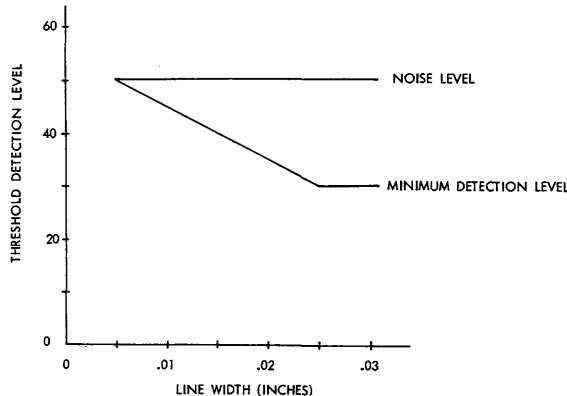


Figure 18. Detection Wedge.

performance may be obtained from the combination of parameters which provide the widest wedge located furthest to the left with respect to line width. Extensive tests are now being conducted on a range of documents to determine these limits of detection. These test results will serve as a guide line for judging the quality of all documents.

CONCLUSIONS

This system can best be described as a line digitizer which is being used as an experiment in processing graphical data. It combines the speed and accuracy of automatic line tracking with the decision capability of a human operator. The system as conceived and implemented has proven the feasibility of close man-machine interaction. While it is not practical to operate the equipment and system with no training, users have become proficient in its use after only one or two hours of instruction. Through the use of graphic displays, it has been possible to program the computer to communicate with a console operator in a medium which is easily understandable. Thus, the utilization of a human operator as the key system component has been very successful. This has been particularly true when automatic line tracking is impractical and the user has had to intercede to assist the scanning.

ACKNOWLEDGEMENT

The line scanning system described in this paper was made possible through the efforts of many members of the Computer Technology Department programming staff. In particular, the authors would like to acknowledge the contributions of Gerald J. Devere and Dennis M. Walker.

REFERENCES

1. GRIMSDALE, R. L., "A System for the Automatic Recognition of Patterns," The Institution of Electrical Engineers, Dec. 1958.
2. GREANIAS, E. C., "The Recognition of Handwritten Numerals by Contour Analysis," IBM Journal, Jan. 1963.
3. FULTON, R. L., "Visual Input to Computers," Datamation, Aug. 1963.

4. MCCORMICK, B. H., "The Illinois Pattern Recognition Computer-Illiacc III," IEEE Transactions on Electronic Computers, Dec. 1963.
5. JACKS, E. L., "A Laboratory for the Study of Graphical Man-Machine Communication," Proceedings of the Fall Joint Computer Conference, San Francisco, California, Oct. 1964, (this volume).
6. COLE, M. P., DORN, P. H., LEWIS, C. R., "Operational Software in a Disc Oriented System," Proceedings of the Fall Joint Computer Conference, San Francisco, California, Oct. 1964, (this volume).
7. HARGREAVES, B., JOYCE, J. D., COLE, G. L., FOSS, E. D., GRAY, R. G., SHARP, E. M., SIPPEL, R. J., SPELLMAN, T. M., THORPE, R. A., "Image Processing Hardware for a Man-Machine Graphic Communication System," Proceedings of the Fall Joint Computer Conference, San Francisco, California, Oct. 1964, (this volume).
8. ALLEN, T. R., FOOTE, J. A., "Input/Output Software Capability for a Man-Machine Communication and Image Processing System," Proceedings of the Fall Joint Computer Conference, San Francisco, California, Oct. 1964, (this volume).

# A GENERAL PURPOSE PROGRAMMING SYSTEM FOR RANDOM ACCESS MEMORIES

*C. W. Bachman*  
*General Electric Company*  
*Phoenix, Arizona*  
*and*  
*S. B. Williams*  
*General Electric Company*  
*New York, New York*

## I. INTRODUCTION

During the past ten years, information processing technology has made significant advances in many directions. Faster, less expensive, more flexible *hardware* has been continually announced by the various computer manufacturers. In the *software* area, the FORTRAN, ALGOL, and COBOL languages have been developed and improved and more efficient compilers are now available. *Applications* now include the complete spectrum ranging from free-standing analytical programs to large complex information processing systems.

Computers have been applied to business information processing problems with varying degrees of success. Many accounting operations and facets of historical record-keeping have been mechanized with proven time, cost, and accuracy benefits. Those types of business operations dealing with planning and control (or command and control if you are part of a military establishment) are receiving considerable attention from the mechanization standpoint. While many mechanization attempts have been made in this area, the proven successes are few. To some extent this can be attributed to the greater complexity of these classes of problems and the fact that information must be stored,

retrieved, communicated, and processed concurrent with the flow of orders and materials.

The information processing field seems to be moving exponentially in the direction of "real time" and total or highly integrated information systems. This movement has been accelerated by the introduction of larger, faster, and more economical mass random access memory devices coupled with faster computers and better communication equipment. These new facilities offer the information system designer a new opportunity 1) to organize his information files with minimum duplication and redundancy, 2) to provide a better man-machine interface by giving people quick access to information, 3) to store, retrieve, and process information when the need arises rather than when the computer schedules dictate, 4) to provide a single data base for many applications as opposed to the arbitrary sequencing of single files for each particular application.

Any attempt to exploit the opportunities presented by the new mass memory devices places a high burden on the information system designers and programmers. This is true because it is difficult to structure and organize complex information relationships within the parameters of the mass memory devices. It is also

very difficult to write computer programs to store, maintain, retrieve, and process the complex data. To date there has been little if any software available to facilitate these problems.

The General Electric Company through its Corporate Services has been conducting a continuing research program on the manufacturing control problem since 1956. The decision table and TABSOL techniques resulting from this research work were described to the information processing world in 1960.<sup>1</sup> During the past four years a considerable effort has gone into studying the information requirements for manufacturing control and how the information might be organized and processed more effectively by using mass memory devices. As a result of this work, a new approach has been developed—the Integrated Data Store.

## II. THE INTEGRATED DATA STORE— A NEW APPROACH<sup>1</sup>

The purpose of this paper is to introduce the Integrated Data Store, a general purpose programming system for mass random access storage devices. The particular implementation that will be described is now being installed at several General Electric sites using a GE-215 or GE-225 computer. The Integrated Data Store language and functions will be available early in 1965 as extensions to the COBOL compilers for the new GE-400 and 600 series computers. The principles involved, however, are completely general purpose and could be readily adapted to any general purpose computer to which a mass memory device can be attached.

The Integrated Data Store has been designed from a user's point of view by users. Furthermore, it is a product that draws upon the interest and ideas of many General Electric people with vast and diverse experience as users of computers in business. Particular credit is due Homer Carney of the New York Information Processing Center (GE Computer Department) who long served as the senior programmer on the project and Irv Burch and Bill Helgeson of the Internal Automation Operation whose ideas heavily influenced the current organization of the system. Jerry Aman, Ed Dodge, Phil Farmer, John Gallagher, Jane Gilbane, George Hess, Dave Johnson, Dave Lattemore, Ron Pulfer, and Tom Waldron are others who have

had a significant impact upon the specification or programming of the system. Many others have been helpful since the beginning of the Integrated Data Store work in 1961.

## III. INTEGRATED DATA STORE— ADVANTAGES

The original Integrated Data Store software package was used in mid 1963 to make comparisons against conventional random access programming techniques in systems design effort, programming effort, file utilization, and computer running time. The IDS compared very favorable on all counts. Since that time, further refinements have been made to the software package.

Experience to date using IDS has demonstrated the following advantages:

1. Greater insight and understanding of information relationships.
2. Reduced time and cost to design, program, and test comparable applications.
3. More efficient computer processing.
4. Better data storage unit utilization through redundancy elimination.

## IV. INTEGRATED DATA STORE— ORGANIZATION

The IDS can be described best if it is divided into three areas of discussion:

- a. Data Organization—Technique for Mass Memory
- b. Data and Procedural Language
- c. Input/Output Controller

Data Organization refers to the establishment of inter record relationships within the IDS. This association is achieved through the use of chains which provide cross reference linkages between records. These chains provide the integrated force which is implied in the name, "Integrated Data Store."

Data and Procedural Language refers to the definition of records and their chain associations, and the procedural verbs by which these records are stored and retrieved.

The Input/Output Controller refers to the physical manipulation of the mass random access device and the buffering and housekeeping



associated with temporarily storing blocks of data in core memory.

**A. Data Organization.** The record is the major unit of data organization in the Integrated Data Store.

This is a record in the GECOM (General Compiler) and COBOL sense. It contains a set of data fields which collectively describe the event, thing, status, or plan that the record represents. The Integrated Data Store augments these records with additional fields called chain fields which contain the address of other Integrated Data Store records. The chain fields point from one record to the next creating a serial association of records.

This association is constructed according to the data definitions and the executed procedural commands. The chain is the record organization technique used by the IDS for meaningful associations of records.

**B. Data and Procedural Language.** The Integrated Data Store provides its user with the ability and requirement to predefine his records, their data fields, and their chain fields. Once these records and fields have been defined, the user is free to operate upon the records without concern for the physical aspects of input or output, the linking of records into chains, or the protection of the data from erroneous access.

**A. RECORD CONSISTS OF  
DATA FIELDS  
CHAIN FIELDS  
DESCRIBING AN EVENT  
A THING  
PLAN  
STATUS**

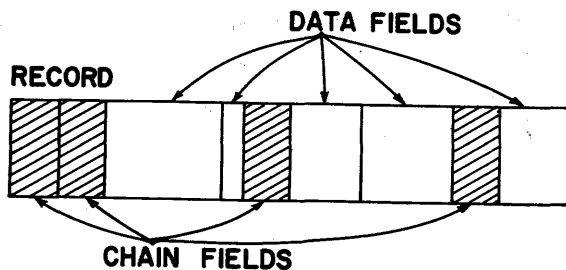


Figure 1. Record Definition.

**A CHAIN CONSISTS OF  
A SERIAL ASSOCIATION  
OF RECORDS**

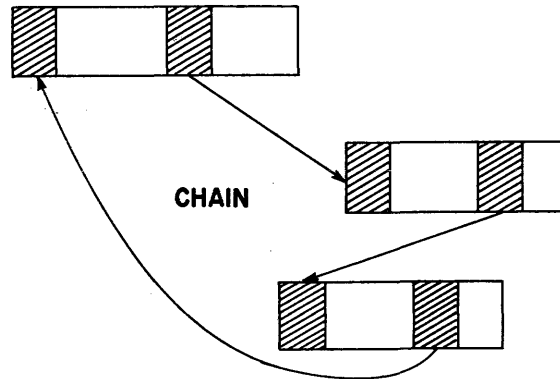


Figure 2. Chain Definition.

The user has four new commands or procedural verbs at his disposal. These verbs provide for the execution of the four basic record processing functions and are complementary to existing COBOL and FORTRAN procedural verbs. These are; "PUT" to store a new record into the file and link it into chains as specified in the data description, "GET" to retrieve a record already in the system, "MODIFY" to change the content of one or several data fields with automatic relinking of chains, if necessary, and "DELETE" to delink a record from its chains and remove it from the file.

**C. Input/Output Controller.** The Input/Output Controller of the IDS controls the data storage device.

It transfers data blocks in and out of core in response to commands to retrieve a specific record, to store a specific record, or to expand or contract a specific record. In order to minimize the data storage device seek and transfer time, an inventory of data blocks is maintained in core memory. These blocks are stored in numerous buffers in core. The number of buffers depends on the amount of space available after the IDS subroutines and the problem solving routine have been loaded. The larger the number of data blocks stored in core, the greater

## Input/Output Controller

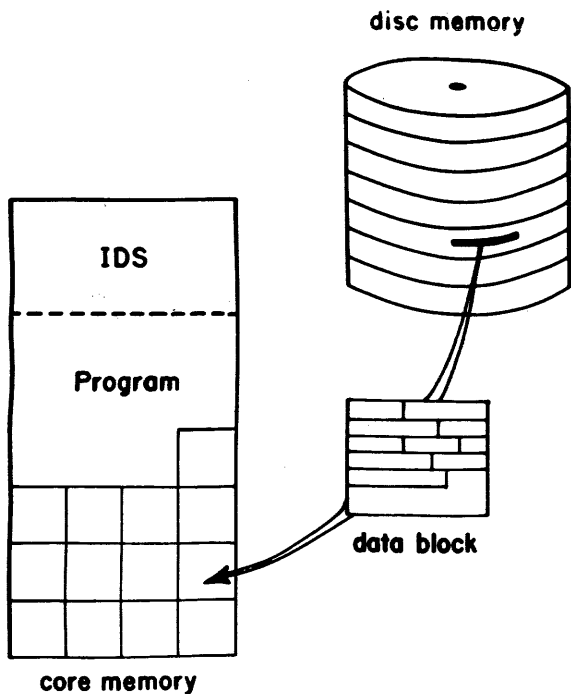


Figure 3. Input/Output Controller.

the possibility that the one needed next will already be in core. To improve the probability of finding the block desired in core, the I/O Controller keeps track of the sequence of block retrieval and utilization and holds the most recently active data blocks in the buffers. Blocks which are not frequently accessed are retired from core to make room as others are called in. The I/O Controller notes which blocks have been modified and writes only the modified blocks back to the data storage unit. The IDS data block manipulation is analogous to the program block "page turning" of the Ferranti Atlas computer.

### V. DATA BLOCKS

Looking closer at IDS data blocks, the following characteristics should be observed.

They have a fixed maximum size which is an environmental constant. They consist of one or more data records which collectively represent the actual size of the block. The maximum number of data records is controlled by the size

of the maximum block. Every block begins with a block header record. The block header record contains several data fields used by the system. One indicates the space available in the block for additional records, or record expansion within the block. Another indicates whether the block has been altered since retrieval. Still another is a chain field which indicates the address of the first record of a chain of records, all of which randomized to that block.

### VI. DATA RECORDS AND FIELDS

The records of the IDS are fixed format, fixed length records in the GECOM, COBOL tradition, i.e. a specific type of record such as a payroll or inventory record has a fixed length and format. Variability in the conventional sense of record length is automatically achieved through the IDS techniques of data structuring. A master record is used with a variable number of detail records.

Records of many different types, each with differing length and format may be used in the system and may be stored within the same

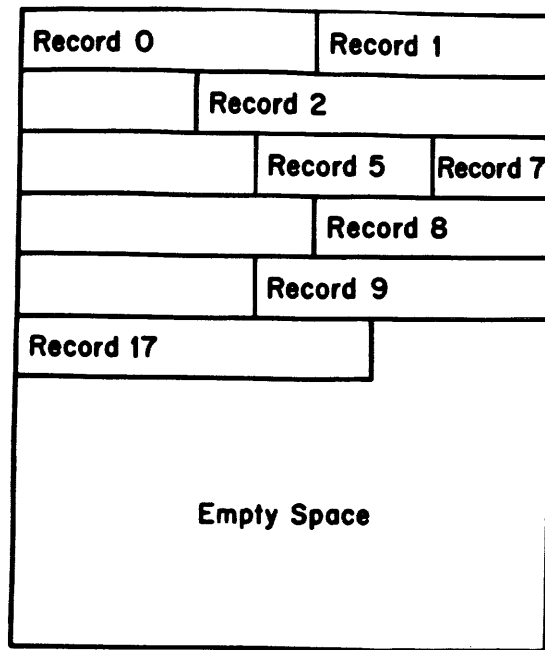


Figure 4. IDS Data Block.

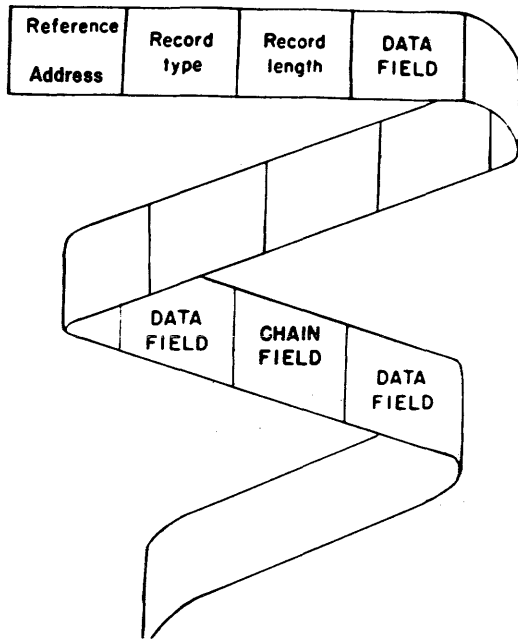


Figure 5. IDS Record Structure.

block. In order that control may be maintained, each record has the same three fields at the very beginning. These fields are the reference code (block number and intra-block record number), the record type, and the record length. The balance of the record consists of data and chain fields in the number and variety to suit the application requirements. Data fields may be defined as being in a logical mode (bits), signed binary numeric mode (one or two words) or an alphanumeric mode (characters). Fields may vary in size from a one bit switch up to many characters for a drawing and part number or a man's name. These fields will be specified by the systems designer.

Chain fields are defined for each chain in which a record participates. Experience in IDS systems indicates that the average record is in only two chains, and an occasional pivotal record in the information integration may be in six or eight chains. There is no upper limit on data or chain fields except that which is provided by the maximum block size. The average record in installations today has been eight to twelve words in total length, with an occasional

record type in the forty to sixty word size. These are twenty bit, three character words.

The reference codes in the IDS chain fields are not physical addresses which specify particular discs, tracks and heads. They are more properly described as relative addresses which indicate a relative position in the total environment of mass storage. Therefore, an expansion or contraction of the number or size of the data storage units does not destroy the existing reference codes. It merely changes the mapping function which translates a particular reference code into its disc, track, and head number.

The IDS Records are stored only once in the IDS.

This has three important advantages. First, the additional space required for duplicate records is eliminated, resulting in a reduction in the total storage capacity required. Second, the work of data maintenance is greatly reduced as there is only one record to retrieve and modify. This eliminates the possibility that one of the copies of a record will not be properly modified. As there is only one copy of a record, all users

- Have any number of data fields.
- May be linked into any number of chains.
- Are stored only once in the IDS

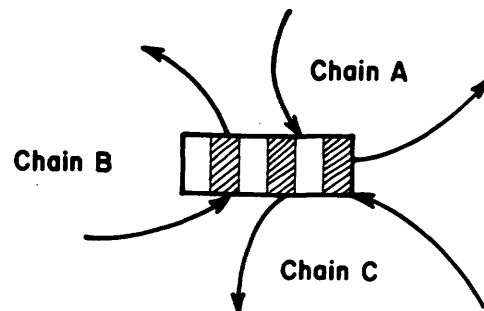


Figure 6. IDS Record/Chain Structure.

have their eyes on it and incorrect information will be quickly spotted and corrected. Finally, all reports, drawn from the file, will be consistent since there is only one set of facts (records).

## VII. CHAINS

The IDS chains have several structural aspects which should be emphasized.

Each chain has only one master record. The record type of the master is specified when the chain is defined in the data definitions. Whenever a master record is "PUT" into the IDS, a chain is created which has no details in it. The chain field in the master record stores the reference code of the next record in the chain, which initially is the reference code of the master record itself. As additional records, that are specified as details, are "PUT" into the file, they are linked into the chain. However, the chain always closes back on its master. The position in the chain of a new detail depends on the chain specifications.

- **Have one master record and any number of details.**
- **Link records together in an endless loop.**
- **Associate related records in meaningful sequences.**

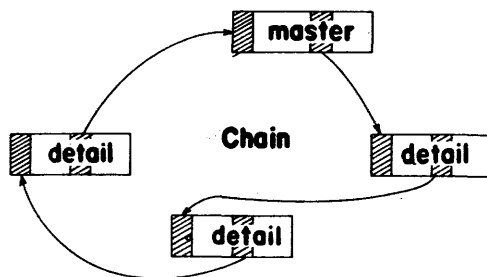


Figure 7. IDS Chain Structure.

It was previously stated that a record may be in any number of chains. Now it is worth expanding this statement to read that a record may serve as a master or detail in any number of chains. The only restraint is that no record may be a detail of itself directly, or through the interaction of several chains.

## VIII. DATA STRUCTURE SHORTHAND

It is frequently desirable to display pictorially the relationship between records. This is particularly important in developing an overall view when planning an information system. A special graphic technique has been developed to display records and their master-detail (chain) relationships.

This technique uses a block shape to designate a record type and an arrow connecting two blocks to designate a chain. The arrow points from the master to the detail. This picture of block, arrow, block carries the following message: 1) there are some number of records in the system of the master type; 2) each of these records is the master of a chain of the specified type; 3) there are some number of records of the detail type (0, 1, 2, 3, ..., n) in

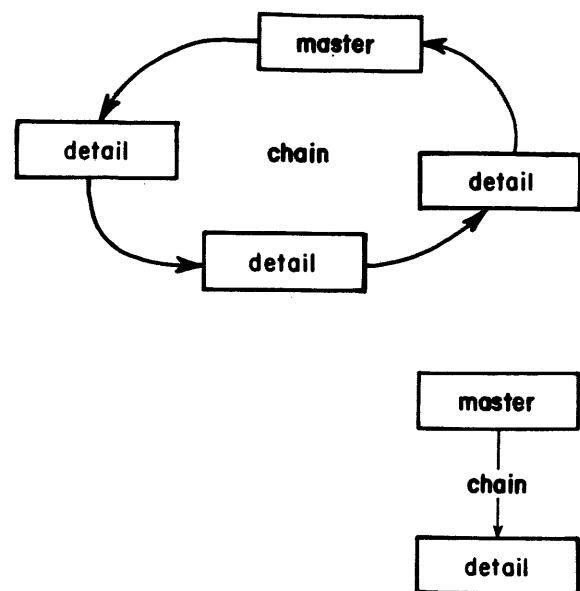


Figure 8. IDS Data Structure Shorthand.

each such chain. Using this graphic technique, very complex data structures may be presented in a condensed and understandable form.

It is believed that the long sought information algebra may be developed around this notation. A new set theory is needed in which the master record is represented by the empty set, and details represent the ordered members of the set.

X. SAMPLE DATA STRUCTURE (PURCHASE ORDER)

The information contained on a purchase order can furnish an example of how information might be structured.

Looking at a purchase order form, three groups of information may be seen. One group is concerned with information about the vendor, i.e. his name, address, and vendor code. Another group is concerned with information about the order, i.e. the order number, due date, mode of transportation, and dollar value. The third group is concerned with the information about a particular item to be purchased, i.e. its identification, description, quantity, unit price, and

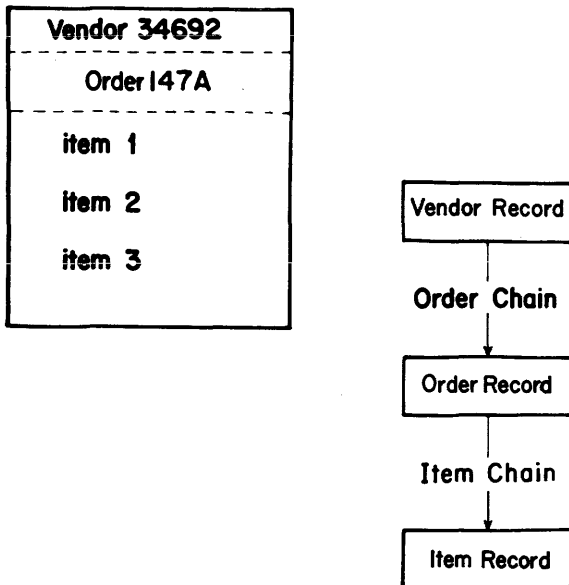


Figure 9. Purchase Order Data Structure.

extended dollar value. Three different records might be designed in order to carry the information contained in these three groups. These three records would be a vendor record, order record, and item record. If a purchasing information system were established along these lines, there would be a vendor record for every vendor with whom the business is concerned. The vendor record would be the master record of an order chain. There would be an order record for each order currently stored in the system. It would be a detail in an order chain. Each order record would, in turn, be the master of an item chain. This item chain would contain one or more item records depending on the number of items on the purchase order. This example contains three records and two chains. The vendor record is only a master. The order record is both a master and a detail. Finally, the item record is only a detail. The IDS Data Structure Shorthand shows all this with only three blocks and two arrows. Very complex systems with thirty or more record types have been clearly described using the IDS shorthand.

A data description for the sample problem is shown in figure 10. Each record must be clearly defined as to the data fields which it contains as well as the chains in which it participates. The appropriate IDS controls for storage and chaining must also be described.

A look at part of a network created by the vendor, order, and item records illustrates both the need for the data structure shorthand and

Data Type	Record Name	Chain Name	Field Name	IDS Controls	Field Image
Record	VENDOR			Calculated	
Field	VENDOR		VENDORNO	Unique	X(6)
Field	VENDOR		VENDORNAME		X(15)
Field	VENDOR		ADDRESS		X(24)
Field	VENDOR		CITY STATE		X(24)
etc. for all fields in VENDOR record					
Chain Master	VENDOR	ORDERCHAIN		Sequenced	
Record	ORDER			Calculated	
Field	ORDER		VENDORNO	Redundant	
Field	ORDER		ORDERNO	Unique	X(12)
Field	ORDER		ORDERDATE		999
etc. for all fields in ORDER record					
Chain Detail	ORDER	ORDERCHAIN			
Chain Control	ORDER	ORDERCHAIN	VENDORNO	Match	
Chain Control	ORDER	ORDERCHAIN	ORDERNO	Ascending	
Chain Master	ORDER	ITEMCHAIN		Sequenced	
Record	ITEM			Secondary	
Field	ITEM		ORDERNO	Unique, Redundant	
Field	ITEM		ITEMNO	Unique	X(3)
Field	ITEM		MATLIDENT		X(18)
Field	ITEM		ORDERQTY		9999999
etc. for all fields in ITEM record					
Chain Detail	ITEM	ITEMCHAIN		Prime	
Chain Control	ITEM	ITEMCHAIN	ORDERNO	Match	
Chain Control	ITEM	ITEMCHAIN	ITEMNO	Ascending	

Figure 10. Purchase Order Data Description.

its power. The arrows indicate that one record "points" to the next record in a chain and that each chain "closes" onto the master record of the chain. Regardless of how many chains a record is in, the record exists in the file only once. It may be "pointed at" by many other records.

Still another way of illustrating the data contained in the sample problem is shown in figure 12. Here the chaining is represented by the appropriate reference addresses.

X. Procedural Commands

The functional verbs PUT, GET, MODIFY, and DELETE, previously introduced, require further explanation for better understanding. These verbs may be used in a GECOM, COBOL, or FORTRAN sense. In fact, there is reason to wonder when they will move out of the developmental world and become part of the industry standard languages. Perhaps, a better phraseology would be to ask whether the industry languages committees will take advantage of IDS to catch up with data storage units, real time, and command and control systems pro-

**VENDOR RECORD**

REF ADDRESS	VENDOR NO	VENDOR NAME		ORDER CHAIN NEXT	
100	34692	ABC CO.	ETC	322	ETC

**ORDER RECORD**

REF ADDRESS	ORDER NO		ORDER CHAIN NEXT	ITEM CHAIN NEXT
285	207A	ETC	100	287

**ORDER RECORD**

REF ADDRESS	ORDER NO		ORDER CHAIN NEXT	ITEM CHAIN NEXT
322	147A	ETC	285	335

**ITEM RECORD**

REF ADDRESS	ITEM NO	MATERIAL IDENT	QTY	ITEM CHAIN NEXT
335	1	75L38	10	337

**ITEM RECORD**

REF ADDRESS	ITEM NO	MATERIAL IDENT	QTY	ITEM CHAIN NEXT
337	2	122A93	310	342

**ITEM RECORD**

REF ADDRESS	ITEM NO	MATERIAL IDENT	QTY	ITEM CHAIN NEXT
342	3	46A95PI	2	322

Figure 12. Purchase Order Data Structure.

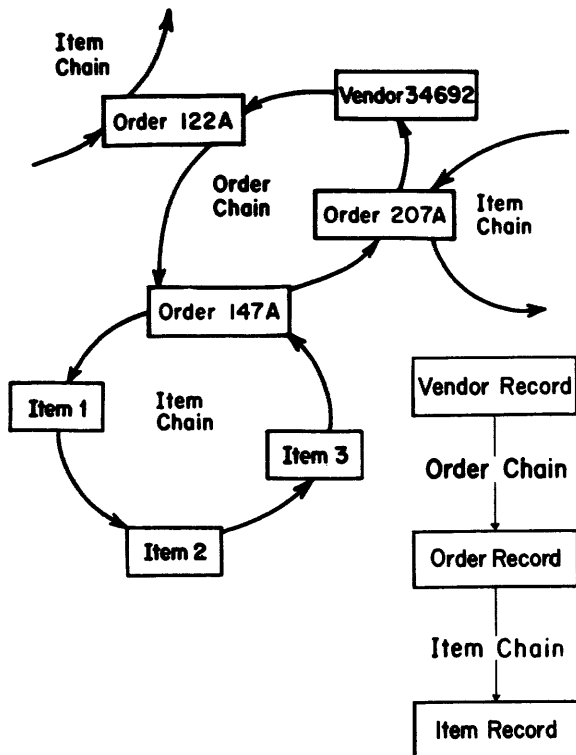


Figure 11. Purchase Order Data Structure.

gramming. They are part of the COBOL compiler language for the GE-400 and 600 series computers now.

The following are examples of procedural statements which would cause the IDS to execute certain actions. The letters in capitals are the required words of the IDS language. The lower case letters are data and procedural variables, i.e. record names, chain names, field names, and sentence names.

A. *Example 1—PUT.* "PUT vendor RECORD." This command stores a new vendor record in accordance with its data description. Its fields' values would be picked up from working storage and packed into the new record skeleton in a data block. The order chain field in the new record would be packed with the reference code of the new vendor record itself because there are no details at this moment and the next record in the order chain is the vendor record.

B. *Example 2—GET.* "GET NEXT order RECORD OF order chain, OR IF vendor RECORD GO TO location~a." This command would retrieve the next record of the order chain and unpack its fields into working stor-

age. If the next record is an order record, the control would be transferred to the succeeding command. If the next record is a vendor record, control would be transferred to the command identified by the sentence name "location~a." The actual record retrieved is not accessible to the programmer—however, the contents of its data fields are unpacked and made available in working storage. This serves two purposes. First, protection is given to the data in the file in a father-son type sense. Second, it means that the data from a record will remain in working storage until another record of the same type is retrieved and unpacked into the same working storage fields. For example, if an item record were first retrieved, followed by its order record, then the order record's vendor record, the fields from all three records would be simultaneously available in working storage for processing.

C. *Example 3—MODIFY.* "MODIFY CURRENT item RECORD, REPLACE quantity FIELD." This command will modify the current item record, i.e. the last item record accessed, regardless of what has transpired since it was processed. It will pack the content of the working storage field "quantity" into the corresponding data field of the record replacing the existing value. A modify command will modify one or several fields in accordance with those specified in the command. Fields may also be modified by adding or subtracting the contents of working storage to that of a record. The appropriate commands would be MODIFY recordname RECORD, ADD fieldname FIELD, or SUBTRACT fieldname FIELD.

It was mentioned earlier that fields were frequently used to sequence the detail records in a chain. These fields are called sequence control fields. If a sequence control field is modified, the detail will be automatically delinked from its master and relinked to it again in accordance with the new value of its sequence control field.

Fields are also used to control the selection of the master records and chains in which to insert detail records. These fields are called match control fields. If a match control field of a detail record is modified, the record will automatically be delinked from its old master.

Its new master will be retrieved, and the record linked to its new master according to the ordering rule specified for the chain.

D. *Example 4—DELETE.* "DELETE vendor RECORD, IF ERROR GO TO error~a." This command will retrieve the vendor record specified by the code stored in the "vendor code" field in working storage. If there is no vendor record with that specific vendor code, the command will respond by setting up an error code to specify the nature of the "fault" and transfer control to the command identified by sentence named, "error~a." If the vendor record is successfully retrieved, its deletion process will begin. If a vendor record is to be deleted, its order chain must be deleted too. Consequently, if a vendor record is to be deleted, its order chain must be searched to ascertain that there are no order records in it. If there are order records, they must be deleted before the vendor record is deleted. In the same manner, an order record may not be deleted if there are any item records in its item chain. Therefore, all item records in an item chain must be deleted before the order record is deleted. This makes the deletion command a very powerful command and one to be used with due respect.

Two optional features have been provided which aid the programmer in using the delete command. If the programmer anticipates that order records may be linked to the vendor record (that a detail may be linked to its master), he may wish to print a control report of the orders deleted by using the phrase "AND IF order RECORD PERFORM reportline~1" with his delete command. This will cause the deletion process to be interrupted everytime an order record has been deleted. The subroutine identified by the sentence name, "reportline~1" will be executed. Because of the frequent desire to produce some form of a control report on deletions, the delete command actually retrieves and unpacks, into working storage, the data fields of a record prior to deleting it.

The second optional feature of the DELETE command permits the programmer to attach an escape phrase. For example, the programmer might attach the phrase, "BUT IF order

RECORD GO TO statement~a.” In this case, the detection of an order record as a detail to the vendor record would immediately terminate the deletion command without the order record, its item records, or the vendor record being deleted. Using the DELETE command in this manner, the programmer need not test to determine the presence of an order record prior to initiating the DELETE command. He may boldly set out to delete a master record, and still escape from the deletion if there is a detail that he wants to protect.

All of the commands, GET, PUT, MODIFY, and DELETE, permit the addition of error test and branch. The user is urged to use them so that he is immediately aware of any fault that occurs and the nature of that fault. A typical fault that could occur during a “PUT” is attempting to put a duplicate record which is prohibited according to the data definitions.

#### XI. RECORD RETRIEVAL SPECIFIERS AND RULES

Three of the IDS macro instructions require that a record be retrieved so that it may be operated on. GET means retrieve a record and unpack its data fields into working storage. MODIFY means retrieve a record and modify specified fields in the record according to the command the the contents of working storage. DELETE means retrieve a record, unpack its data fields into working storage, delete any detail records, and finally, delete the specified record. Only the PUT command lacks the retrieval aspect. It means find space for a new record, link it into its chains, and pack its data fields from working storage.

There are six different retrieval rules from which the programmer may choose. These rules may be used in conjunction with the functional processes; GET, MODIFY, and DELETE. Examples 2, 3, and 4 in Section X used three of these rules, respectively, “NEXT OF CHAIN,” “CURRENT” and the associative retrieval rule which is specified by the absence of a record specifier adjective. These rules may be sub-divided into two classes. The two rules which are absolute in their nature, i.e. there is only one record that satisfies their specifications regardless of when they are executed. They will be discussed first. The other four

rules are relevant to what has transpired previous to their execution.

A. “\_\_\_\_\_” *specifier*. The absence of a specifier indicates that the record to be processed is identified by the data values stored in the fields of working storage. The particular fields concerned are those fields which have been described in the data description of the specified record as the unique fields for that record.

B. “DIRECT” *specifier*. This specifies that the record is to be retrieved based on the reference code (address) stored in the communications field named, “QDIRF” (*DIRECT REFERENCE*). The programmer may store any reference code there and then retrieve the record associated with that reference code.

The IDS system is so designed that once the reference code is assigned to a record, it is permanent. The addition or subtraction of data storage units will not affect it. The modification of the record to add or delete either data or chain fields or modify their content will not affect it. In fact, the uniqueness and permanence of the reference codes make them ideal candidates for dual use as reference code and invoice number, order number, pay number, vendor code, customer code, drawing number, or stock number.

C. “CURRENT” *specifier*. The “CURRENT” record specifier instructs the system to reretrieve the last record of that type processed by a GET, PUT, or MODIFY command. If the last command executed for a given record type were a DELETE command, the last record would have been deleted and it would be impossible to retrieve the current record of that type because there is none. This would create a “fault” and an error would be signalled.

D. “NEXT” *specifier*. The “NEXT” specifier is one of a set of three chain processing specifiers. These specifiers require that a chain name be appended so that the specification would be complete. As an example, the command below specifies that the programmer wishes the program to access the next item record in the item chain:

“GET NEXT item RECORD OF item CHAIN.” The particular record accessed



clearly depends upon which record is the current record in the item chain when the command is executed. An IDS command must have been executed prior to executing any of the chain processing commands. This prior command must have accessed a record in the desired chain and therefore established the current record in the chain. Only then does the phrase "NEXT RECORD OF CHAIN" have any meaning.

E. "*PRIOR*" specifier. The "PRIOR" specifier is used to specify that the chain is to be processed in a backward direction. This command contains the same restraint as the "NEXT" specifier, that the current record of the chain must have been established. The ability to process a chain backwards is optional and dependent upon the chain having been specified in the data description, as a "PRIOR" chain.

F. "*MASTER*" specifier. The "MASTER" specifier directs the chain processing to proceed directly to the master record of the specified chains, accessing but ignoring all intermediate detail records. The optional specification of the chain as a "HEADED" chain provides an additional pointer field in each detail record containing the reference code of the master record. In the presence of this option, the "MASTER" specifier will proceed directly to the master record without accessing the intermediate detail records. As with the other chain processing specifiers, the chain must have been accessed and a current record established, prior to executing a "MASTER" command.

*Alternate Retrieval.* The retrieval rules using the record specifiers, NEXT, PRIOR, and DIRECT exist under conditions where the type of record to be retrieved cannot always be predicted. These commands, therefore, permit the insertion of one or more "OR IF recordname RECORD GO TO sentence name" phrases. The program logic is then able to branch to the specified sentence following the execution of the functional portion of the command in accordance with the record retrieved. The use of an "IF recordname RECORD GO TO sentence name" phrase (note the "OR" is missing) will cause the program logic to branch after retrieval, but *before* the execution of the func-

tional portion of the command. This permits the execution of the functional portion of the command (GET, MODIFY, or DELETE) when using the record specifiers DIRECT, NEXT or PRIOR on selected record types and the by-passing of the function if other types are retrieved. As an example, the following command might be used:

"DELETE NEXT order RECORD OF order CHAIN, IF vendor RECORD GO TO sentence~a."

The repeated use of this command would delete successive order records which are in the order chain. However, when the vendor record is retrieved, it will not be deleted and control will be transferred to sentence~a.

The data structuring abilities of the IDS permit the definition of more than one detail record type in a chain. In the case of the PRIOR and NEXT OF CHAIN retrieval actions, unspecified record types in the chain will be accessed and skipped over until a record of a specified type is retrieved. If the chain is completely traversed without the retrieval of a specified record type, an error is signalled. The retrieval of an unspecified record type by the DIRECT specifier will cause an error to be signalled.

*Error Conditions.* All of the commands of the IDS are structured with the provision for an error statement. As an example:

"GET item RECORD, IF ERROR GO TO sentence~b."

If this command were attempted and had failed because no item record could be retrieved with an order number and line number, matching those in working storage, then the program control would be transferred to sentence~b. This permits the program to test whether the function has been carried out successfully. If the command has not been successful, the error condition may be tested to determine the nature of the fault and the appropriate action initiated.

## XII. HISTORY OF DEVELOPMENT

Historically, the IDS's foundation in well disciplined data structure goes back to the file

structures developed by General Electric at Hanford for their 702 Report Generator and File Maintenance System.<sup>2</sup> These structures reached greater generality and power in the SHARE 9PAC system which was largely guided and programmed by GE Hanford and supported by The Dow Chemical Company, Union Carbide Company, GE Heavy Military Electronics Department and others. The deliberations of the SHARE committee on The Theory of Information Handling<sup>3</sup> also contributed to the early thinking on the Integrated Data Store.

The current implementation of the Integrated Data Store is based on a set of free standing subroutines written in the General Assembly Program language for the GE-200 series computers. The original version was prepared as an adjunct to GECOM. It had a compiler generator which processed data definition cards and IDS macro instructions and produced mixed GECOM and General Assembly Program statements which were subsequently compiled by GECOM to produce an object program. The macro instructions were executed as generated in-line coding. IDS was first operated in this form in January, 1963, with hand compiled macro instructions. The application that it was applied to was the IDS compiler-generator itself which was used to generate IDS coding for subsequent application programs. The first completely generated program was a product materials file maintenance and explosion routine. This routine was used during the summer of 1963 to run comparative speed tests with another routine performing the identical tasks which had been hand coded employing conventional disc programming techniques. The machine generated IDS program ran twice as fast as the comparison program and used less file space to store the data.

During the Fall of 1963, the current implementation of the IDS was programmed. This version switched from compiled in-line coding to an interpretive subroutine organization with calling sequences. This is another step in the separation of the data structure from procedural logic and parallels the dictionary used by 9PAC which was brought together with the procedure at load time. The parametric ver-

sion appears to operate at about the same speed as the in-line coding version.

### XIII. SUMMARY

The Integrated Data Store is an operational tool for programming the GE-225 with Disc Storage Unit. It automatically processes the complex file maintenance and retrieval problems presented by a data storage unit. It gives a high degree of file protection and through data structuring and redundancy elimination, it accomplishes considerable file compression. The user has the option of many storage and retrieval techniques. It yields efficient programs with buffered operation of the disc file. The requirement to structure the data before programming greatly reduces redesign and debugging problems. IDS provides for the first time an effective method for describing the complex interrelationships of data present in most information systems. It further provides the means for efficiently processing and maintaining these in the environment of a mass memory system. It moves list processing techniques out of current limitations of core memory and thus makes them available for practical data processing.

We challenge the national standards committees for COBOL, FORTRAN, and ALGOL and the designers of the "New Programming Language" to survey their current accomplishments, which are many, and to determine whether the above capabilities offered by IDS should be added to their languages.

### REFERENCES

1. KAVANAGH, T. F., "TABSOL—A Fundamental concept for Systems Oriented Languages," Eastern Joint Computer Conference, December, 1960.
2. MCGEE, W. C., "Generalization—Key to Successful Electronic Data Processing," *Journal ACM*, January, 1959.
3. SHARE Committee on Theory of Information Handling, "TIH #1 Report," 1959.
4. BACHMAN, C. W., "The Integrated Data Store Function Specifications," General Electric Co. internal publication, January, 1962.

# THE IBM HYPERTAPE SYSTEM

*B. E. Cunningham*

*Development Lab, Data Systems Div., IBM Corp., Poughkeepsie, N.Y.*

## GENERAL DESCRIPTION

The IBM Hypertape system was designed as a high-speed I/O device for the IBM 7074-7080-7090-7094 computers.\* It is composed of a two-channel control unit, the IBM 7640 (Figure 1), and IBM 7340 tape drives (Figure 2). Ten 7340s can be attached to each 7640 channel.

The 7640 is attached to a computer through a simplex interface consisting of 33 lines. Four commands can be issued from the computer to the 7640 across this interface—a Read, a Write, a Control, and a Sense. The Control command instructs the 7640 to perform such operations as Space, Backspace, Rewind, etc. The Sense command interrogates the status of the 7640 and 7340. It allows the computer to determine, for example, if the tape drive is loaded or busy, or what type of errors occurred in the control unit on the last operation. The two channels of the 7640 time-share a Read and a Write section. This allows one channel to write while the other channel is reading. Instead of the NRZI method of recording, the system uses IBM's phase-encoding technique,\*\* which records a signal for both a one and a zero. This system is very reliable, since the possibility of either losing a weak signal or picking up noise is greatly reduced.

The tape used is 0.1-mil oxide on a 1-mil polyester base. It is one inch wide, with 1800 feet on a reel. Written across the tape are ten bits; eight are information bits, and two are check bits used for error correction during

reading. These two check bits, in conjunction with a signal-strength monitor, provide detection of all errors and correction of all single- and 33 of 45 possible double-bit errors. In alphanumeric mode, six of the information tracks are utilized per character. In packed-numeric mode, two four-bit characters are written side by side across the tape, as shown in Figure 3. The character density is 1511/inch, and tape speed is 112.5 inches/sec. This results in an alphanumeric data rate of 170,000 characters/sec, or a packed-numeric data rate of 340,000

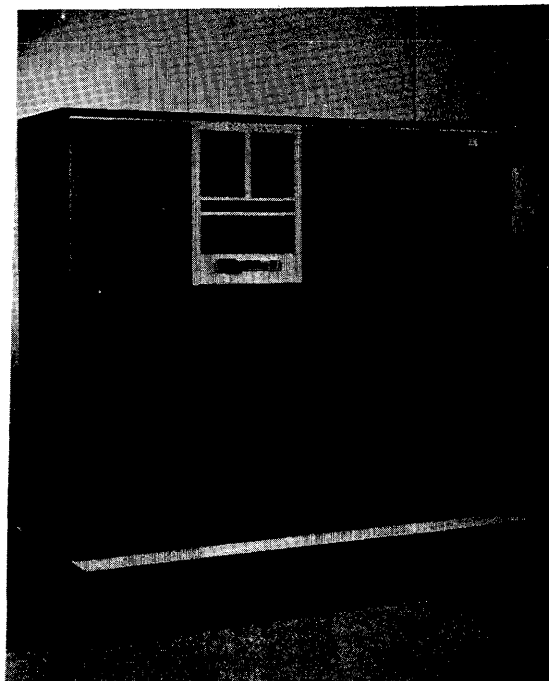


Figure 1. IBM 7640 Hypertape Control Unit.

\* A modified version is also offered for the IBM System/360 (see page ).

\*\* Williams Patent #2734186

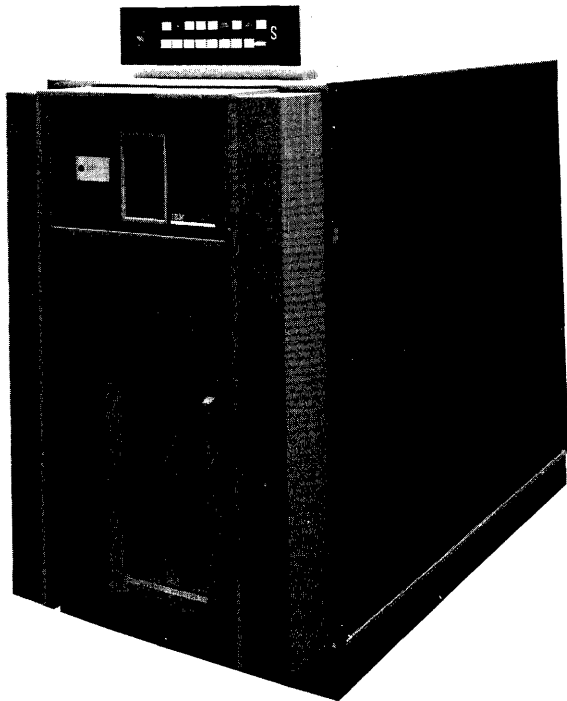


Figure 2. IBM 7340 Hypertape Drive.

characters/sec. Figure 4 shows a comparison of the reel capacity of Hyper and IBM 729 reels.

The tape has been completely enclosed in a dust-resistant cartridge to eliminate as much contamination as possible, and to provide a means of automatic loading and unloading without physically handling the tape. Another advantage of the cartridge is a programmable file-protect device. A tape can be file-protected under program control, but it can be "un-file-protected" only manually.

An Automatic Cartridge Loader (ACL) is available as an optional feature (Figure 5). A

HYPERTAPE BIT TRACKS	BCD	PACKED FORMAT
CO	CO	CO
CI	CI	CI
0	0	8
1	0	4
2	B	2
3	A	1
4	8	8
5	4	4
6	2	2
7	1	1

Figure 3. Hypertape Character Formats.

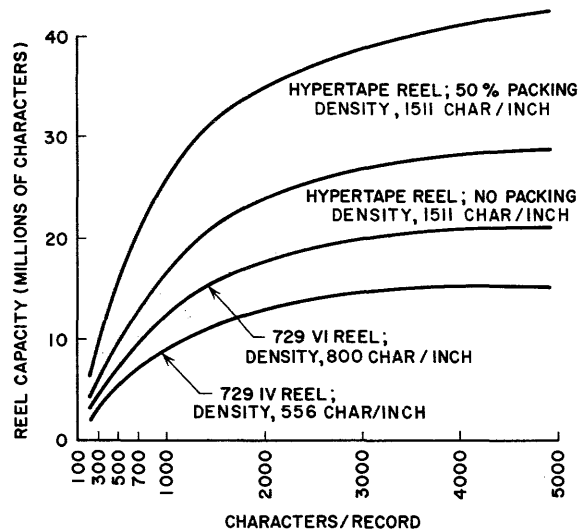


Figure 4. Comparison of Character Capacities—Hyper and IBM 729 Reels.

cartridge for the next job can be inserted in the ACL while the current cartridge is being processed. Then, under program control, the processed cartridge can be unloaded, and the cartridge which was stored in the ACL can be loaded automatically, all within approximately 40 seconds.

The average read-write access time is 4.2 milliseconds, and the average gap is 0.45 inches. The fast processing time of the Hypertape system is accomplished by the high data rate, narrow gaps, short access times, ability to read backward, and a very low rerun time due to extremely reliable operation.

### THE IBM 7340 TAPE DRIVE

A major requirement of the 7340 was high reliability. To help achieve this, the tape is designed so that there is no physical contact with the oxide side of the tape during motion (Figure 6). The vacuum columns hold the tape taut over the single, rubber-surfaced capstan. The capstan can accelerate the tape to nominal velocity or decelerate it to a halt in 3.0 milliseconds. Note that the wear of such motion occurs on the *Mylar* side of the tape. When the tape passes by the read-write head, the oxide side of the tape rides on an air bearing and hence undergoes no wear.

Once a tape is initially loaded into the dust-resistant cartridge, it is normally opened only in

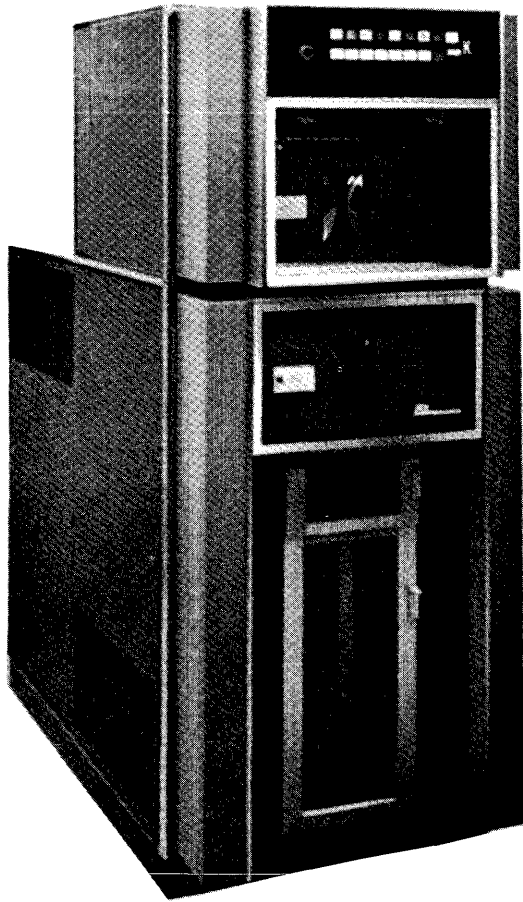


Figure 5. The 7340 Hypertape Drive with Automatic Cartridge Loader Option.

the pressurized chamber of the 7340. However if the tape in a cartridge must be changed, the front of the cartridge can be removed easily by means of a small Allen wrench. To load a cartridge into the 7340, the operator must raise the top cover of the 7340, and lower the cartridge into the cartridge receiver (Figure 7). When the top cover is closed, the tape reels in the cartridge are moved backward to engage the reel hubs, the servo motors on the hubs release the tape from both reels, the vacuum in each column draws the tape into the columns, and the head moves into place. The load operation takes less than 15 seconds. After tape is loaded into the columns, it is backspaced a short distance; thus the overall effect of the unloading and later re-loading is to position the head slightly closer to

the Beginning Of Tape (BOT) than it was before unloading.

Note that tape can be loaded and unloaded without the operator's hands touching the tape. Also, since a cartridge contains two reels, a tape no longer need be rewound before it is unloaded. This allows off-line rewinds, or a very short search on-line for the last record processed before the tape was unloaded. Rewinding can occur either under program control by means of a Control command, or by pushing the Rewind button. Rewind occurs at 112.5 inches/sec (normal processing speed) for 10 seconds; if the BOT mark is not encountered, the head moves  $\frac{1}{4}$  inch away from the tape and a 225 inches/sec rewind "in the columns" is initiated. Rewind continues at full speed until the BOT is sensed; this is made possible through the continuous tape control provided by the capacitive sensing of the tape location in the vacuum column.

As previously mentioned, each cartridge can be "file-protected" either under program control or manually. The control instruction "Set File Protect" will cause a mechanism in the 7340 to depress a plunger on the back of a

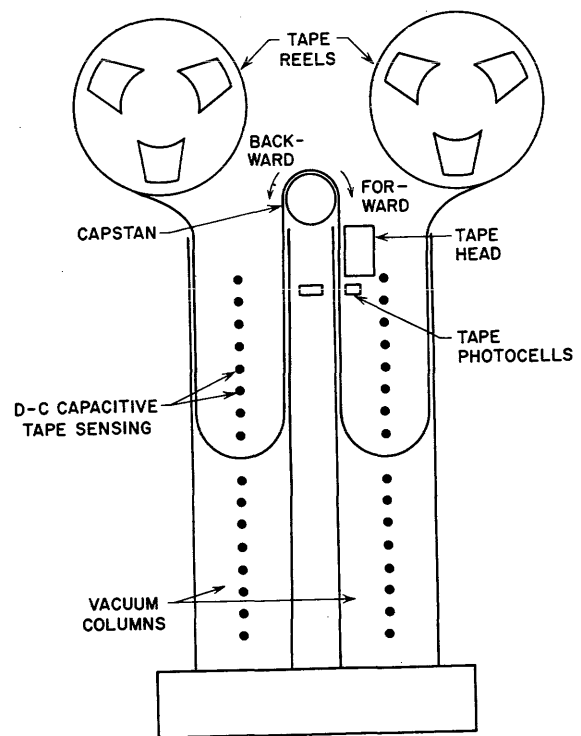


Figure 6. Tape Path in the 7340.

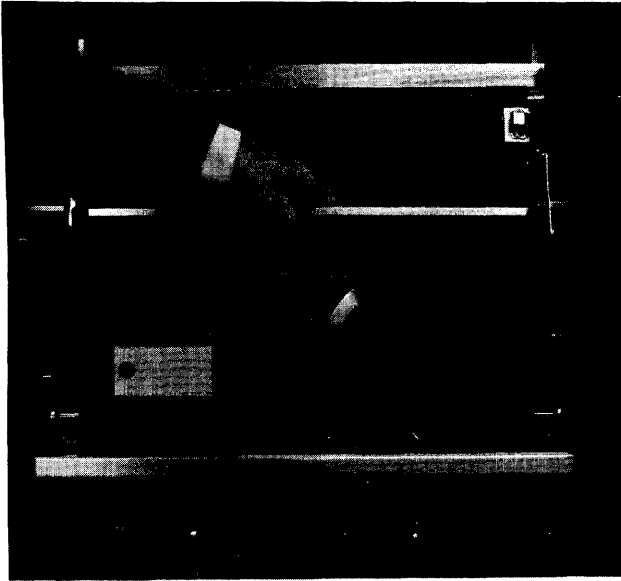


Figure 7. Inserting a Tape Cartridge.

loaded cartridge; this plunger can also be depressed manually. When the cartridge is file-protected, an indicator on the back of the cartridge displays the letters "FP." The cartridge can be "un-file-protected" only by manually releasing the plunger by means of a slide-release mechanism (Figure 8).

The Beginning Of Tape (BOT), End Of Tape (EOT), and End Warning Mark (EWM) are detected by three photosensed markers. Backward tape motion is halted when the BOT marker is sensed. The marker is about 15 feet from the physical beginning of tape on the take-up reel; it consists of 12 holes occupying about 1.5 inches along the edge of tape nearest the drive. Forward motion of tape is halted when the EOT marker is sensed, and hence tape cannot run off the end of the reel. This marker is about 15 feet from the physical end of tape on the supply reel; it also consists of 12 holes, but they are located midway between the edges of the tape. The EWM is about 40 feet from the EOT marker. It consists of 23 holes occupying about 1.5 x 0.07 inches along the edge of tape nearest the operator; the holes do not interfere with recording of data on tape. When EWM is sensed on a selected drive, the end-warning-

area status indication in the IBM 7640 is available to the program.

The Automatic Cartridge Loader (ACL) can be used very effectively if a number of tapes must be loaded at the same time for a new job. During the processing of the previous job, the operator can insert the next cartridge to be processed in the "load storage position" (back of the ACL). Then, under program control, all the processed cartridges can be unloaded at the same time and deposited in the "discharge storage position" at the front of the ACL, and the next cartridges to be processed will be loaded in. The entire load operation of all the drives will take approximately 40 seconds, and is accomplished without operator intervention.

A cartridge may also be loaded into the ACL-equipped drive by direct operator control. The operator merely places the new cartridge in the load-storage position of the ACL, and when the door is closed the cartridge automatically is loaded into the 7340, if no other cartridge was in the loaded position. If a cartridge was in the loaded position, the operator would have to push the Unload button. The loaded cartridge would then be placed automatically on the discharge

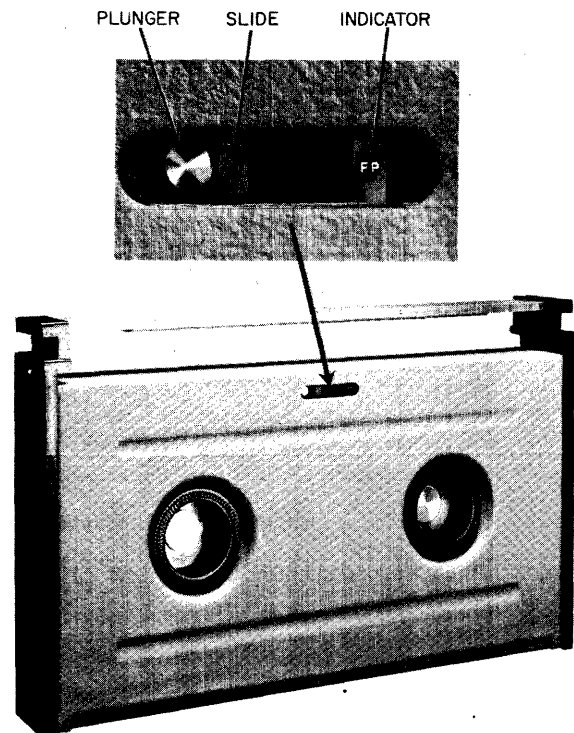


Figure 8. Cartridge File-Protect Device.

shelf at the front of the ACL, and the other cartridge would be loaded.

**SIMPLEX INTERFACE**

The simplex I/O interface is a set of lines which connect the 7640 to the computer. There are a total of 33 lines, 18 from the computer to the 7640, and 15 from the 7640 to the computer. (See Fig. 9.) Generally speaking, any signal must be maintained until its response is provided by the receiving unit.

The 7640 can be operated in one of three modes: CE, Ready, or Diagnostic. Normally the 7640 is operated on-line in Ready mode, but for testing and repair it can be operated off-line in CE (Customer Engineering) mode. Diagnostic mode is identical to Ready mode except that three switches on the CE panel are operable. These will be discussed later.

Initially, the computer and 7640 are interlocked by the Operational Out line and the Operational In line respectively. The Operational In line is conditioned by the 7640 if it receives the Operational Out line from the computer and it is not in CE mode.

An operation is initiated on one of the four command lines: Read, Write, Control, or Sense. Upon receipt of the command, the 7640 will answer with a signal on the Command Response line.

During both a Write and a Control operation, information is transferred from the computer to the 7640 via the Write Bus (9 lines). The 7640 asks for the information by means of a signal on the Service Request line, and when the information is available on the Write Bus, the computer answers with a Service Response. The Write command is terminated by a STOP, and the Control command is terminated by either a STOP or a special instruction called an End of Sequence (EOS). The 7640 indicates successful completion of the command by sending an End to the computer, or an unsuccessful completion by means of an Unusual End. In either case, the computer responds with End Response. The cause of an Unusual End may be determined by analysis of the status information obtained by a subsequent Sense command.

During both a Read and a Sense command, information is sent to the computer from the 7640 over the Read Bus (9 lines). When the information is available on the Read Bus, the 7640 sends a Service Request to the computer. When the information has been accepted, a Service Response is returned. During a Read operation, the transfer of information occurs until a STOP is received from the computer or the end of the record is reached. At this point the 7640 indicates a successful completion of the Read by sending an End to the computer or an unsuccessful completion by means of an Unusual End. The computer responds with End Response to either case. Since the Sense operation is simply a request for the status information of the 7640, only an End can occur at its completion. The 7640 will indicate there is status information on the Read Bus by means of a Service Request, and the computer will answer with a Service Response when this information has been accepted. The operation will be terminated when STOP occurs or when 14 "bytes" of information have been sent to the computer. End response will answer the End signal.

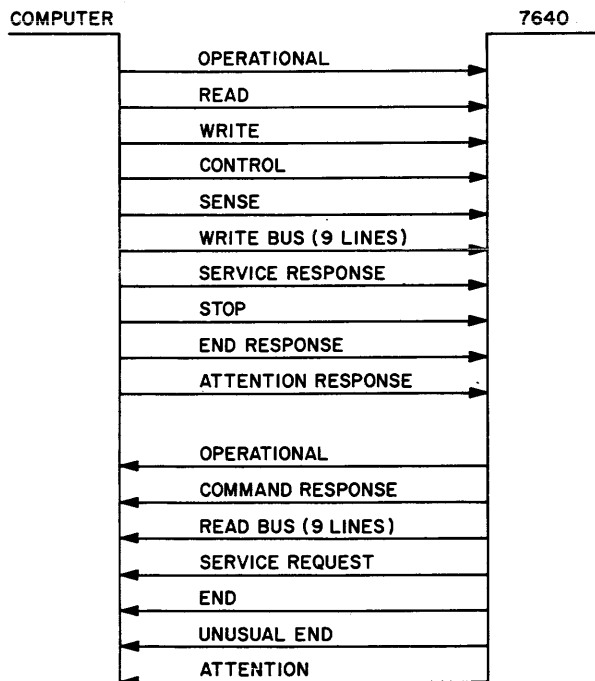


Figure 9. Interface Lines.

## WRITE OPERATION

As previously defined, the Write section of the 7640 is time-shared between both channels. If a Write command occurs in Channel A, and then later a Write command occurs in Channel B, the command in Channel B will be stacked in the 7640 until Channel A releases the Write section.

Upon receiving a Write Command, the 7640 tests for the following two error conditions:

1. Operator Required—indicated if the selected drive is not ready;
2. Program Check—indicated if the selected drive is not loaded, or if it is file-protected, busy, or at EOT.

If either of these conditions exists, an Unusual End will be signaled immediately. If neither exists, a write Delay must be taken before actual writing can commence. If the Write Command occurs while the tape is stationary and the 7340 is in Write status, a 3.1-msec delay is taken before writing is started. The Start specification of the 7340 is 3.0 msec. To test if the proper speed has been achieved, a burst of zeros is written in *one* track for a certain period of time; this is called the "velocity burst." The 7340 has a two-gap head; hence, information written on a tape passes under the read head approximately 1.3 msec later. At this time the duration of the velocity burst is checked. If the tape was not up to speed when the velocity burst was written, the burst's duration will be less than expected.\* A variation of approximately minus 10% is allowed before an Unusual End is signaled. The error is identified as a Track Start in the Sense Data, and the programmer has the option of rewriting the record.

As previously stated, the stop time of the 7340 is 3.0 msec. After the Move Tape line from the 7640 to the 7340 is deconditioned, there is a period of 300  $\mu$ sec during which the line could be conditioned again. This is allowed because of the mechanical delays in actuating the stop mechanism. If the Move Tape line does not rise again during this time interval, the tape must be allowed to stop completely before it can be told to move again. However, the 7640 provides to the computer a 1.0-msec time interval, during which the computer can rein-

struct with another Write command and the tape will not stop between Writes. This is accomplished, in conjunction with the above-described 300  $\mu$ sec, by holding up the Move line for 700  $\mu$ sec after the completion of a Write. Upon completion of any Write operation, another Write operation is *anticipated*, and a short write delay of 2.6 msec is started. If the Write command does not materialize in the allotted 1 msec, the tape is stopped. From the time the Move line is deconditioned, a 3-msec interlock prevents the Move line from being conditioned again except during the initial 300  $\mu$ sec. Reinstruct within the 1.0 msec results in minimum gaps and fast access times. This is called a Continuous Write, for which there is no velocity burst since tape continued to move at nominal velocity. Figure 10 illustrates the 1.0-msec Reinstruct.\*

The 7340 is in Write status if the last operation it performed was a Write operation. If a Write command is sent from the computer to the 7640, and the 7340 is not in Write status, a "backhitch" (reorienting of the head in relation to the previous record) must occur. This is because the previous operation could have been a Backspace or a Read, etc., and the exact position of the read-write head in the gap would be unknown. A Write without reorienting the head could result in a partially erased record in the gap. Thus, in order to erase a proper gap, a Write command, with the 7340 not in Write status, will cause the tape to backspace until the previous record is encountered. The tape will stop, and then move forward out of the record. Just as the read head leaves the record, Write status will be set, and a 2.6-msec Write delay will be taken at full velocity. In addition to eliminating the partially erased record in the gap, the "backhitch" results in a minimum interrecord gap.

As previously mentioned, the Hypertape system uses a phase-encoding type of recording rather than NRZI. With phase-encoding recording a signal exists for both ones and zeros, with their differentiation depending on the phase (or direction) rather than on the magnetic strength of the recorded signal. Thus, phase encoding is less sensitive to noise than NRZI, and hence is inherently more reliable.

\* Patent applied for.



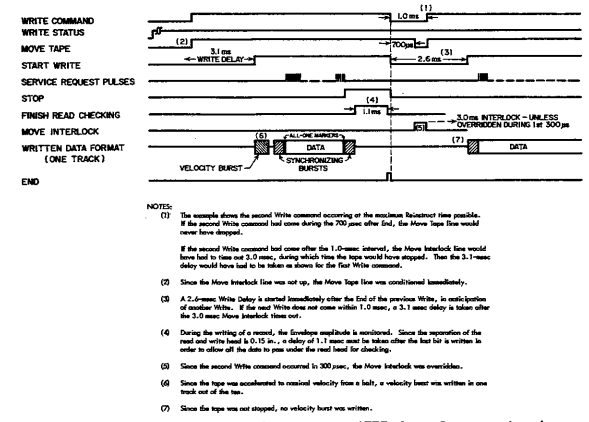


Figure 10. 1.0-msec Reinstruct (Write Operation).

Figure 11 illustrates the phase relationship of a one and zero. As mentioned, the data rate of the 7640 is 170 kc. If the data is all-ones or all-zeros, this results in 340,000 flux changes/sec to produce the 170,000 characters/sec. When data switches from a one to a zero or from a zero to a one, a long wavelength results. Hence, data consisting of alternating ones and zeros results in 170,000 flux changes/sec. Figure 12 shows a series of bits for one of the ten tracks. Note that a long wavelength marks a change from a one to a zero or vice versa, rather than the presence or absence of a signal.

After the velocity burst is written, or after a 2.6-msec Continuous Write delay is taken, a synchronizing burst of 40 zeros is written on all tracks, followed by a one in all tracks, followed by the data. (See the data format in Figure 10.) After a STOP is received from the computer by the 7640, another all-one marker is written, followed by another synchronizing burst of 40 zeros. Hence, the record is symmetrical and can be read backward or forward. The synchronizing bursts are used during reading to synchronize the read clocks and phase the data properly. The all-one marker indicates that data will follow. As previously described, the transfer of information from the computer to the 7640 over the 9-line Write Bus occurs on a Service Request, Service Re-

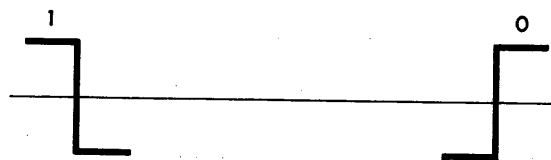


Figure 11. IBM Phase-Encoding "1" and "0" Waveforms.

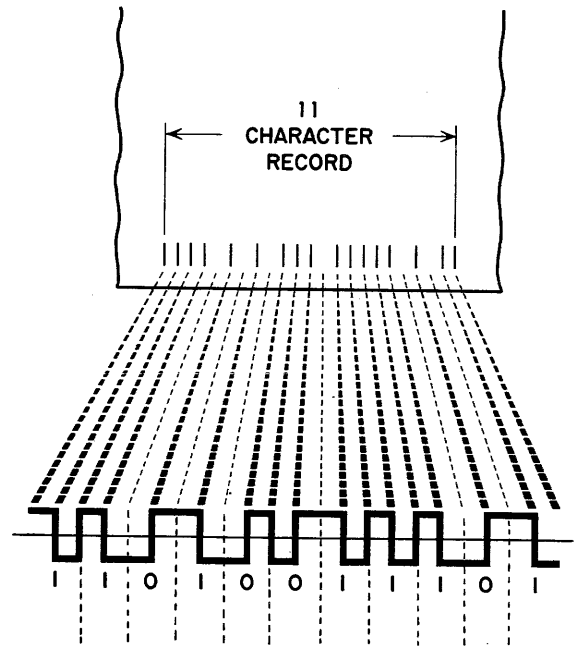


Figure 12. Waveforms for a Series of Bits.

sponse basis. Eight of the lines are for data, and the ninth is a parity check. The 7640 checks the parity of every character received on the Write Bus, and will call an Unusual End at the completion of the Write operation if incorrect parity is received. If the computer does not answer a Service Request with a Service Response before the next Service Request occurs, an Overrun Error is called resulting in an Unusual End.

In alphanumeric mode, only six of the eight lines contain data for each Service Response. In packed-numeric mode, two 4-bit characters are sent to the 7640 with each Service Response. Actually, no packing or unpacking is done in the 7640. It simply writes the information on the Write Bus on tape, along with two check bits. The check bits are generated during the Write operation according to the following two equations.\*

$$C_0 \vee I_0 \vee I_1 \vee I_3 \vee I_4 \vee I_6 \vee I_7 = 1$$

$$C_1 \vee I_0 \vee I_2 \vee I_3 \vee I_5 \vee I_6 = 1$$

where  $\vee$  means Exclusive Or,

C is a check bit, and

I is an information bit.

\* Patent applied for.

The generated check bits are rechecked again during writing, and if they are wrong, a Code Check indication is stored for later sensing, and an Unusual End will occur at the termination of the Write operation. The check bits are regenerated during a Read operation and are used in conjunction with the amplitude of the Read signal to correct all single errors and most double errors.

During a Write operation, as the written data passes under the read head it is monitored by a Motion Integrator. If the signal amplitude of the envelope in any one track drops below a specified level, an Envelope Check is stored in the 7640 to be later sensed. This error will result in an Unusual End at the termination of the Write operation. At this point it is recommended that the programmer perform a Backspace and an Erase over the faulty section of tape. The Erase is called an Erase Long Gap (ERG), and is a Control instruction. It will result in the erasure of eight inches of tape. Then the error record should be rewritten.

When the EWM is sensed by the 7340 during a Write operation, an Exceptional condition is stored for sensing and an Unusual End will terminate the operation. Hence the programmer is informed he is near the end of tape.

## READ OPERATION

Channel A and B time-share the Read section in the same manner they share the Write section. A Read command will be stacked in one channel if the other channel is utilizing the Read section. When the Read section is available the 7640 checks for Operator Required and Program Check, as described for a Write operation. If no errors exist, the Move Tape line is activated to the 7340, and a 2-msec Read delay is taken. After this delay, the Motion Integrators start to look for a record. The ten tracks are divided into three different groups, and a signal in one track of each of the three groups will identify a record. The group approach was used to eliminate noise from being recognized as a record. As in writing, 1 msec is allowed by the 7640 for a Read Reconstruct without stopping tape. Again the Move Tape line is held up for 700  $\mu$ sec after recognizing the end of data in a record. The additional 300

$\mu$ sec is the time during which the Move Tape line can be deactivated and activated again without affecting the stop mechanism of the 7340.

Clearly the optimum use is made of the Hypertape system if the tape does not stop between records.

Once a record is recognized, a variable-frequency clock (VFC) in each track must be given time to synchronize with data. Since there is a signal for both a one and a zero, the problem of a clock dropping out of sync during a long string of zeros does not exist. After approximately 25 zeros of the start burst have been read, the VFCs should be synchronized, and the phase of the signal in the detection circuitry must be corrected if necessary. The detection circuitry will be interpreting the start burst as either zeros or ones, since it will either be in or out of phase respectively. At this point, the data is sampled. If the data is not recognized as zeros, the phase is incorrect and has to be inverted. After this has been done, a search is started for the all-one marker which will indicate that data will follow immediately.

One problem in reading a wide, high-density tape is skew in the characters across tape. This is handled in the 7640 by means of a six-position skew buffer and a Read-In Counter (RIC), for each track, which counts the bits 1 through 6 which have been read from tape. Pulses generated from the VFCs step the RICs midway between the data bits and hence act as gates into the skew buffer. A single Read-Out Counter (ROC) controls the reading of deskewed characters. When all the RICs have stepped ahead of the ROC, a character has been deskewed and can be read from the skew buffer into the Error Correction Register. Between four and five bits of skew can be handled in the skew buffer. Under normal operation the skew is less than two bits. If the skew is so great that the RIC overlaps the ROC, an "excessive skew error" is stored for sensing and an Unusual End is signaled at the termination of the Read operation.

Once the characters have been deskewed and gated into the Error Correction Register, the check bits  $C_0$  and  $C_1$  are regenerated and compared with  $C_0$  and  $C_1$  read from tape.

The bits can be divided into three zones from the two check-bit equations:

Zone 1 contains the bits only in the  $C_0$  equation

Zone 2 contains the bits only in the  $C_1$  equation

Zone 3 contains the bits only in both the  $C_0$  and  $C_1$  equations

Hence,

Zone 1 contains bits  $C_0, I_1, I_4, I_7$

Zone 2 contains bits  $C_1, I_2, I_5$

Zone 3 contains bits  $I_0, I_3, I_6$

An Envelope Detector in each track monitors the read-detector amplitude. If the signal from the read detector drops below a certain level in one track, the track is "dead tracked." This essentially resets that track for the rest of the record, since it is assumed that the VFC could have lost synchronization. If a parity error is found to occur in the zone containing the "dead track," the track which has been "dead tracked" is assumed to be in error, and the output of that particular track's Error Correction Register is inverted. Hence a zero is made a one or vice versa, and single-error correction is accomplished.

If two tracks in the same zone are dead tracked, or more than two tracks are dead tracked, an uncorrectable error called an Envelope Check is stored for sensing. This will result in an Unusual End at the termination of the Read operation. Two tracks in the same zone are obviously uncorrectable since the parity error would be negated. However, the  $C_0$ ,  $C_1$  equations were generated such that no adjacent tracks would be in the same zone. Hence, all adjacent double errors are correctable along with some nonadjacent double errors. Any double error that does occur is more likely to be an adjacent double error due to the character of tape defects.

There are 45 possible combinations of double errors, of which 33 are correctable. When a correctable double error occurs, a dead track exists in two different zones, with the parity error appearing to be in the third zone. For example, consider the case of a dead track in both  $I_0$  and  $I_1$ . If both bits were incorrect they would negate each other in equation  $C_0$ , but

equation  $C_1$  would appear incorrect. Hence, the parity error would appear to be in zone 2.

If a parity error occurs without a corresponding dead track, an uncorrectable error called a Code Check is stored for sensing and an Unusual End is signaled at the end of the Read operation. For both a Code Check and an Envelope Check, the 7640 sends the first uncorrectable character to the computer with incorrect parity. No further characters are sent to the computer for the duration of the record. This is to prevent uncorrected characters from appearing like packed characters and hence overflowing allotted memory when unpacked. If the complete record is desired, HECF (Error Correction Off) may be programmed prior to the Read command. This Control instruction will allow uncorrectable characters to be transmitted.

Another error that can occur on a Read is an Overrun Error. During reading the 7640 sends a Service Request to the computer when data is available on the Read Bus. A Service Response from the computer indicates the information was received. If the Service Response does not come back from the computer before the next Service Request is issued, an Overrun condition exists, and the computer could miss a character. The Overrun Error is stored for sensing later and an Unusual End is initiated at the termination of the Read operation.

## CONTROL OPERATIONS

The Control command transmits instructions such as Backspace, Backspace File, Space, Rewind, Write Tape Mark, etc., to the 7640 from computer core storage. Each instruction is defined by an 8-bit code. The instruction is assimilated in the 7640 Instruction Register by means of two 4-bit bytes. The computer sends a Control command to the 7640, which responds with a Command Response. The 7640 then obtains two 4-bit bytes of information from the computer through a Service Request, Service Response exchange. The instruction is decoded and the 7640 checks to see if the operation can be done. If the instruction is a Select, a third 4-bit byte is requested to identify the drive to be selected. If there is an error condition, an Unusual End is signaled immediately to the

computer. If there is no error condition, the operation is started. Upon successful completion of the operation, the 7640 will request two more bytes of information, assemble them and decode the instruction, etc. This processing of a so-called "chained sequence of instructions" will continue until either a STOP is received from the computer or an End of Sequence (EOS) instruction is received. Either will cause the 7640 to terminate the Control command with an End signal. At all times any error condition will break the chain and an Unusual End will be sent to the computer. There are five "free-running" control operations, HCHC (Change Cartridge), HCCR (Change Cartridge and Rewind), HRUNL (Rewind and Unload), HFPN (file Protect ON), and HRWD (Rewind). These operations are *initiated* in the 7340 by a Control command, and then the 7640 continues on to the next instruction as if the free-running instruction had been completed. A drive performing a free-running operation is in Busy status, but the 7640 channel which initiated the operation is free to perform other instructions, internally, or on other drives. Additional instructions to a busy drive must be delayed until the drive indicates the completion of the free-running operation with an Attention signal.

Upon receipt of an Attention pulse, the computer responds with an Attention Response. All Attention pulses are stored in the 7640 relative to a specific drive address, such that the program can sense to see which drive completed the free-running operation. The next operation (other than a Sense) done on that selected drive resets the stored Attention.

A group of control instructions exists for diagnostic purposes only. They allow diagnostic functions to be performed on-line under program control. These functions can also be performed off-line by means of toggle switches on the Customer Engineering panel (as described in the following section on "Customer Engineering Facilities").

#### SENSE OPERATION

A Sense operation enables the computer to learn the cause of an Unusual End, the status of the 7340 and 7640, and which 7340 signalled Attention.

A Sense command from the computer causes a Command Response from the 7640. Then the 7640 gates the status condition of the 7340 and 7640 to the computer by means of 14 four-bit bytes of information over the Read Bus. Again the transfer of information occurs on a Service Request, Service Response basis. The sense bytes contain information such as Operator Required; the selected drive address; reasons for a Program Check or Data Check—e. g., Selected drive Busy, or a Code Check, respectively; Attention pulses from drives which completed free-running operations; Diagnostic conditions; etc.

The Sense operation is terminated either by a STOP signal from the computer or by the 7640 after the 14-byte transfer is complete. Since the Sense operation is merely a status transfer, it can be terminated only by an End, rather than an Unusual End.

#### CUSTOMER ENGINEERING (CE) FACILITIES

In order to use the entire CE section of the 7640 for off-line debugging, the Mode switch must be set to CE Test. However the 7640 can also be operated in a Diagnostic mode which makes some of the CE switches functional during computer usage.

The CE panel (Figure 13) attempts to simulate a computer. A three-step program loop is available with all four commands, Read, Write, Control, and Sense, and also with HNOP (No Operation), HBSR (Backspace), and Read Backward. Switch controls govern the tape selected, the number of characters/record during a Write, and the loop interval (the time between the termination of one command and the issuance of the next). The loop interval can vary from eight microseconds to two seconds.

A Write command causes the information in the bit switches of bytes 1 through 4 to be written on tape until the record length is satisfied, determined by the Characters per Record switch. A Read operation can then be performed to check the record written. If the Read Compare toggle is on, the characters read from tape are compared bit for bit with the characters set up in the byte switches. The Stop On Check toggle switch will cause the loop to stop

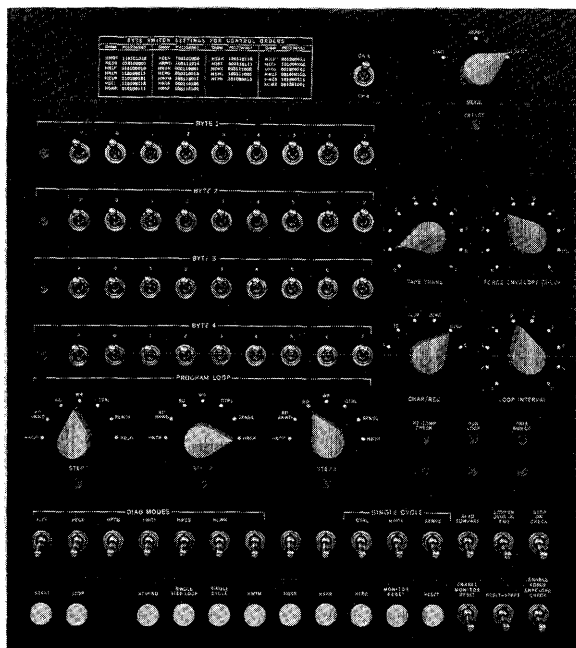


Figure 13. CE Panel on the 7640.

immediately on any error, including any compare error that might occur. It is also possible to stop at the *end* of an operation on all errors, except compare errors, by using the Stop on Unusual End toggle.

For debugging purposes, a Single Cycle button and a Single Step Loop button are provided. The Single Cycle button is used in conjunction with the appropriate Single Cycle toggle switch to single-step through a Write, Control, or Sense; while the Single Step Loop button merely steps the program loop one step at a time, as opposed to running continuously if the Start button is pushed.

A number of indicators in the 7340 are not reset at the start of a new operation if a switch called Enable Monitor Reset is on. This toggle prevents all normal resets from resetting these holdover indicators until a Monitor Reset button is pushed. This allows selected indicators to be monitored over a period of time.

The error-correction circuitry can be tested through manipulation of a Force Envelope Check switch. During a Write operation this switch will cause groups of 16 information bits

to be degated from tracks selected by a rotary switch. Hence, during a Read operation, the Envelope Detectors should sense the absence of signal in these tracks and indicate the appropriate "dead track," causing the error correction and detection circuitry to be exercised.

The Reset Start toggle switch is used to help debug a failing operation. Operations which are functioning properly are set up in steps 1 and 2 of the program loop, and the failing operation is set up in the third step. Pushing the Start button will cause the first two steps to be performed, and the failing third step will be attempted. After a certain time has elapsed, depending on the loop interval selected, a Reset will be initiated and the loop will step back to program step 1. Hence, an oscilloscope can be conveniently used on the failing operation, since it is repetitive.

Another very powerful CE tool is the Loop Write to Read (LWR). It essentially gates the information, which is normally written on tape, directly from the Write section into the read-detection circuitry. Thus the tape path is eliminated. Two modes associated with LWR are Write Clock Fast (HWCF) and Write Clock Slow (HWCS). These modes are selected by toggle switches and cause either a 10%-fast oscillator or a 10%-slow oscillator to be gated into the write clock. The functions performed by these toggle switches are also programmable.

If the 7640 mode switch is in the previously mentioned Diagnostic position, the relation between the computer and the 7640 is exactly the same as in Ready mode, except that the following three toggle switches on the CE panel are operable:

1. Enable Monitor Reset
2. Stop On Check
3. Enable Force Envelope Check

Hence, these three switches may be used while running programs on the computer.

#### OFF-LINE EQUIPMENT

A low-speed, single-channel control unit, the 7641, is available to connect into an IBM 1401, an IBM 1410, or an IBM 1460 computer. The 7641 controls a drive, the 7340 Model 2, which

has a data rate of 34kc. There is complete compatibility of tapes between the 7340 Model 1 and the 7340 Model 2. Therefore, for example, a job can be processed on an IBM 7094 computer using 7340s, and the off-line printing can be handled on a 1401 or 1410 using a 7340 Model 2.

#### HYPERTAPE MODIFIED FOR IBM SYSTEM/360

A version of the Hypertape system is now offered as I/O for the new IBM System/360. A slightly modified 7340, designated as the 7340 Model 3 Tape Drive, is controlled by a single-channel control unit, the IBM 2802, attached to a CPU channel through a new interface. The 7340 Model 3 tape speed is 112.5 inches/sec, but the bit density has been increased to 1511 or 3022 bits/inch. Hence the data rate is either 170,000 or 340,000 alphanumeric characters/sec, or 340,000 or 680,000 packed-numeric characters/sec. The desired data rate is program-selected. The lower data rate allows interchangeability with the 7340 Model 1 or 2. The nominal gap has been reduced to 0.38 inches and the average access time is 3.5 msec.

The 2802 consists of five major sections—Read, Write, Control, Sense, and Customer En-

gineering Facilities—all of which function in a manner similar to those described in the 7640. The major differences are:

1. Employment of miniature circuitry (IBM's Solid Logic Technology);
2. A modified control section due to the System/360 interface;
3. Reduction of the Sense information to four bytes;
4. No overlap of operations because of the single-channel design;
5. Increased data rate under program control.

The 2802 Hypertape control unit can control up to eight drives. However, an optional sixteen-address feature enables the 2802 to address as many as sixteen drives when used with the IBM 2816 Switching Unit Model 2, which is available for switching 4, 8, 12, or 16 drives to any one of up to four 2802 units (under program control).

The 2802 controlling 7340 Model 3 Tape Drives maintains the high reliability previously described for the Hypertape System, while increasing the data rates by a factor of two, and decreasing the interrecord gap and access time appreciably.

# DESIGN CONSIDERATIONS OF A RANDOM ACCESS STORAGE DEVICE USING MAGNETIC TAPE LOOPS

*Andrew Gabor, Janos T. Barany, Louis G. Metzger,  
and Eleuthere Poumakis  
Potter Instrument Company, Inc.  
Plainview, New York*

## SUMMARY

The random access storage device described in this paper is a cartridge loaded machine which uses continuous magnetic tape loops for storage medium. Cartridges may be built to cover a range of storage capacities; the one described in this paper contains 16 loops of one inch wide tape, each approximately 3 feet in circumference.

Storage capacity of each cartridge ranges from 3 million to 40 million alpha-numeric characters depending on both the cartridge and the machine. Storage capacity in a cartridge may be increased with a corresponding increase in access time. Average random access time for the smallest cartridge is 168 milliseconds.

In the interest of long tape life the loops are air floated and, in one machine configuration, are stopped when not engaged in data processing. During read-write operation only the one selected loop is in motion; all other loops remain stationary until specifically addressed.

Lateral head positioning is accomplished by means of a binary whiffle-tree mechanism. A novel head construction permits "write-wide, read-narrow" operation with a single read-write gap. Information is recorded in single channel serial-serial format, using the self-clocking double transition recording method.

Typical loading time for a cartridge is 18 seconds.

## *General Design Considerations*

The tape loop storage device was designed to have performance characteristics comparable to those of some other random access devices of the replaceable cartridge type, such as machines with disk cartridges and magnetic cards. While the inherent performance capabilities, such as storage capacity and access time, of the tape loop approach were found to be quite similar to those of the other devices, there are a number of factors in favor of the tape loop system.

In contrast with disks, the tape loop system uses a flexible substrate and thus renders the cartridge free of critical mechanical requirements, resulting in a package that is lower in cost and less susceptible to damage by external mechanical influences, such as shock and vibration.

In contrast with magnetic cards, the tape loops are at all times under the positive mechanical control of the machine, the handling of the medium is gentle and shock-free, all moving parts have small mechanical excursions, and the recording medium never makes contact with any solid matter, thus minimizing wear and contamination. The inactive side of the substrate is in light non-slipping contact with a plastic-coated capstan, which is the only surface the

loop ever touches, all other points being air floated. The results are highly reliable operation, low maintenance cost, and long life expectancy on both machine and cartridge.

The following paragraphs describe in some detail the operating principles and performance characteristics. The discussion will be limited to the minimum capacity cartridge.

### *The Basic Storage Element*

A 38 inch long continuous loop of standard computer grade magnetic tape forms the basic storage element (see Figure 1). The loops are first cut to length and then spliced using an ultrasonic sealing process. Special tooling permits the fabrication of loops without manual handling, and with mechanical handling only in the immediate vicinity of the seam. When recording on the loop, the splice area is magnetically sensed and avoided. Thus, the longitudinal recording format is keyed to the splice.

The tape is recorded serially a single channel at a time with a longitudinal density of 750 bits per inch. There are 48 tracks recorded across the one inch width of the tape.

There is, as will be seen later, a plurality of such loops in each cartridge.

### *The Individual Tape Drive Unit*

Figure 2 is a schematic illustration of the tape drive principle. The upper and lower turn-

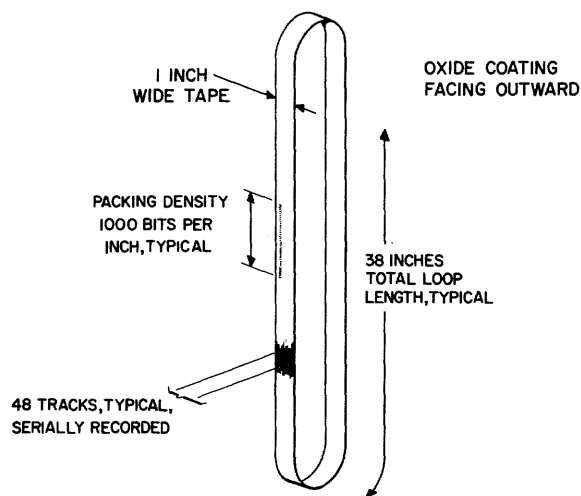


Figure 1. The Basic Storage Element.

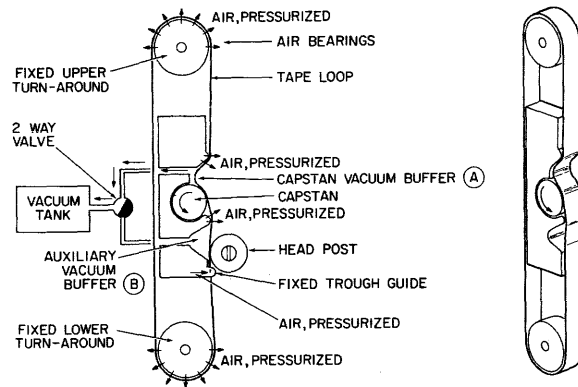


Figure 2. Individual Tape Drive Unit in "Drive" Condition.

arounds are hollow cylinders provided with air holes which supply pressurized air for the air bearings. The tape loops turn around stationary cylinders called "turn-arounds." The loops are permanently supported by the turn-arounds and both are integral with the cartridge while everything else in the figure is part of the machine.

The "two-way valve" on Figure 2 is shown to be evacuating buffer "A" (capstan vacuum buffer), causing the tape to be wrapped around the constantly turning capstan. The wrap and air pressure combined with the friction between the tape and the rubber coated capstan are sufficient to drive the loop which, as a result of air lubrication, offers virtually no drag. Thus, tape drive is afforded by friction against the non-coated side only.

As mentioned above, the non-coated side of the tape runs over air cushions except for the capstan where the driving is effected. Let us now examine the coated side. As may be seen on Figure 2, the only questionable point is the head since there is nothing else on the coated side that the tape could make contact with. While there is no pressurized air supplied to this point by the machine, the tape flies on air-cushion nevertheless, for the fast moving tape (400 inches per second) and cylindrical head column form a perfect foil bearing.<sup>1,2,3</sup> Thus, the pressure which provides the tape with an air cushion at the head is derived from aerodynamic effects. The flying altitude of the tape may be controlled by tape speed, tape tension, and radius of head column. However, for a



given set of these conditions the flying altitude remains remarkably stable and consistent.

Figure 3 shows the tape drive unit with the valve turned to evacuate buffer B. In this condition the tape is drawn away from both the capstan and the head and consequently remains stationary. This is the stand-by condition for any loop not in the process of data transfer.

### The Multiple Loop Cartridge

A random access storage device using the single loop with its drive, as described in the preceding two paragraphs, would be limited to a rather small storage capacity. Making the loop longer would increase the storage capacity but only at the cost of increasing the access time in roughly the same proportion. To obtain the desired storage capacity and still maintain a reasonable access time, multiple loops must be used. Figure 4 shows how an array of 16 loops is contained in a single cartridge. Since the turn-arounds air-lubricate every loop, each loop is able to move or stand by independently.

The shell of the cartridge keeps the loops enclosed and protected from dust. However, owing to the flexible medium, the loops are far less sensitive to dust than random access storage devices using rigid media. Indeed, since the loops are floated, they are less critical to environmental conditions than tapes in conventional tape handlers.

While the cartridge is fully enclosed for shelf storage, one side may be opened when the cartridge is loaded into the machine. During operation the front plate of the machine covers the removed side of the cartridge with a small

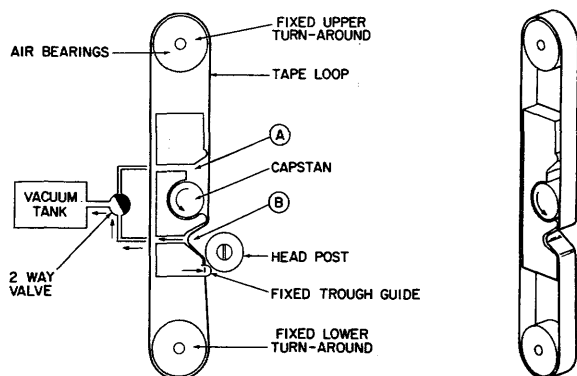


Figure 3. Tape Loop in "Stand-By" Condition.

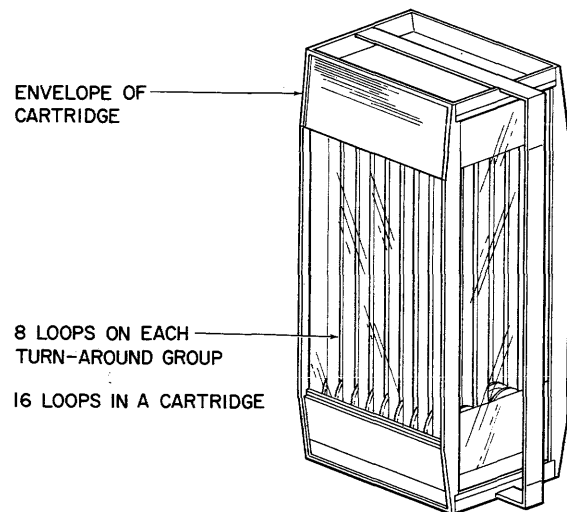


Figure 4. Multiple Loop Cartridge.

clearance to allow outward air flow from the pressurized turn-arounds.

### Multiple Drive Blocks

The multiple loop cartridge requires a multiple tape drive unit in the machine, i.e., a separate drive to each loop. Figure 5 shows how this is arranged. Each of the two blocks has a single capstan long enough to cover the width of eight tapes. Each block has eight separate compartments for both lower and upper vacuum chambers, and eight separate valves actuated by separate solenoids for routing the vacuum to the upper or lower chamber as required for tape driving.

The blocks are easily removable for cleaning the tape path, the capstan, and the head.

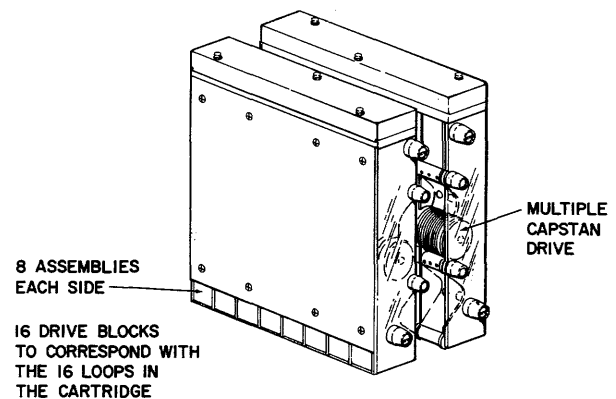


Figure 5. Multiple Drive Blocks.

### Read-Write Head

A single head post, as shown on Figure 6, serves as read-write head for all sixteen tape loops. The head post contains three tracks of read-write head for each tape, giving twenty-four tracks ( $8 \times 3$ ) per head face, a total of forty-eight read-write tracks. The head post is axially positioned to one of sixteen discrete positions, thus covering each of the forty-eight track positions on each tape loop. (Note: the reader may find the recurrence of the number 48 confusing. Actually, 48 being the total number of read-write tracks on the head post as well as the number of tracks per tape loop is merely a coincidence.)

The physical configuration of head and tape was designed to satisfy the requirements of the flying tape. The cone-shaped nose, as will be seen later, plays an important role during the loading of a cartridge.

### Head Positioning Mechanism

Figure 7 is a schematic illustration of the head positioning mechanism. Four solenoids with mechanical stops adjusted for equal strokes act as prime movers. Each actuator has two discrete mechanical positions, thus the complete system may be regarded as a mechanical digital-to-analog converter having a four-bit binary input. The levers, called whiffle-trees, and linkages connect the head post with the

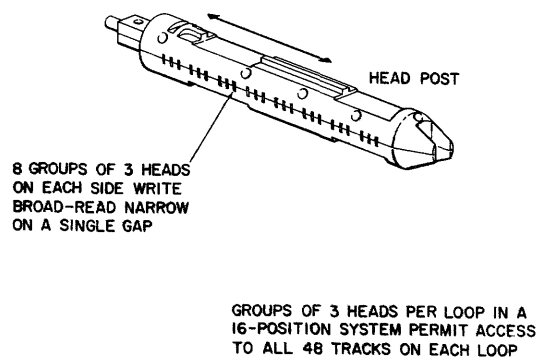


Figure 6. Read—Write Head.

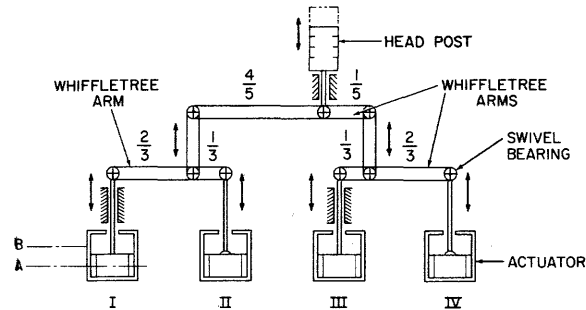


Figure 7. Head Positioning Mechanism.

actuators through backlash-free flexural spring pivots, thus, errors arising from bearing inconsistencies and wear are eliminated.

There are other binary-input mechanical head positioners in existence. The following error analysis will point out the advantages of the whiffle-tree mechanism. Following the notations of Figure 8, the four actuators, according to their binary significance, are designated 1, 2, 4, 8. The connecting levers have the lengths  $l_1$ ,  $l_2$ ,  $l_3$ , and the swivel points are at fractional lengths  $k_1$ ,  $k_2$ , and  $k_3$ , respectively. Let us first find the nominal arm ratios for the error-free case. The following four equations may be written:

$$\left. \begin{aligned} k_1 k_3 S &= d \\ (1 - k_1) k_3 S &= 2d \\ k_2 (1 - k_3) S &= 4d \\ (1 - k_2) (1 - k_3) S &= 8d \end{aligned} \right\} \quad (1)$$

where  $d$  is the required unit increment for the head (or track pitch on the tape), and  $S$  is the actuator stroke (all actuators have equal nomi-

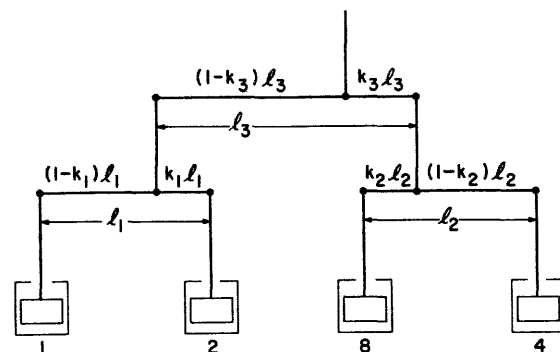


Figure 8. Illustration to the Error Analysis of the Whiffletree Mechanism.

nal strokes). By first adding the four equations, then successively eliminating the variables, the four unknowns are found to be

$$\left. \begin{aligned} S &= 15d \\ k_1 &= 1/3 \\ k_2 &= 1/3 \\ k_3 &= 1/5 \end{aligned} \right\} \quad (2)$$

which is in agreement with the arm ratios indicated on Figure 7.

Let us examine now the effect of an error in the location of any of the swivel points, i.e.,  $k_1$ ,  $k_2$ , and  $k_3$  are assumed to have values slightly different from those in (2). It will be shown that such errors may be compensated for by adjusting the actuator strokes. The four equations in (3) are the same as in (1) except that the strokes are generally all different and  $k_1$ ,  $k_2$ , and  $k_3$  are now known but slightly different from their ideal values.

$$\left. \begin{aligned} k_1 k_3 S_1 &= d \\ (1 - k_1) k_3 S_2 &= 2d \\ k_2 (1 - k_3) S_4 &= 4d \\ (1 - k_2) (1 - k_3) S_8 &= 8d \end{aligned} \right\} \quad (3)$$

The solutions for the strokes

$$\left. \begin{aligned} S_1 &= \frac{d}{k_1 k_3} \\ S_2 &= \frac{2d}{(1 - k_1) k_3} \\ S_4 &= \frac{4d}{k_2 (1 - k_3)} \\ S_8 &= \frac{8d}{(1 - k_2) (1 - k_3)} \end{aligned} \right\} \quad (4)$$

are unique, thus the compensation is always possible.

Let us examine now the effect of a small error in the stroke. For the purposes of the analysis equal nominal strokes and ideal arm ratios will be assumed. Modifying equation (1) for the present model we get

$$\left. \begin{aligned} k_1 k_3 (S \pm \Delta S) &= d + \epsilon_1 \\ (1 - k_1) k_3 (S \pm \Delta S) &= 2d + \epsilon_2 \\ k_2 (1 - k_3) (S \pm \Delta S) &= 4d + \epsilon_4 \\ (1 - k_2) (1 - k_3) (S \pm \Delta S) &= 8d + \epsilon_8 \end{aligned} \right\} \quad (5)$$

where  $\Delta S$  is the magnitude of the error in stroke and  $\epsilon_1, \epsilon_2, \dots$  are the errors in the output head position. For  $\epsilon_1, \epsilon_2, \dots$  we get

$$\left. \begin{aligned} \epsilon_1 &= \pm k_1 k_3 \Delta S = \pm \frac{S}{15} \\ \epsilon_2 &= \pm (1 - k_1) k_3 \Delta S = \pm \frac{2}{15} \Delta S \\ \epsilon_4 &= \pm k_2 (1 - k_3) \Delta S = \pm \frac{4}{15} \Delta S \\ \epsilon_8 &= \pm (1 - k_2) (1 - k_3) \Delta S = \pm \frac{8}{15} \Delta S \end{aligned} \right\} \quad (6)$$

which means that the output error contributions are binary weighted fractions of the input error (error in actuator stroke), the largest contribution being 8/15 or little over one half of the input error. Furthermore, the magnitude of the total error at the maximum output stroke is

$$|\epsilon_1 + \epsilon_2 + \epsilon_4 + \epsilon_8| = \Delta S \left| \frac{\pm 1 \pm 2 \pm 4 \pm 8}{15} \right| \leq \Delta S \quad (7)$$

This means that the maximum output positioning error can never be greater than the largest of the four input errors. This feature marks a contrast between the whiffle-tree positioner and the commonly used mechanism which stacks segments of varying length to obtain mechanical binary to analog conversion. In the latter system the output error is a direct algebraic sum of the input errors. Another advantage of the whiffle-tree positioner is that it permits a fixed mounting of the actuators. This in turn allows the use of ordinary solenoids, thus the need for hydraulic actuators is eliminated.

Typical dynamic responses of the whiffle-tree positioner are shown in Figure 9. The two oscillograms show head position vs. time for minimum and maximum strokes, respectively. It may be seen that the head is positioned to within .001" in 50 msec for the minimum stroke and 80 msec for the maximum stroke.

### Cartridge Loading

The smooth execution of moving the loops into operating position, and out of the machine, requires certain features from the machine. Figure 10 shows how the conical nose of the head post and the retractable constraining arms

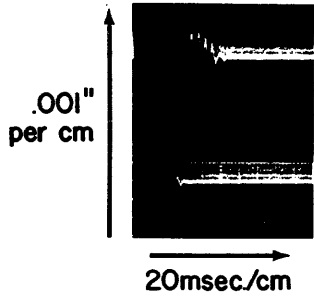


Figure 9. Dynamic Responses of the Whiffletree Position.

act as tape deflectors during loading and unloading of the cartridge. Figure 11 is a photograph of an experimental cartridge in pre-loading position with the head post and one of the tape drive blocks removed for clear visibility.

The loading and unloading involve several automatically cycled steps such as extruding and retracting the constraining arms, turning air pressures on and off, etc. Total loading time, after the cartridge is manually inserted, is 18 seconds.

*Packaging*

Figure 12 shows a photograph of the main panel with the tape drive blocks and cartridge carriage. Figure 13 is a photograph of the finished unit. Overall dimensions are 36" × 25" × 53" high.

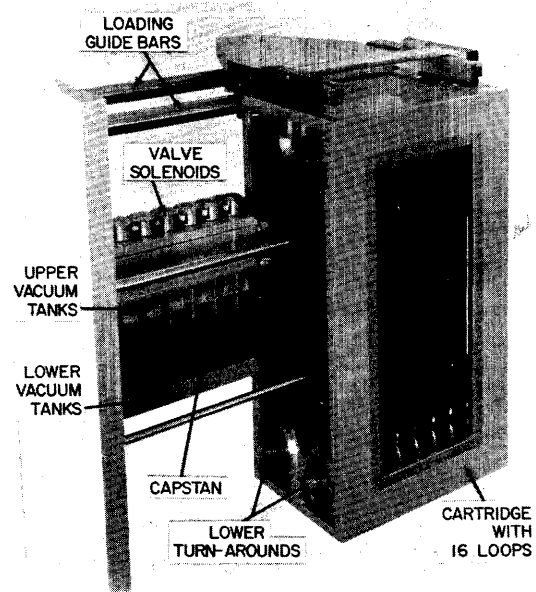


Figure 11. Experimental Cartridge in Loading Position.

*Performance Characteristics*

The following list shows typical operating ranges for various machine configurations.

Loops per cartridge	up to 16
Length of loop, inches	20 - 300
Tape width, inches	1 - 2

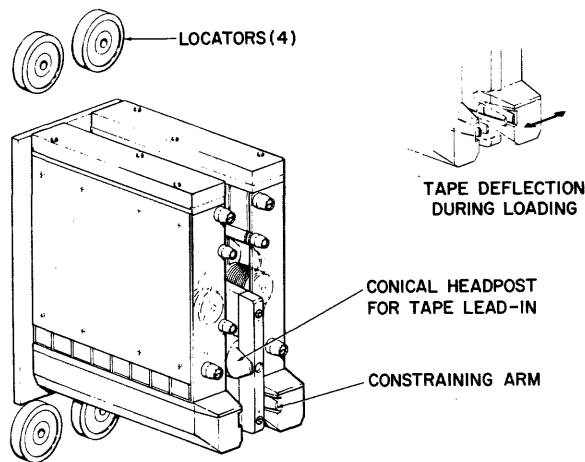


Figure 10. Loading Features.

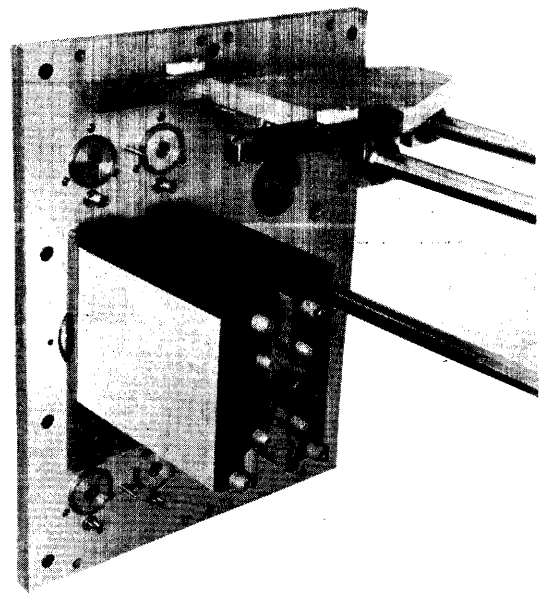


Figure 12. Photograph of Front Panel with Tape Drive Blocks and Loading Carriage.

Tape speed, IPS	400 - 800
Rotation time, msec	25 - 750
Head positioning time, minimum, msec	30 - 40
Head positioning time, maximum, msec	70 - 80
Tracks per inch	50
Longitudinal density, bits per inch	750 - 1000
Information per cartridge, characters	3 million - 40 million
Tape loop life expectancy, num- ber of actuations	30 million

### Conclusion

The concept described in this paper represents the outgrowth of a long evolutionary development of mass storage devices at Potter Instrument Company and elsewhere. The experimental results obtained on the first model and many breadboard tests indicate that tape loop random access storage devices offer performance characteristics comparable to those obtained with other media but have several advantages.

### Acknowledgment

The authors are grateful for the many helpful suggestions made by Messrs. G. E. Comstock, 3rd, T. P. Foley, L. J. Higgins and J. T. Potter.

### REFERENCES

1. GROSS, W. A., *Gas Film Lubrication* (John Wiley & Sons, 1962).

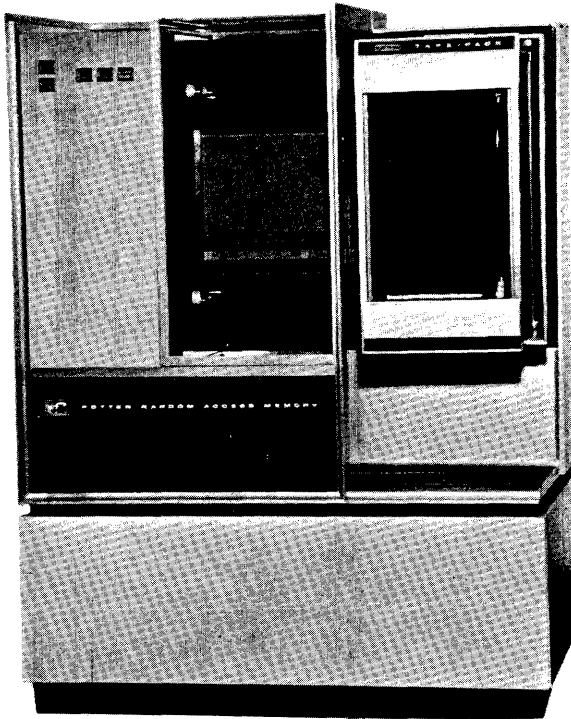


Figure 13. Photograph of Random Access Memory Prototype.

2. LANGLOIS, W. E., "The Lightly Loaded Foil Bearing at Zero Angle of Wrap." *IBM Journal of Research and Development*, Vol. 7, Number 2, April 1963.
3. BAUMEISTER, H. K., "Nominal Clearance of the Foil Bearing," *IBM Journal of Research and Development*, Vol. 7, Number 2, April 1963.



# THE TIME-SHARING MONITOR SYSTEM

*H. A. Kinslow  
International Business Machines Corporation  
Advanced Systems Development Division  
2651 Strang Blvd., Yorktown Heights, New York  
PEekskill 7-6600*

## INTRODUCTION

The IBM Advanced Systems Development Division is currently operating the Time-Shared Monitor System (TSM), an experimental, general-purpose, time-sharing system based on the IBM 7090 Data Processing System. This system is capable of serving 24 remote users simultaneously. It requires a minimum of computing equipment, and gives the remotely located user a maximum amount of control over the 7090 itself; and over the content of his program files within it.

The basic programming language of the system is Fortran Assembly Program (FAP) symbolic.<sup>(1)</sup> There are no restrictions on the type of language capability which can be added to the system, or on the type of programs which the system will accept, compile and execute. At the present time the system contains a FORTRAN compiler, a time-sharing version of the General Purpose Systems Simulator (GPSS)<sup>(2)</sup>, and an interpretive sub-system called PAT (Personalized Array Translator)<sup>(3)</sup>.

Users' programs, and the data files which they operate upon, are permanently stored in an IBM 1301 Disk File. A user of the TSM system submits his program once, either from a remote terminal or by a batch run in the machine room. Thereafter the program is part of the system's memory and can be manipulated at will from the terminal.

The FAP language and assembly capability of the system is nearly identical to that in com-

mon use at other 7090 installations. One instruction (Load Channel—LCH) has been deleted from the 7090 instruction set, and some new instructions have been added. The added instructions allow a complete range of conversational capability, both between a single program and its terminal and among a group of programs associated with a group of terminals.

## THE COMPUTER SYSTEM

Figure 1 is a schematic diagram of the hardware complex in the TSM system. It is a single-processor, single-memory 7090 system. The bulk storage device for programs, data files, and most of the TSM system itself is the 1301 Disk File. A 7320 Drum Storage is used as an adjunct to core memory for program-swapping purposes. Each of these units has its own 7909 Data Channel. There are two 7607 Data Channels for tapes and a 7281-II Data Communications Channel for terminal multiplexing.

The 7090 CPU is equipped with three extra features for time-sharing purposes; relocatability, memory protection and an interval timer. The relocatability feature allows all programs to be compiled at origin 0000 and executed in any available memory space. The memory protection feature sets upper and lower bounds for a program and prevents it from referencing any space outside those bounds. The interval timer, which can be set only by the supervisory program, will generate an interrupt at the end of any predetermined time interval. This in-

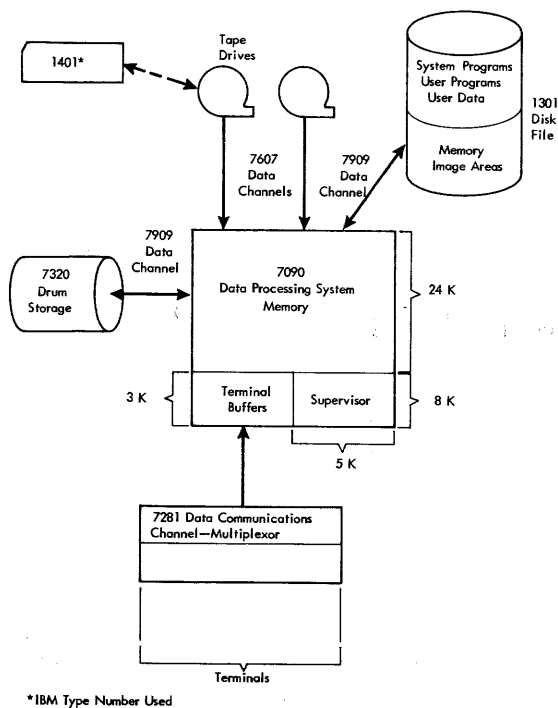


Figure 1. TSM Computer System.

sures that the supervisor can examine the executing program at intervals of its own choosing.

The memory protection feature has one additional virtue; it traps all of the 7090 Input/Output instructions before execution. This gives the supervisor an opportunity to monitor all I/O activity of the executing program.

The 7281 Channel stores data from terminals directly into pre-assigned core memory buffers. A unit of information from a terminal is usually a character, although it can vary in nature and number of bits according to the terminal transmission characteristics. Each character, or unit of information, is accumulated by a "subchannel" in the 7281 and stored in a single word in the pre-assigned memory buffer. When the buffer is full, an interrupt is generated. One of the functions of the TSM Supervisor is to service these interrupts and empty or refill the buffers as needed.

A 7281 subchannel acts as an interface between the transmission line and the computer memory. The multiplexor has physical space for 32 such subchannels, each of which can be thought of as an independent avenue of access to memory. In practice certain timing and buf-

fer-space parameters indicate that the maximum number of simultaneously active subchannels should be approximately 24. Therefore capacity of the system is said to be 24 active terminals, i.e., users.

A subchannel can be designed for virtually any type of terminal. At the present time the terminals on the system are from the IBM 1050 Data Communications System. Subchannels have been designed for an experimental facsimile printer and a cathode-ray tube device. The TSM supervisor has been written to be independent of the nature of the terminal.

### DATA CHANNEL RELOCATION AND MEMORY PROTECTION

All of the above equipment items are either taken from the IBM product line or are readily available special features. There is one system hardware feature which is not standard. This is relocatability and memory protection in the 7909 Data Channel for the 1301 Disk Storage file, and in each of the two 7607 Data Channels for the tape drives. These features have been designed and implemented by our own staff.

Relocation and memory protection in the data channels serve the same purpose as their counterparts in the CPU. They allow channel "programs" to be executed from any available space, and prevent these "programs" from accessing memory outside their boundaries.

These features added to the data channels mean that a user program being time-shared can use tape drives and the 1301 file with a great deal of freedom which would not otherwise be allowed. Without them it would be necessary for the TSM supervisor to either interpret all I/O commands, or restrict the user to only certain limited command sequences. This would not be too great a restriction with respect to the 7607 Data Channels, which have only a limited command set to begin with. But it would cripple the capability of the user to exploit the potentialities of the disk file.

Given relocation and memory protection in the data channels, which means that the channels themselves do a fair amount of monitoring, the TSM Supervisor has its hands relatively free to attend to other work. The net result is that there are virtually no restrictions placed on



a user with respect to Input/Output programming. A program being time-shared can, for example, process its own 1301-stored files in either serial or random fashion; use cylinder, track, or record modes; and establish its own data formats.

## THE PROGRAMMING SYSTEM

The TSM Programming System is a complete, integrated, real-time software complex. It contains its own self-loading and shutdown mechanisms, diagnostics for both programs and hardware, and can maintain itself and its files while operating. It also contains a large and open-ended set of tools for program construction and maintenance. Some years ago these tools would have been called "utility programs." In TSM (and in other time-sharing systems) they are called "service routines."

The program components of the system can be categorized as follows:

### 1. Auxiliary Programs

These are the Startup and Shutdown functions. They are capable of automatic loading and initiation of the system and subsequent unloading and preservation of it. This includes saving and restoration of the 1301 file contents, i.e., user programs and data. The Startup and Shutdown programs can create and maintain a tape which contains the system; they are also the basic disaster recovery mechanisms.

### 2. The Supervisor

The Supervisor is responsible for program scheduling and monitoring, space and facility assignment, and message transmission. In essence its only function is to regulate and protect the system. It resides in lower memory at all times and is written in "read-only" style for self-checking purposes. In a very real sense the Supervisor can be thought of as an extension of the 7090 CPU to enable time-sharing.

### 3. Service Routines

A service routine is a system program which is capable of executing a command received from a terminal. A terminal command is usually a request for the system

to manipulate a user's program in some way; DISPLAY it, for example, or RUN it. The basic set of service routines in the TSM system is designed to allow a remote terminal to be used as an operating console for the 7090. This set is open-ended and has already passed through several evolutionary stages. All service routines are time-shared, and all of them are permanently stored in the 1301 file. Most of the system, in terms of number instructions, consists of service routines which are retrieved and executed on command from the terminal. They execute in time-shared fashion, obeying substantially the same rules as user programs. In fact the only difference between a system service routine and a user program is that a service routine can access the tables and directories of the system while a user program cannot. Service routines, in other words, operate in a "privileged mode."

## TIME AND SPACE SHARING

The basic time-sharing strategy of the TSM System is nothing new. It is essentially the same method employed in the CTSS System at MIT <sup>(4)</sup> and the SDC-ARPA Time-Sharing System at SDC <sup>(5)</sup>. Each program in the set which is being time-shared is allowed to execute for a "slice" of computer time, after which the CPU is turned over to the next program in line. In theory, each program gains access to the computer often enough so that its operator at the terminal appears to have continuous control of the machine. The drum is used to store programs while they are out of memory and waiting for their next slice of time.

Figure 2 illustrates the time-sharing process. Programs A, B, C and D are sharing the machine. Their combined size exceeds the capacity of memory (24,000 words). The solid black areas indicate periods of time during which a program is being either read from the drum into memory, or written from memory on the drum. In the first time slice A is operating, while B and C are being read in. In the second time slice B is operating while A is being written out. The diagram also illustrates the fact that the system is designed to "space-share" core memory as well as "time-share" the CPU.

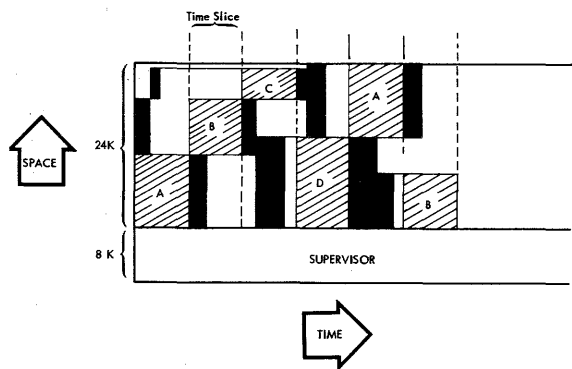


Figure 2. Time and Space Sharing.

The TSM supervisor, which regulates this process, has been designed to use all the facilities of the computing system as efficiently as possible, while at the same time placing the absolute minimum of restrictions on the user. In brief its functions are as follows:

1. Scheduling of programs and transferring of programs to and from core memory.
2. Assignment of core space, drum space and 1301 file space.
3. Monitoring and controlled execution of instructions trapped by the memory protection feature. This includes simulation of some instructions and address translation for most of the Input/Output instructions.
4. Execution of the pseudo-instructions which transfer data between programs, and between a given program and its terminal. These are the conversational mode instructions.

THE BASIC PROGRAMMING LANGUAGE

Under the TSM system a programmer may ignore the fact that the machine is time-shared. To a professional programmer at a terminal the machine behaves like a normal 7090, and the terminal has the same kind of capability as the computer console.

Writing a program to run under TSM is equivalent to writing a FAP program for a normal installation. The instruction set and the FAP assembly features are unchanged; the computing capabilities of the 7090 are augmented, but not altered.

Figure 3 will make sense only to readers familiar with the IBM 7090 instruction set. It is a list of all instructions trapped by the memory protection feature. All instructions are trapped which could do damage to the system. In particular these instructions are:

1. those which control I/O units, and
2. those which control the 7090 interrupt (or "trapping") system.

The left-hand set are either simulated or "translated" by the supervisor, and in effect are executed in normal fashion. "Translation" means that any reference to an I/O device is translated from what the user thinks he is using to what the system has actually assigned for him. A reference to tape drive A4, for example, may result in an operation on tape drive B2.

The right-hand set are instructions which the system considers to be illegal, and which will stop the program. When one of these instructions is encountered, the user's program will leave the time-sharing cycle, to be preserved in a memory image area on the disk file. The user at the terminal will receive a diagnostic message telling him where the program stopped and why. Readers of this article who are familiar with the IBM 7090 will note that the only real restriction on their programming is the fact that they cannot use the LCH (Load Channel) instruction.

PSEUDO-INSTRUCTIONS

By the addition of "pseudo-instructions" the instruction set of the 7090 has been augmented to provide for conversational programming and communication with the supervisor. These pseudo-instructions are disguised forms of the 7090 "Store and Trap" instruction. Whenever it is executed this instruction generates a trap (i.e., interrupt) to the supervisor, which proceeds to decode it and execute the function

EXECUTE			STOP	
ENB	SCH	STC	LRI	ECTM
ETM	SCD	TCO	LPI	ESTM
LTM	RDS	TCN	PSL	LSNM
EFTM	WRS	BIT	LCH	ESNT
ENK	BSR	ETT	HTR, HPR	
LFTM	BSF	RIC	DVH, FDH	
IOT	REW	RDC		
RCT	RUN	TRC		XEC → XEC → XEC
ICT	RCH			
TEF	RSC			

Figure 3. 7090 Instructions Trapped by Memory Project.

called for. Control returns after execution to the instruction following the pseudo-instruction. A programmer using these added instructions need not be aware of the fact that they are executed by the TSM Supervisor rather than the 7090 CPU.

### CONVERSATIONAL MODE

Figure 4 illustrates the conversational capability which has been added to the instruction set. A program can converse with its terminal by means of two pseudo-instructions called Receive (RCV) and Transmit (TMT). It can also converse with any other program, or set of programs, using two other instructions called TALK and LISN.

The RCV and TMT instructions allow a user to write a program which includes him in the feedback loop. This is a man-machine capability which can be exploited in many ways, most of which are as yet untried. The TSM system's own service routines are constantly conversing with the terminal operator, either asking for instructions or printing out errors.

The TALK and LISN instructions allow a group of programs, and consequently a group of terminal operators, to engage in any sort of multiple-person computing activity, plus obviating some otherwise sticky time-sharing problems. For example:

1. The computer can be used as a message switching center. Data entered at one ter-

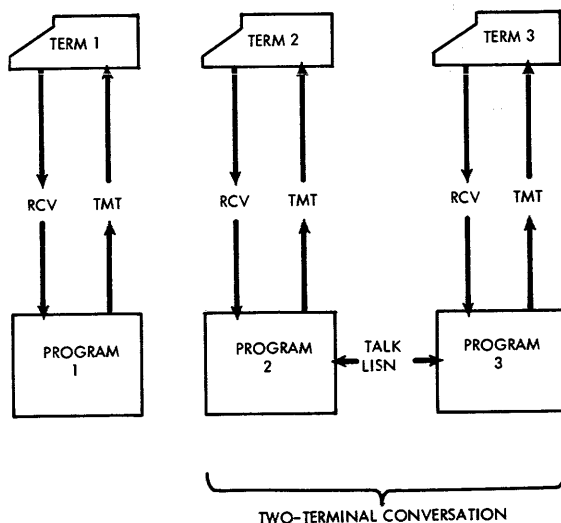


Figure 4. Conversational Mode.

terminal can be sent out to any other terminal, or all terminals.

2. A user at any terminal has direct access to the terminal in the machine room.
3. Unattended "write-only" terminals ( a remote printer for example) can be activated and controlled from inside the system, and automatically shut off when not in use.
4. Experiments in computer-based group education can be handled very flexibly. There can be a master program, which would be activated and controlled by an instructor, and a slave program which prompts and responds to a student. Multiple copies of the slave program, one for each student in the system, would execute independently, each at its own pace and communicating with the instructor's program as required.
5. Multiple-person gaming situations are a similar application.

These pseudo-instructions behave like normal 7090 I/O instructions. They are requests for the supervisor to send a block of data to a terminal or to another program. With respect to terminals the user is not concerned with the fact that various buffers are filled and emptied periodically while a TMT command is being executed. This is an activity executed by the supervisor.

### INTER-PROGRAM COMMUNICATION

For each subchannel on the system there are, as mentioned above, a set of terminal message buffers. These buffers are normally used as storage for incoming or outgoing messages during transmission. They are also used as the medium of information exchange between independent time-shared programs. A TALK instruction causes the transfer of a block of data from the working space of the calling program to the message buffers of the called program. The LISN instruction allows the called program to read the transferred data into its own working space. It is not necessary for the two programs exchanging data to be in memory at the same time.

Connections between programs are recorded in a "crossbar matrix" within the supervisor

(Figure 5). Each subchannel on the system is represented by a specific row and column in this matrix. A bit in the row of a program is a signal that some other program is trying to establish a "connection." The bit position identifies the program (i.e., subchannel). A program may interrogate its matrix row by issuing a CHEK pseudo-instruction. This is the equivalent of asking "Has anyone called?"

A data transfer between two programs proceeds as follows:

1. The calling program issues a CONN instruction, which sets a bit in the called program row. It then CHEK's and waits for a response.
2. The called program CHEK's for an incoming call. When a bit appears in its row it responds by issuing a CONN in return.
3. The calling program issues a TALK, which results in the data transfer and resets the CONN bit in the called program now.
4. The called program issues a LISN, which reads out its buffers and resets the CONN bit in the calling program row.

Any number of independently time-sharing programs may exchange data in this manner. The net effect is that a set of programs—and hence a set of terminal operators—can interact with each other on the solution of a computing problem.

In summary the TSM system does not restrict use of the 7090 for conventional computing purposes, nor is the style of programming necessarily different. A wide range of conversational capabilities has been added to the computer which enable experimentation both with new applications, and with new methods of handling traditional applications.

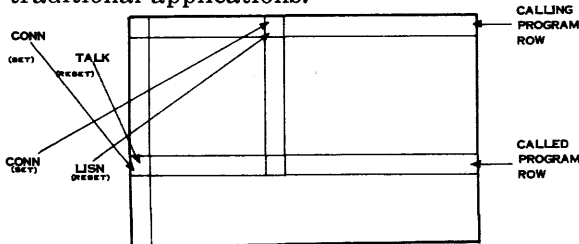


Figure 5. Crossbar Matrix.

## ADDITIONAL LANGUAGE CAPABILITY

All of the above comments have been made with reference to the basic FAP programming language, which assumes a great deal of computer knowledge on the part of the user. This is by no means the only language capability of the system. The TSM system also contains a FORTRAN compiler, a version of the General Purpose Systems Simulator, and an interpretive subsystem called PAT. All of these source languages are acceptable as input from the terminals.

## THE PAT SUBSYSTEM

The Personalized Array Translator (PAT) Programming System is a self-contained, interpretive subsystem which operates under the control of TSM. A terminal operator who is using the PAT subsystem is, in effect, using his terminal in the PAT mode. All terminal activity is generated or processed by the PAT System, even though it must pass through TSM in transit.

The PAT language is a subset of the Iverson language<sup>(6)</sup>. It permits operations upon entire arrays of numbers—matrices or vectors—as well as characters or bits. It includes relational statements for comparison of arrays according to various criteria, and statements which will generate arrays. This latter feature is valuable for obtaining argument arrays and test data.

Included in the PAT subsystem are some 11 console commands which allow a user to construct, debug and execute a program in source language alone.

## USE AND MAINTENANCE OF THE IBM 1301 DISK FILE

The 1301 Disk File is the primary medium of storage in the TSM System. It contains all of the system service routines, all user programs, and most of the data files which user programs operate upon.

Each module of the disk file has 40 disks, with 250 concentric tracks per disk. A vertical section of 40 tracks is a "cylinder." Each cylinder has a special non-data track called the format track, which controls the pattern of in-

formation. A special "Write Format" order allows the user to specify the number and size of records to be filed on all tracks of a cylinder. A "Write Track with Addressess" order allows records to be given unique identification for random record processing purposes. The 1301 file is a very flexible device, and the TSM system is designed to let a user exploit this flexibility.

The system considers the basic unit of file space to be the cylinder: 40 tracks, or approximately 18,000 words. A program operating under TSM may have up to 150 cylinders assigned for its use, which can be as many as 30 separate data files.

The supervisor keeps track of each cylinder with respect to occupancy and usage. A cylinder is "occupied" if it contains a data file or program (or a section thereof). A cylinder which is occupied may not be assigned by the system for scratch purposes, and it must be saved by the system on each shutdown operation and restored at startup time. A cylinder is "in use" if it has been assigned to a program for a job run. A cylinder in use may be either an occupied cylinder (data file) or a vacant scratch cylinder.

User programs cannot request the use of system facilities. All requirements of a program—core space, tape drives, data files and scratch cylinder space—must be known to the system before a program can start execution. Each program in the file contains a set of parameters which are examined by service routines. The service routines request the supervisor to assign such facilities as are needed by the program. If these facilities are available, the supervisor notes them as being "in use," and the user program can then manipulate them any way it sees fit.

The supervisor only keeps track of the status of a facility, not of its content. Information about the contents of the 500 file cylinders is recorded in a system table which itself is on the file. This table has a three-word entry for each cylinder. The entry contains the six-character name of the file, information about the type of file (program, data, read-only, etc.), and the job code and serial number of the user. The

table is created and maintained by the service routines.

A program or a data file may be entered into the system under three levels of protection. It can be for use by the user only, by all users with the same job code, or by the public in general. It can also be file-protected, which will prevent either a program or the system from altering it.

Special format tracks for data cylinders may be written by the system at the user's request, or by the user's program at execution time. Either data files or programs may be many cylinders in length. Even though the TSM system may scatter successive cylinders of such a file throughout the 1301, the user can manipulate them as if they were in contiguous space.

A single service routine called LOAD creates all initial files. This is a command which is available only to operators of the machine room terminal. The load program operates from a batch tape. From each file on the tape it creates a file in the disk. It operates time-shared, which means that the system can maintain its files while it is operating.

The load program will accept almost any form of input; source language programs (or subroutines), object code in either relocatable or absolute form, data files in either standard or special formats. If no format specifications are given, a standard one-record-per-track format is generated, otherwise the format track will be written as the user desires.

A program being processed by the load program must have various parameter cards supplied with it. Among other things these parameter cards specify the facility requirements of the program (tape drives, files to be referenced, etc.), and its size in terms of core space. This information is attached to the program as it is written in the file. Once the program has been loaded these specifications may be altered at will from a terminal, but they are never accessible by the program itself.

In writing a program to use cylinder-stored files of data, a user refers to relative cylinder numbers in much the same ways as he now uses relative tape drive numbers. The load program must be told, for example, that as far as

the user is concerned, file N is on cylinders 0000-0002. This information becomes part of the program parameters. The load program may actually store file N on a set of cylinders such as 0052, 0267 and 0138. Translation of file orders from the relative cylinder numbers supplied by the programmer to the absolute locations assigned by the load program is done at execution time by the TSM supervisor.

Programs are stored in both source language and object form. From the terminal a user can manipulate either. The same is true of subroutines. A user may have his own collection of subroutines stored away, as well as having access to the system library.

The terminal operator can also manipulate the facility assignment parameters of a program. Among other things this means that he can select a program for execution and choose, from among alternatives, a particular data file for it to operate on.

#### OPERATING A TSM TERMINAL

Operation of a TSM Terminal is a process of issuing a series of commands to the system. Each command is a short message of the form "OP;OPERAND", and is executed by a unique service routine.

The basic set of commands which the system will recognize is designed for a user who is operating a machine language program. In this mode the terminal is a remote 7090 operating console. Figure 6 illustrates the functions of the commands in this set. The heart of the structure is the Terminal Core Image area. Each subchannel in the system has such an area on the 1301 file permanently assigned to it. It is two cylinders in size, or approximately 36,000 words. The core image is core memory as far as the terminal operator is concerned. When his program is actually executing in the time-shared cycle, it may at any moment be either on the drum or in actual core memory. Whenever it stops executing, for any reason, it returns to the image area. The image area will always contain the user's program as it looked when it stopped running, plus a complete set of status information with respect to the 7090 itself.

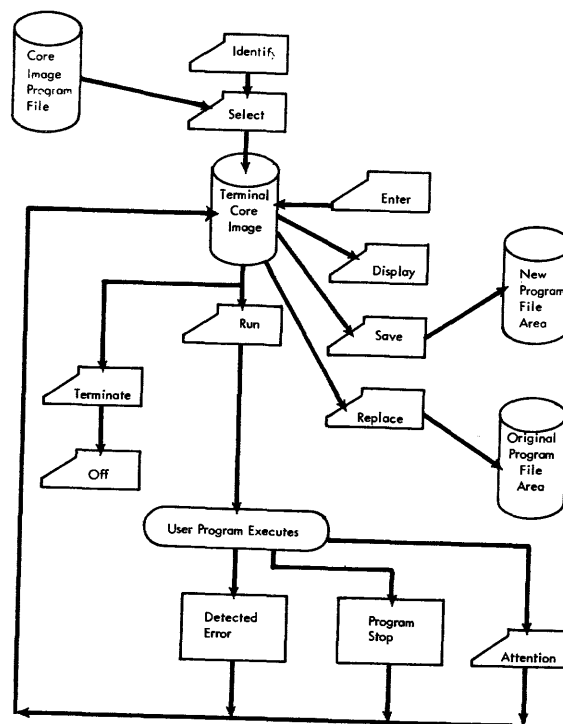


Figure 6. Service Routines for Program Execution and Control.

A typical execution sequence in this mode of operation is:

1. ID; Operator Number, Project Number.  
This signs the user on the system.
2. SEL; Program Name.  
This identifies the program to be executed. The Select service routine retrieves the program from the 1301 file and writes it into the core image area. This is the equivalent of loading the program into memory.
3. RUN  
Since the program has been identified and isolated for action, there is no need to keep repeating its name as an operand. The RUN service routine performs a non-trivial function of determining the requirements of the program in terms of core space, data files to be operated upon, tape reels to be mounted, special instructions to be sent to the machine room, etc. It then informs the TSM Supervisor that the program is ready, and the user's program enters the time-sharing cycle.
4. User Program Execution  
The user is on his own. His program is executing and in control of the terminal.

It will continue to time-share until (a) the system detects an error, (b) a normal program stop is encountered, (c) he hits the ATTENTION button on the terminal. In all three cases the machine status will be preserved and the program will be returned to the core image area. In cases (a) and (b) the operator will receive a message at the terminal which might be:

```
LOCATION 0142
PROGRAM STOP
```

The operator now has a choice of actions. A variety of Display commands are available which will type out machine status conditions or sections of the program (or all of it), and an equivalent set of Enter commands which can be used to alter either the machine status or the program. These commands operate upon the contents of the image area, which is the equivalent of patching the program. There is a SAVE command which will preserve the image as a new file for later reference, and a REPLACE command which will dump the image in the original file area. A RUN XXXXX command can be given, which will restart the program from any specified instruction.

There is also a TRACE command which starts controlled execution of the user's program. This function has several options such as "stop at location X," "stop on reference to X," "print all transfers," etc.

In summary, the operator can manipulate his program with a great deal of freedom; more freedom, in fact, than the typical computer console allows.

This is the basic set of commands for remote IBM 7090 operation. There are others. "PROCESS" is the command which triggers the compiler-assembler-loader functions of the system (Figure 7). "UPDATE" allows a user to make changes to a source language file (or data file) in preparation for recompilation or reassembly. The command set is easily expanded as need arises, perhaps too easily. It has, in fact, a Parkinsonian tendency to grow.

One other command should be mentioned. "CALL; subsystem" is a request for the system to switch the terminal over to a specified alternate mode of operation. GPSS is an alternate

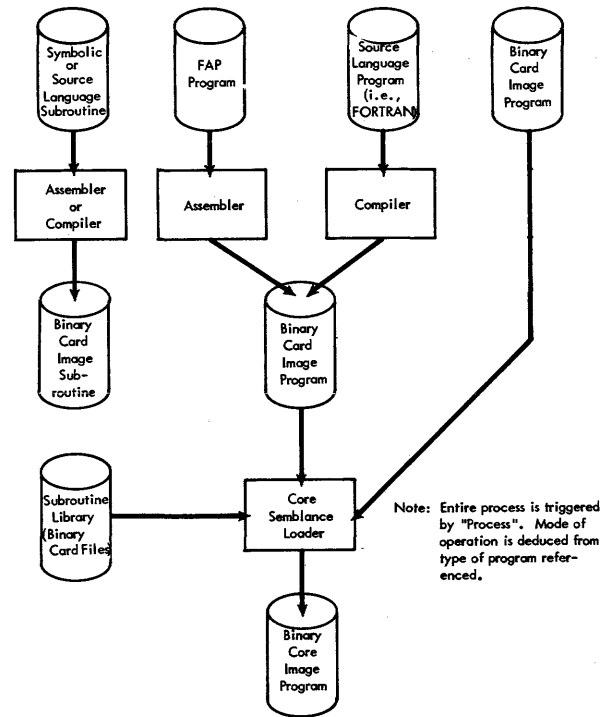


Figure 7. Program Assembly or Compilation.

mode. It is a closed, interpretive subset of TSM with its own interdependent set of commands and operating procedures. A "CALL; GPSS" command will in effect substitute a new set of commands for the basic set. The terminal will thereafter operate under control of the GPSS subsystem. If a command is issued which GPSS does not recognize, the terminal will automatically revert to the basic TSM mode.

## BACKGROUND OPERATION

A terminal operator can request that a program be executed in the background mode. The background is simply a list, maintained by the supervisor, of jobs which are to be run as time permits. There is no difference between the handling of foreground (terminal controlled) and background jobs, other than the fact that there is no terminal associated with a background time slice.

Some background job is always being time-shared with the terminals. If no terminals are active the current background job is run continuously, and the TSM system behaves very much like a normal batch-processing system. If one terminal is active it is time-shared with the

current background program, with each program using the machine 50% of the time.

**RESPONSE TIME**

The response of the system to a command from a terminal is quite variable. It depends upon the number of programs being time-shared at the time the command was received, the sizes of those programs, and the size of the service routine which must execute the command. When a command is received by the system the service routine must have space assigned for it in memory and then it must be retrieved from the disk file. Retrieval time varies from .2 seconds to 1.8 seconds depending upon the size of the service routine. Space assignment time is negligible if the load is light and memory is free. In the worst case, where 24 active users are each running 24,000 word programs, the response time can be as much as 50 seconds. Assuming ten active terminals, each controlling a 6000-word program, the average response time is approximately 3 seconds.

**THE SCHEDULING ALGORITHM**

The TSM Supervisor has been designed to use the computing equipment as efficiently as possible. It "space shares" core memory as well as time-sharing the CPU, and takes full advantage of the fact that the data channels can operate in parallel with the computer.

The Supervisor (Figure 8) is a set of "trap processors" (one for each type of trap which the hardware system can generate) and "queue processors" (one for each shared facility) regulated by a Sequence Control routine. A trap processor recognizes the fact that an I/O facility has completed an operation and is now available for use. A queue processor initiates an operation on an idle facility.

A section of the Sequence Control Table, which contains the set of queues in the system, is pictured in Figure 9. Three system facilities are represented: the drum channel, drum space and core space. Each facility is represented by three words of data:

1. A status word: if the sign is plus the facility is available; if minus it is busy. The address portion of the word indicates

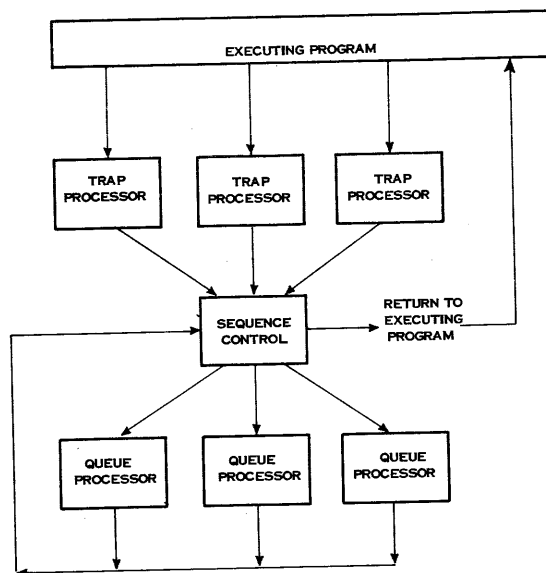


Figure 8. The TSM System Supervisor Design.

the location of the associated queue processor routine.

2. The queue: each bit position of this word represents a specific terminal. In the example shown the programs being time-shared for terminals 3 and 18 are in the core space queue.
3. A sequencer: this is a single bit which is cycled to the right to select the next program (i.e., terminal) for use of the facility. The sequencer always sits at the position of the last program chosen. In the

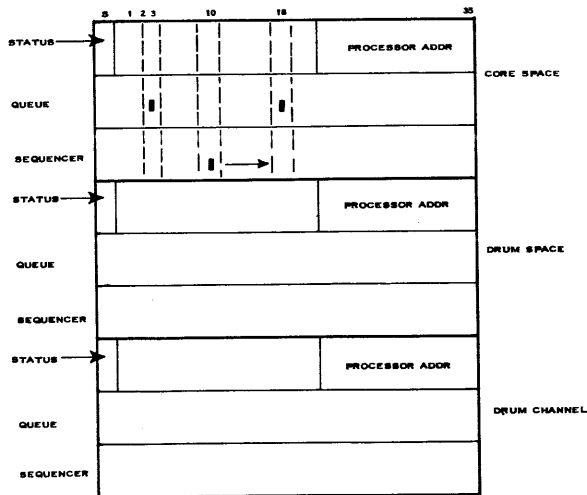


Figure 9. Section of the Sequence Control Table.



core space block the sequencer has stopped at bit 10. The program for terminal 10 was the most recent one to have been assigned core space. Terminal 18 will be the next one.

All trap and queue processors exit to the Sequence Control routine upon completion. Sequence Control scans its table, looking for the combination of an idle facility and a non-zero queue. If it finds one it chooses the next program for execution on that facility by moving the sequencing bit to the right, and then exits to the appropriate queue processor.

The queue processor will initiate operation of its facility, and in so doing may move the selected program to another queue. It will return to Sequence Control, which will scan the table again. This succession of events will continue until Sequence Control is able to make a complete pass through its table without finding an idle facility which can be put to work. Control then returns to the interrupted executing program.

This design is a descendent of the control program system for Project Mercury<sup>(7)</sup>. It has certain advantages:

1. The supervisor can be and has been written as a set of modules (trap and queue processors) relatively independent of one another.
2. It operates "asynchronously," keeping all facilities of the system continuously busy if there is any work which can be done.
3. It guarantees that all terminals get equal treatment.
4. It is inexpensive. The sequence control table together with the sequence control program total approximately 100 words of space.

## SUMMARY

The TSM System is experimental. It was built as a base for exploration of various terminal designs, terminal-oriented applications, human factors considerations, and an entirely new mode of computing. It has not yet been in operation long enough to allow any firm conclusions to be stated. Our experience in building

it has taught us a great deal about general-purpose remote computing with present-day equipment, but we are not yet ready to extrapolate this experience into future hardware design.

Our experience in adding language capability to the basic system has been enlightening. Almost any existing 7090 compiler or interpreter can be made a feature of the TSM system. These existing programs can be added to the system as service routines by cutting down their size, so that they operate in 24,000 words of core memory, and revising them to operate on disk file data instead of tape-stored data. The most difficult part of this process is altering a compiler so that it is disk file oriented. It must reside in the file, get its source language from the file, and write its object program in the file. In practice even this has not turned out to be particularly difficult. As soon as the modifications have been made, the revised compiler can be added to the file of programs and debugged during a normal time-sharing period.

The result of this process, however, is not necessarily a good compiler for time-shared operation. A time-shared computer is not an ordinary computer, even though it can be treated like one. It is time-shared, which means that the start-to-termination time of a specific job must be multiplied by the average number of users on the system; it has conversational capability which should be used; and there is a human being at the terminal who is expecting some sort of response. These factors all work against the design of existing compilers (and applications!), which are usually massive, long-running, and unsociable. The right way to add a language to TSM (or any other basic time-sharing system) is to create a compiler which fits the time-sharing situation. This right way is not yet completely defined. Some of the most original work in this area to date has been in the development of remote computing languages based on interpreters rather than compilers, the IBM experimental remote FORTRAN system<sup>(8)</sup> being an excellent example, our own PAT subsystem being another.

The TSM System is more than a 7090 with extra gear and a special software system. It is a new type of computer. It is reasonably compatible with existing computing capability—

which is to say that a user can ignore the time-sharing factor and operate the system as a 7090 if he sees fit—but it is in fact time-shared and conversational. Our experience in adapting existing software to this environment indicates that while it is feasible to do so, the results are not entirely satisfactory. The compilers and assemblers run slowly, use the system inefficiently, and behave (as of course they were designed to behave) as if there were no people connected with the process.

The implications are that time-sharing, if it proves to be a popular, successful mode of computing, will require a substantial revision of software technology, and an equivalent revision of application methods. The TSM System has been constructed and is being operated as a vehicle for studying these implications.

#### REFERENCES

1. Reference Manual, IBM 709/7090 Programming Systems: FORTRAN Assembly Program (FAP), IBM Form Number C28-6235.
2. Reference Manual, General Purpose Systems Simulator II, IBM Form Number B20-6346.
3. H. HELLERMAN, "Experimental Personalized Array Translator System," *Communications of the ACM*, July, 1964, pp. 433-438.
4. F. J. CORBATO, et al, "The Compatible Time-Sharing System—A Programmer's Guide," The MIT Press, May, 1963.
5. E. G. COFFMAN, JR., J. I. Schwartz, C. Weissman, "A General-Purpose Time-Sharing System," *Proceedings of the Spring Joint Computer Conference*, 1964, pp. 397-411.
6. K. IVERSON, "A Programming Language," John Wiley and Sons, Inc., 1962.
7. M. J. BUIST and G. M. WEINBERG, "Real-Time Multiprogramming in Project Mercury," *Ballistic Missile and Space Technology*, Vol. 1, Academic Press, Inc., 1960.
8. T. M. DUNN and J. H. MORRISSEY, "Remote Computing—An Experimental System," *Proceedings of the Spring Joint Computer Conference*, 1964, pp. 413-423.

# JOSS: A DESIGNER'S VIEW OF AN EXPERIMENTAL ON-LINE COMPUTING SYSTEM\*

*J. C. Shaw*

*The RAND Corporation, Santa Monica, California*

## INTRODUCTION

The JOHNNIAC Open-Shop System (JOSS) is an experimental, on-line, time-shared computing system<sup>1</sup> which has been in daily use by staff members of The RAND Corporation since January 1964.\* It was designed to give the individual scientist or engineer an easy, direct way of solving his small numerical problems without a large investment in learning to use an operating system, a compiler, and debugging tools, or in explaining his problems to a professional computer programmer and in checking the latter's results. The ease and directness of JOSS is attributable to an interpretive routine in the JOHNNIAC computer which responds quickly to instructions expressed in a simple language and transmitted over telephone lines from convenient remote electric-typewriter consoles. An evaluation of the system has shown that in spite of severe constraints on speed and size of program, and the use of an aging machine of the vacuum-tube era, JOSS provides a valuable service for computational needs which cannot be satisfied by conventional, closed-shop practice.

This paper concentrates on the numerous, small, hardware and software design decisions which have influenced the acceptance of the system by its intended users. Several figures,

produced on-line, are included, providing readable examples of features of the JOSS language.

## *Background*

From the earliest days of construction of the JOHNNIAC computer, a Princeton-class machine built at The RAND Corporation in 1950-53, it has been the author's dream to have an economical, personal, remote communication station for on-line control and programming of a computer. With so much to be learned about programming and operating large, general-purpose computers, it isn't surprising that the additional investment in communications equipment, remote stations, and corresponding software was postponed.

In its early days, JOHNNIAC served well as a production machine. Then, because it has only a 4096-word core memory, a slow 12,288-word drum, slow copy-logic for card I/O and printing, no tapes, and a *very* austere order code, production computing was gradually shifted to more modern IBM equipment. Yet, the very accessibility to this unsaturated second machine made JOHNNIAC attractive as the basis for simplified programming systems for small, open-shop problems and for experimental work in heuristic programming,

---

\* Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views or opinions of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors.

\* An austere version of the system saw limited use during most of 1963.

new software systems, and hardware for better interaction with a computer. In November 1960, after years of discussion of personal remote consoles with T. O. Ellis, I proposed to the management of RAND's Computer Sciences Department that JOHNNIAC be committed full time to providing a modest computing service to the open-shop via remote typewriters.

The purpose of the JOSS experiment was not to make JOHNNIAC machine language available, but rather to provide a service through a new, machine-independent language which had to be designed specifically for the purpose. It was to be an experiment with the goal of demonstrating the value of on-line access to a computer via an appropriate language, and was intended to contribute to a project with the long-range goal of a sophisticated information processor. T. O. Ellis, I. Nehama, A. Newell, and K. W. Uncapher were the other participants in that project.

In 1961-62, Ellis and M. R. Davis designed and directed the construction of the required multiple typewriter communication system adjunct to JOHNNIAC. The hardware was ready well in advance of the first version of the system program, and only a few select users were subjected to this very limited system. Their feedback, including encouraging remarks on the usefulness of JOSS, helped shape the full version.\*

#### *Comparison*

Other on-line, time-shared computing systems have become operational in recent years.<sup>2-6</sup> All are pioneering efforts. By comparison, JOSS is special-purpose, even though it encompasses a wider class of problems than one might guess at first reading. Most of the others provide the user with access to machine language. F. J. Corbató has aptly described them as *open* systems and JOSS as a *closed* system. In the open systems, an executive routine is prepared to help the user at the machine-language level or to pass control to one of several subsystems providing adapta-

tions of pre-existing programming systems. JOSS, however, was designed with on-line interaction in mind, and resources were devoted to making it smooth and easy to use. The future lies with the open systems, but it remains to be seen whether the open-system executive will absorb JOSS-like systems simply as additional subsystems, or whether JOSS-like systems will absorb the executive function and thus serve as the user's computing aide and single contact with the computer.

#### HARDWARE COMPONENTS OF JOSS

Physically, JOSS consists of the JOHNNIAC computer, ten remote consoles, and a multiple typewriter communication system to mediate between JOHNNIAC and the consoles.

#### *Johnniac*

RAND, as did several universities and research institutions in the early 1950s, constructed a computer (called "JOHNNIAC" for John von Neumann) more or less on the pattern of the machine of the Institute for Advanced Study at Princeton. JOHNNIAC was upgraded in 1954 with the replacement of the original 256-word Selectron memory with a 4096-word magnetic core memory. The word length is 40 bits. Because JOHNNIAC was ill-equipped to handle the message traffic required in JOSS service, a special-purpose buffering system was built to process characters within messages and to monitor the remote stations. The alternative of modifying the main frame to handle the message traffic directly would have required a major rework of the JOHNNIAC control and would still have yielded degraded performance in JOSS service. Thus, JOHNNIAC remains a very primitive machine with no indexing, no indirect addressing, no floating point, no error checking, no memory protect, no interrupts, no channels, no compare, no zero test, a miserable format of two single-address instructions per word, and a 50- $\mu$ s add time.

The JOSS system program runs about 6000 words, the low-frequency portions residing on drum and overlaying each other in core when called in for execution. A large part of the JOSS system program resides permanently in core. It was a considerable challenge to com-

\* We wanted to do a controlled evaluation of the system at the time of the introduction of the full version of JOSS, but the new users taught others so quickly that we had to resort to after-the-fact questionnaires!

press it sufficiently to leave room for the processing of a user's block in core. More than once I regretted the lack of an adequate subroutine linkage operation; it would have saved much space in this deeply hierarchical program.

The 12,288-word JOHNNIAC drum is divided into three sections, accessed by moving heads at a rate not quite so fast as a modern disk unit unless the heads are luckily in the correct position. Average swap time (i.e., the time to write one user's block of information out onto drum and read a second user's block into core for processing) is, therefore, quite slow at about half a second.

### *Communication System*

The multiple typewriter communication system provides sixteen line-buffers, controls the states of all ten remote consoles, and registers signals from them. The limit is 81 consoles—well beyond our needs and our budget. The JOSS system program in JOHNNIAC commands block transfers between core and the line buffers. It also commands the communication system to enable or disable a console, request or relinquish control of a console, clear a line buffer, assign a line buffer to a console, or transmit a line buffer to a console. It also commands the communication system to report any signals from consoles indicating a carriage return, a page ejection, or the depression of one of the console control keys.

### *Remote Console (Fig. 1)*

Lights and switches in a small box augment the IBM model 868 typewriter to indicate the status and to control the functions of the local communication terminal electronics. The switches are: a POWER switch; an ON switch to connect the terminal to JOSS; an OFF switch to disconnect; a READY switch to reactivate the typewriter after inserting a fresh supply of paper; and IN switch to request control of the typewriter for input; and an OUT switch to relinquish control for output. Indicators are provided as follows: a POWER light; an ENABLE light showing that JOSS service is available; a READY light showing that output is acceptable at the typewriter; a red light to show that JOSS controls the typewriter; a green light to show that the user controls it;

an IN REQUEST light to show that the user has depressed the IN button for control but JOSS hasn't yet responded; and an OUT REQUEST light to inform the user that JOSS has an administrative message for him (such as "Shutting down at 2330.").

The READY light goes out if the paper supply is exhausted or if the paper jumps the sprockets. The user may also switch the READY light off any time he wants to hold up output momentarily. To continue with the output, the READY light is turned back on; no information is lost. The philosophy is one of exclusive control of the typewriter. When JOSS has control, the red light is on, the keyboard is locked, and the typewriter ribbon color is black. As JOSS turns control of the typewriter back to the user, the light changes to green, the keyboard unlocks, the ribbon color changes to green, and a soft gong rings. These visual, tactile, and audible signals leave no doubt as to who controls the station.

If a remote typewriter console is to be a personal instrument, it must also serve as a simple typewriter. This consideration dic-

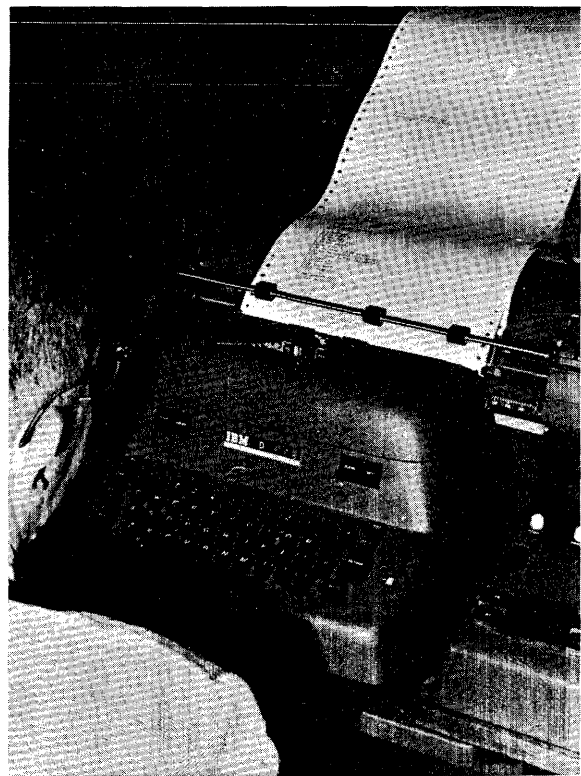


Figure 1. JOSS Console and Station Local Control Box.

tated that the console allow for off-line use and that the character set include all the normal punctuation of a typewriter. The sprockets and paging mechanisms restrict the stations from being entirely satisfactory as personal typewriters, because of the problem of changing paper and the excessive noise. However, the hard copy produced is excellent—quite acceptable for reports without further transcription and chance for error.

#### Keyboard (Fig. 2)

The choice of character set and key positions for any on-line keyboard input device isn't to be taken lightly, especially if one hopes to encourage senior technical people to use the keyboard in the direct solution of their problems. It is customary for these people to pay others to drive teletypewriters, keypunches, and even typewriters. For the JOSS remote typewriters, we left the comma, period, semicolon, colon, slash, question mark, quotes, space sign, dollar sign, parentheses, and hyphen in their customary positions. The less essential characters of standard sets were sacrificed in order to make room for all six numerical relation symbols.

To linearize numerical expressions requires an explicit sign for exponentiation, for which we chose a five-pointed, upward-pointing, slightly elevated asterisk in upper case (all of which contribute proper associations for exponentiation). The plus, minus (hyphen), centered dot (for multiplication), slash (for division), and equals sign are all in lower case. Left and right brackets were included, in place of the upper-case comma and period, in order

to improve readability of linearized expressions (otherwise, such expressions tend to become cluttered with parentheses). The absolute value bar also contributes somewhat to readability. Parentheses and brackets are interchangeable in pairs for all grouping functions: subexpressions, arguments for functions, indices, and interval size in iteration expressions.

The punctuation and capitalization rules for JOSS are quite conventional, but three symbols are used in very special ways. The space sign (#) is used as a strikeover character, causing a character already in the input line buffer to be replaced by a space. This is needed since, if the typed line is to reflect the contents of the input buffer, the space bar and backspace key must never *enter* any character into the buffer (although they do control position). The asterisk (\*) at either end of an instruction input line leads JOSS to ignore the line and thus provides a device for annotating one's work and for cancelling lines. (For most errors, however, the process of simply backspacing and striking over is adequate.) The dollar sign (\$) may be used in any expression and carries a numerical value equal to the line number (from 1 to 54) of the typewriter's current position on the page. This value is updated by JOSS so the user can easily control format on the page. If no format is specified, JOSS supplies an automatic one-inch margin at the top and bottom of each page.

The tab may be used to speed output typing by skipping instead of spacing over blank positions. The typing speed of ten characters per second, less shifting time, has not been a source of dissatisfaction. A key interlock, intended to insure that only one key at a time was depressed, was abandoned because of the frustrating effect on the user. The action of the keys without the interlock is admirable, but it is possible to hit two keys at once, superimposing their character codes for transmission. This risk is more acceptable than the interlock but it means that JOSS must be prepared to receive any 7-bit character code—not just the legal ones. The keyboard lock (not to be confused with the now-abandoned key interlock) is incomplete in that it locks only the typing keys, and even then it can be over-

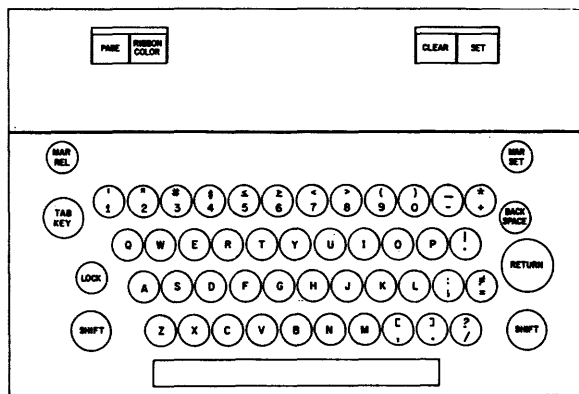


Figure 2. JOSS Typewriter Keyboard.

riden. This deficiency has not been a problem however since, at most, the user can spoil only his own output by trying to use the typewriter out of turn.

## SOFTWARE

JOSS services the requests of users at the remote consoles in such a way that the users' activities are logically independent of one another. Up to eight of the ten stations may be served concurrently by time-sharing techniques. In addition to administering input/output and swaps of user blocks, JOSS interprets and executes both direct and indirect (i.e., stored-program) instructions couched in a readable and easily learned language.

### *Time Sharing*

The basic JOHNNIAC computer provides no parallel processing. The multiple typewriter communication system does provide for parallel activity at many consoles by high-speed line-scanning and time-shared use of the logic circuits. JOSS takes advantage of this independent parallel processing in the communication system by time sharing—i.e., by switching its attention rapidly from one user to another to give adequate service to all active users. Each active user is represented by a block of information which resides on the drum, except when JOSS is actually processing it in core.

First priority for JOSS' attention goes to the servicing of signals from the consoles: carriage return, page, on, off, in, out, and end-of-transmission. JOSS looks for these signals in the communication system when idling, and between interpretive steps when executing a user's program. An end-of-transmission signal requires only that JOSS record that the line buffer is available, and direct the transmission of the next line of output to the same station if one is ready. JOSS then continues with its previous activity. A carriage return, however, like several other signals, requires that JOSS break off its current activity, move the current user's block out to drum, move the signaling user's block into core, and, finally, interpret and act on the line of input just released by the carriage return.

Second priority is given to users who have given JOSS output-limited tasks which have

been set aside until the typewriters have nearly caught up. Third priority is given to users with unfinished tasks, on which JOSS works for two seconds apiece in round-robin fashion. A user's priority changes dynamically according to this discipline, which successfully exploits the parallel processing of the communication system. Under a typical load, JOSS responds to simple requests in a fraction of a second and rarely in as long as three seconds. Users who are skilled in typing can maintain impressive rates of interaction with JOSS.

### *Language*

The reader will observe that throughout this section on software, the term "JOSS" is used to refer to that single active agent at the computer end of the telephone line connecting the user's remote console. It is convenient to consider JOSS to be a "computing aide" interacting with the user by means of a simple language. The reader should now read the examples (Figs. 3a-3i) before continuing in this section. The examples fall short of being an adequate instruction manual for the system, but they suggest the readability of the language, the high degree of interaction, and the power of expression.

A striking feature of the system is that the user commands JOSS directly in the same language that he uses to define procedures for JOSS to carry out indirectly. A numeric label as a prefix to a step is an implied command to JOSS to store the step in sequence according to the numeric value of the label. JOSS differs from other on-line systems by requiring the user to supply his own step numbers on all steps of his stored program. This seems appropriate, for the user always has the option of typing a direct command or an indirect step, without having to explicitly call for another *mode* to get the desired option. The numeric label determines whether an indirect step is an addition, an insertion, or a replacement for another step. The step numbers really do pay their way. Elsewhere, the language is very explicit. For example, it requires full words, in conjunction with numerical expressions, to denote steps, parts, or forms. This too contributes to readability. A step is limited to a single line, and a line is

<p>U: Type 2+2. J: 2+2 = 4</p> <p>U: Set x=3. Type x. J: x = 3</p> <p>U: Type x+2, x-2, 2-x, x/2, x*2. J: x+2 = 5 x-2 = 1 2-x = 6 x/2 = 1.5 x*2 = 9</p> <p>U: Type [( x-5 .3+4).2-15].3+10. J: [( x-5 .3+4).2-15].3+10 = 25 (3a)</p>	<p>U: Type sqrt(3), sqrt(4). J: sqrt(3) = 1.73205081 sqrt(4) = 2</p> <p>U: Type sqrt(-1). J: Error above: Negative argument for sqrt.</p> <p>U: Set e=2.71828183. Type log(1), log(2), log(e). J: log(1) = 0 log(2) = .69314718 log(e) = 1</p> <p>U: Type exp(0), exp(.5), exp(1). J: exp(0) = 1 exp(.5) = 1.64872127 exp(1) = 2.71828183</p> <p>U: Type sin(.5), cos(.5), sin(.5)*2+cos(.5)*2. J: sin(.5) = .479425539 cos(.5) = .877582562 sin(.5)*2+cos(.5)*2 = 1</p> <p>U: Type arg(1,0), arg(0,1), arg(-1,0), 4*arg(3,3). J: arg(1,0) = 0 arg(0,1) = 1.57079633 arg(-1,0) = 3.14159265 4*arg(3,3) = 3.14159265 (3b)</p>	<p>U: 2.15 Line if fp(c)=0. 2.2 Type a, b, c in form 1 if fp(c)=0. Type part 2. 2.1 Set c = sqrt(a*2 + b*2). 2.15 Line if fp(c)=0. 2.2 Type a, b, c in form 1 if fp(c)=0. Do part 1 for a=1(1)15.</p> <p>J: a = 4 b = 3 c = 5.000 a = 8 b = 6 c = 10.000 a = 12 b = 5 c = 13.000 a = 12 b = 9 c = 15.000 a = 15 b = 8 c = 17.000</p> <p>U: Delete part 2. Type part 2. J: Error above: No such part. U: Type all values.</p> <p>J: a = 15 b = 15 c = 21.2132034</p> <p>U: Delete all. (3c)</p>																																																		
<p>U: Set y = 123.456. Type y, ip(y), fp(y), dp(y), xp(y). J: y = 123.456 ip(y) = 123 fp(y) = .456 dp(y) = 1.23456 xp(y) = 2</p> <p>U: Type 1.23456*10*2. J: 1.23456*10*2 = 123.456</p> <p>U: Type sgn(-3.5), sgn(0), sgn(3.5). J: sgn(-3.5) = -1 sgn(0) = 0 sgn(3.5) = 1</p> <p>U: Type max(1,2,3), min(1,2,3). J: max(1,2,3) = 3 min(1,2,3) = 1 (3c)</p>	<p>U: 3.1 Type x, sqrt(x), log(x), exp(x), __. Do step 3.1 for x=1,2,3,100.</p> <p>J: x = 1 sqrt(x) = 1 log(x) = 0 exp(x) = 2.71828183</p> <p>x = 2 sqrt(x) = 1.41421356 log(x) = .69314718 exp(x) = 7.3890561</p> <p>x = 3 sqrt(x) = 1.73205081 log(x) = 1.09861229 exp(x) = 20.0855369</p> <p>x = 100 sqrt(x) = 10 log(x) = 4.60517019 exp(x) = 2.68811714*10*43 (3f)</p>	<p>U: 3.1 Type x, sqrt(x), log(x), exp(x) in form 3. Form 3: -----</p> <p>J: Do step 3.1 for x = 8.50(.01)8.54(.02)8.60, 9, 9.5.</p> <table border="1"> <tr><td>8.50</td><td>2.915</td><td>2.1401</td><td>4.915</td><td>03</td></tr> <tr><td>8.51</td><td>2.917</td><td>2.1412</td><td>4.964</td><td>03</td></tr> <tr><td>8.52</td><td>2.919</td><td>2.1424</td><td>5.014</td><td>03</td></tr> <tr><td>8.53</td><td>2.921</td><td>2.1436</td><td>5.064</td><td>03</td></tr> <tr><td>8.54</td><td>2.922</td><td>2.1448</td><td>5.115</td><td>03</td></tr> <tr><td>8.56</td><td>2.926</td><td>2.1471</td><td>5.219</td><td>03</td></tr> <tr><td>8.58</td><td>2.929</td><td>2.1494</td><td>5.324</td><td>03</td></tr> <tr><td>8.60</td><td>2.933</td><td>2.1518</td><td>5.432</td><td>03</td></tr> <tr><td>9.00</td><td>3.000</td><td>2.1972</td><td>8.103</td><td>03</td></tr> <tr><td>9.50</td><td>3.082</td><td>2.2513</td><td>1.336</td><td>04</td></tr> </table> <p>(3g)</p>	8.50	2.915	2.1401	4.915	03	8.51	2.917	2.1412	4.964	03	8.52	2.919	2.1424	5.014	03	8.53	2.921	2.1436	5.064	03	8.54	2.922	2.1448	5.115	03	8.56	2.926	2.1471	5.219	03	8.58	2.929	2.1494	5.324	03	8.60	2.933	2.1518	5.432	03	9.00	3.000	2.1972	8.103	03	9.50	3.082	2.2513	1.336	04
8.50	2.915	2.1401	4.915	03																																																
8.51	2.917	2.1412	4.964	03																																																
8.52	2.919	2.1424	5.014	03																																																
8.53	2.921	2.1436	5.064	03																																																
8.54	2.922	2.1448	5.115	03																																																
8.56	2.926	2.1471	5.219	03																																																
8.58	2.929	2.1494	5.324	03																																																
8.60	2.933	2.1518	5.432	03																																																
9.00	3.000	2.1972	8.103	03																																																
9.50	3.082	2.2513	1.336	04																																																
<p>U: Delete all.</p> <p>1.1 Do part 2 for b=1(1)a.</p> <p>2.1 Set c = sqrt(a*2 + b*2). 2.2 Type a, b, c in form 1.</p> <p>Form 1: a = ___ b = ___ c = ___</p> <p>J: Do part 1 for a=1(1)3. a = 1 b = 1 c = 1.414 a = 2 b = 1 c = 2.236 a = 2 b = 2 c = 2.828 a = 3 b = 1 c = 3.162 a = 3 b = 2 c = 3.606 a = 3 b = 3 c = 4.243</p> <p>U: Type all steps.</p> <p>J: 1.1 Do part 2 for b=1(1)a. 2.1 Set c = sqrt(a*2 + b*2). 2.2 Type a, b, c in form 1.</p> <p>U: Type all forms.</p> <p>J: Form 1: a = ___ b = ___ c = ___</p> <p>U: Type all.</p> <p>J: 1.1 Do part 2 for b=1(1)a. 2.1 Set c = sqrt(a*2 + b*2). 2.2 Type a, b, c in form 1.</p> <p>Form 1: a = ___ b = ___ c = ___</p> <p>a = 3 b = 3 c = 4.24264069 (3d)</p>	<p>U: 4.1 Demand b(1). 4.2 Set a = a*b(1).</p> <p>Set s=0.</p> <p>Do part 4 for i=1(1)4.</p> <p>J/U: b(1) = 543 b(2) = 237 b(3) = 411 b(4) = 733</p> <p>U: Type a. J: a = 1924</p> <p>U: 5.1 Set b(1) = b(1)/a. Do part 5 for i=1(1)4. Type b.</p> <p>J: b(1) = .282224532 b(2) = .123180873 b(3) = .213617464 b(4) = .380977131</p> <p>U: Set s=0. Do step 4.2 for i=1(1)4. Type s.</p> <p>J: s = 1</p> <p>U: Delete all. (3h)</p>																																																			

U - Denotes inputs of the JOSS user; normally typed in green.  
J - Denotes outputs from JOSS; normally typed in black.

Figure 3. Samples of JOSS Language and Interaction.



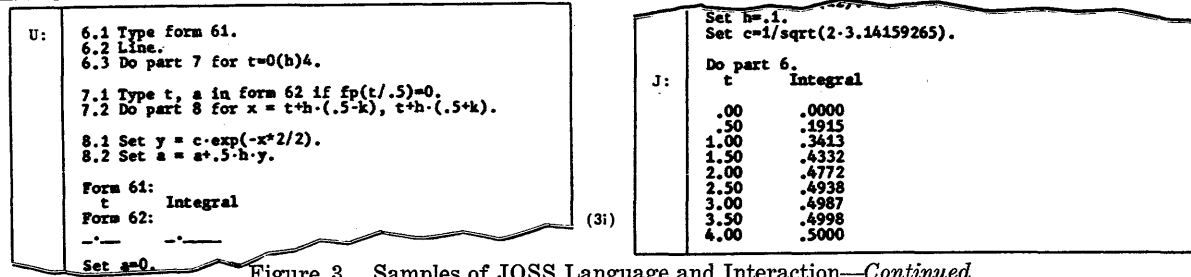


Figure 3. Samples of JOSS Language and Interaction—Continued

limited to a single step, neither being much of a constraint. As a result, a step number serves to identify not only the logical step but the stored string and typographical line as well. Arbitrarily complex expressions may be used everywhere, except as step label prefixes which must be explicit decimal numerals. The 52 upper- and lower-case letters are the only identifiers to which the user can assign numerical values. (If pressed, he can extend the set by indexing letters, but indexing is normally used in the customary way—to identify elements of vectors or matrices.) Again, generality of expression, single-letter identifiers, and two sets of groupers all contribute to readability. (For an experienced typist, readability implies writeability as well—text is easy, expressions take time but can be mastered, highly encoded implicit material is difficult.)

JOSS represents all numbers internally in scientific notation—nine decimal digits of significance and a base-ten scale factor with an integer exponent in the range  $-99$  through  $+99$ . JOSS presents an exact input interface, familiar decimal arithmetic internally, and an exact output interface. Addition, subtraction, multiplication, division, and square root are carried out by JOSS to give true results rounded to nine significant decimal digits (except on overflow which yields an error message, or on underflow for which zero is substituted). The decimal nature of JOSS gives the user easy control over exact calculations that would require especially careful attention in a binary system.

The functions in the language include a set of logical functions which, together with the numerical relations and *and* and *or*, lead to powerful direct expressions of conditions which can be attached to any step. Care has been taken in a basic set of elementary functions to hit certain “magic” values on the nose

and to provide reasonably full significance of results. The general exponential routine to compute  $a^b$ , for example, factors out error situations and the special cases of  $b = 0$ ,  $a = 0$ ,  $b = 1$ ,  $b$  an integer and  $a$  an integer power of  $10$ ,  $b = .5$ ,  $b = -.5$ , and  $b$  an integer with  $2 \leq b \leq 29$ , before resorting to  $\exp [b \log(a)]$ .

The interpretive technique on which JOSS is based enables the user to edit his stored program freely and quickly—even when JOSS interrupts at the user's request or suspends work on a task to report an error. Inserting and replacing steps or forms is implicit in the treatment of any new line of input. Deleting and typing are called for explicitly and the language provides “handles” at various levels of aggregation so the user isn't forced to do his editing piecemeal at the level of individual steps, forms, and values. Steps are organized into parts according to the integer parts of the step numbers. Parts then become units that can be typed or deleted, as well as natural units for specifying procedures in hierarchical fashion. Values, too, may be organized into vectors and arrays if indexed letters are used, and the letters by themselves may be used to refer to entire arrays for purposes of typing or deleting. Still higher aggregates may be typed or deleted by using the expressions: *all steps*, *all parts*, *all forms*, *all values*, and *all*.

JOSS and the user take turns controlling the typewriter. It is critically important that the current status of JOSS with respect to any task it may have been working on be perfectly clear each time control is returned to the user. To this end, JOSS transmits error messages, interrupt messages, and stop messages to distinguish these states from the state of having just successfully completed a task. The user obviously does not want a message for successful completion, because it would be so frequent and because it would intrude on his formal output. Error messages are of two

types: those that report violations of language constraints (such as indices not within the permissible range of integers from 0 to 99); and malformations of expressions, steps, etc. The first type is infrequent and the message is long enough to be very explicit about the violation. The second type covers a multitude of situations which are easy for the human eye to detect, but for which a precise error message is extremely difficult. All these errors are reported by the very brief message "Eh?". Thus, the user is forced to read his step to find the error, rather than possibly being misled by a message unrelated to the actual error. In every error situation, the user is able to proceed. Frequently he can repair an erroneous step or form and continue with a *Go* command. Some errors may require that the user ask JOSS to start over after the repair, which is accomplished by simply giving JOSS the same *Do* command used to initiate the task. Even when JOSS has run out of space in pursuing the task, it stands ready to help. The user may ask JOSS to delete portions of the program which are no longer essential to getting final answers (such as forms or steps no longer needed) and to continue with a *Go* command, this time with additional space for JOSS to work in.

The user need do no preplanning in composing his procedures before sitting down at the JOSS console, since he can depend on interacting with JOSS to perfect his program. This ideal situation holds for the two users in RAND who have personal consoles. The other stations are public and, because of heavy usage, some users prefer to plan their work before going to the console—but it isn't necessary.

All input to the system is free form. It is unreasonable to demand that certain items of input be typed in specified columns on the page. On output, however, it is important that the user be able to require that JOSS type answers in conveniently specified forms. It is also important that JOSS choose a reasonable output form when the user hasn't specified one. JOSS' choice here is one number per line. Each number is identified by the very expression used in the step calling for the output. JOSS tries to line up equals signs and decimal points, and uses fixed-point notation except when the

magnitude of the number makes this unreasonable.

For formal output, the user has the entire width of the line in which to specify the literal information and the blank fields to be filled in with numeric answers. Just two types of fields prove adequate. A string of underscores with an optional decimal point indication is used for fixed point. A string of periods specifies a tabular form of scientific notation in which only the digit part and the corresponding exponent part are typed with the base ten understood. The number of digits typed is determined by the length of the field, and JOSS rounds the answers to fit the fields. *Page* and *Line* steps may be used to direct JOSS in formatting the output. As mentioned above, JOSS relieves the user of having to count output lines, by maintaining the line number on the page as the value of the dollar sign. The user can, for example, call for a new page at line 50 (i.e., "*Page if \$=50.*"). JOSS provides margins automatically, and identifies each page with time, date, and user's initials typed at the very top, where it can be clipped off if the page is to be incorporated into a report. The saving of time and errors by eliminating necessity for transcription of results is no small part of the system's attractiveness.

It should now be clear that the user always interacts with JOSS at a reasonable language level, never in machine language, and that the suggestion to think of JOSS as a computing aide is entirely appropriate. In fact, except for machine malfunctions, no lower-level model of JOSS' activities can be used to explain behavior of the system which is not adequately explained in terms of the simple higher-level model. Thus, there are no JOSS system experts to call in for consultations—the checked-out user can explain every result even though he has no knowledge whatever of JOHNNIAC, the system routines comprising JOSS, or the representations of the entities of his program.

#### *Implementation*

The administration of the time-sharing aspects, drum slots, line buffers, states of remote consoles, selection of tasks, etc., is accomplished by detailed but straight-forward

machine-language routines in JOHNNIAC. The priority scheme never shuts out any user indefinitely. It responds quickly to input signals, such as carriage return, and keeps output-limited stations typing at full speed. The routines for interpretation, execution, and sequence control of the user's program, however, represent solutions to many new problems. The user's block of information initially contains certain tables, storage for value assignments (to 52 letters), heads of empty push-down lists, working storage, and a list of available space units.

List processing routines are used to store away the literal strings of characters for steps and forms as list structures, and to perform inserts, replacements, and deletes on these structures. Similar routines are used to build list structures holding numerical representations of the elements of arrays, and to perform inserts, replacements, and deletes on them. A vector, then, is represented by a list of elements, each labeled explicitly with its index. It need not be dense, since values are looked up by scanning the list for a match on the explicit index. Similarly, a matrix is represented by a list of lists of elements where the lists for the rows also carry explicit indices. Pushdown lists for operators and operands, as well as an auxiliary pushdown list, are used in the process of evaluating a numerical expression. The evaluation is programmed conveniently as a recursive routine. The most elaborate list structure arises in the bookkeeping JOSS must do to record the current step number at each level in a hierarchical task. Each *Do* causes JOSS to descend a level to perform the required part as a subroutine, then return and advance from the *Do*. If the *Do* step carries a *for* clause, then a list structure is built to record information essential to control of the iteration. This discussion is to point out that list processing is a cornerstone of the JOSS implementation.<sup>7-9</sup>

But, the list processing too is strictly an internal matter to JOSS and is completely sealed off. The user reaps the benefits in flexibility and interaction at a high language level. The challenge was not in the list processing as such, but rather in the clean-up and backing off to

the beginning of a step to report an error to the user. It was essential that no irreversible change be made in the user's block until the interpretation of a step could guarantee that the execution would proceed to completion without an error. This consideration had to be modified slightly for the *Do* step with a *for* clause, but satisfactory stopping positions were found, even for the case where the user deletes a part being iterated. This attention to detail has paid off. It is most satisfying to watch open-shop users with no previous computer experience, and little JOSS training, extract themselves from errors with JOSS' help, then continue with their problems.

## CONCLUSION

JOSS now has more than 100 qualified users among staff members at The RAND Corporation. Their most frequent requests have been for more scheduled JOSS time, for more storage space, and for long-term storage of programs.\* No one complains of the speed, although JOSS is slow. Everyone is enthusiastic about the simple language and ease of interaction. The distinguishing features of JOSS are: typewriters with an excellent touch and carefully selected keyboard; quick response to trivial requests, report-quality output; highly readable language; English capitalization and punctuation rules; exact input; familiar decimal arithmetic; exact output; no declarations; easy editing; powerful language for small numerical problems; and high-level language interaction at all times.

In this designer's view, the acceptance of an open-shop computing system depends on the little things—hundreds of them!

## ACKNOWLEDGMENTS

Many individuals have contributed to the building and installation of JOSS. The critical feedback and enthusiastic response of N. Z. Shapiro, A. C. Smith, and the other initial users of JOSS was invaluable.

\* In July of this year, JOSS service was extended into the evening hours. At this writing, a version of JOSS is being prepared to double the present user's block of 512 words and to record programs in punched cards.

## REFERENCES

1. BAKER, C. L., *JOSS: Scenario of a Filmed Report* (The RAND Corporation, Santa Monica, June 1964), RM-4162-PR.
2. BOILEN, S., FREDKIN, E., LICKLIDER, J. C. R., and MCCARTHY, J., "A Time-Sharing Debugging System for a Small Computer," Proc. 1963 SJCC, 51.
3. COFFMAN, E. G., JR., SCHWARTZ, J. I., and WEISSMAN, C., "A General-Purpose Time-Sharing System," Proc. 1964 SJCC, 397.
4. CORBATÓ, F. J., MERWIN-DAGGETT, M., and DALEY, R. C., "An Experimental Time-Sharing System," Proc. 1962 SJCC, 335.
5. DUNN, T. M., and MORRISSEY, J. H., "Remote Computing—An Experimental System. Part 1: External Specifications," Proc. 1964 SJCC, 413.
6. MCCARTHY, J., "Time-Sharing Computer Systems," *Management and the Computer of the Future* (MIT Press, Cambridge, and Wiley, New York, 1962), p. 221.
7. BOBROW, D. G., and RAPHAEL, B., "A Comparison of List-Processing Computer Languages," Comm. ACM 7, 231 (1964).
8. NEWELL, A., and SHAW, J. C., "Programming the Logic Theory Machine," Proc. 1957 WJCC, 230.
9. SHAW, J. C., NEWELL, A., SIMON, H. A., and ELLIS, T. O., "A Command Structure for Complex Information Processing," Proc. 1958 WJCC, 119.

# CONSEQUENT PROCEDURES IN CONVENTIONAL COMPUTERS\*

*Donald R. Fitzwater*  
*Institute for Atomic Research and Department of Chemistry*  
*Iowa State University, Ames, Iowa*  
*and*  
*Earl J. Schweppe*  
*Computer Science Center, University of Maryland*  
*College Park, Maryland*

## 1.0 INTRODUCTION

The complexity of the tasks performed by computer systems has been expanding rapidly throughout the short history of these machines, but only in recent years has the basic feature of sequential control of operations been seriously violated. The large computing system of the future will have multiple processing capabilities and will be operated in a shared manner in order to obtain the potential efficiencies and expansions of application areas which are possible in such a system. A shared computer system will be heavily dependent on real time interactions with people and other machines. The effectiveness of such expansion in the application areas of shared systems depends upon advances in both hardware and software structures, and upon the feedback between them. In order to solve the language problems of such a system, it is not sufficient to try to find a more convenient language to describe conventional program structures. Brown<sup>1</sup> has indicated the need for a new concept of programming in the environment and has discussed many of the necessary features including the ability to leave sequencing control in the hands of the system.

The concepts of consequent procedure networks, discussed by Schweppe and Fitzwater<sup>2</sup>, provide an effective framework for consideration of hardware and software design for such systems. Although the use of such general network implementation in controlling computer complexes would be quite effective, their full implementation within a given conventional processor becomes inefficient. The purpose of this paper is to discuss the application of a related but somewhat degenerate form of consequent procedure networks inside a conventional computer.

The results of this study have been used in the design of a shared on-line computer control system as described by McFarland, Fitzwater and Stewart<sup>3,4</sup>.

### 1.1 *Need for Consequent Procedure Networks.*

The procedures which are available in a complex system must be carried out in a sequence which is dictated by the demands of the people or equipment involved and the existence of the necessary inputs. Since this order is determined only during execution of the procedures—and indeed may vary from one execution to

---

\* Contribution No. 1509 work was partially supported by the Ames Laboratory of the U. S. Atomic Energy Commission and partially by the National Aeronautics and Space Administration under grant NsG-398 to the Computer Science Center of the University of Maryland.

another—the program structure should be an invariant of the order of execution. Clearly most present languages and systems do not allow such flexibility. The requested procedure itself may interact asynchronously with the requesting entity as well as other entities and such interaction may arise from a variety of causes, some of which are requirements for competitive processes, random demands for data acquisition, and indeterminate processing times. As a result the language used to describe procedures should contain, neither implicitly or explicitly, any artificial requirements as to the order in which operations are to be carried out.

Many ad-hoc system programs have been designed and operated successfully within real time computers. These systems have usually been dependent on a specific design for a specific set of tasks which are known at the time the system is developed. An excellent example of this type is the multicomputer programming system described by Pickering, Mutschler and Erickson<sup>5</sup>. In this case any structure which will process the specific tasks is satisfactory. The problem of designing a general purpose system for processing unspecified procedures for possibly unspecified users is quite different and is much more akin to the design of a general purpose computer. Obviously, in such a system, one cannot guarantee that all of the unspecified requests of the unspecified users can be satisfied. The goal is to design a general purpose system which is capable of satisfying a large class of such requests. Such a system must recognize that the structure of the processes which it must carry out is not sequential in nature.

An implicit recognition of non-sequentiality in an on-line system may be effected by the technique of memory swapping as discussed by McCarthy, Boilen, Fredkin and Licklider<sup>6</sup>. In this scheme, the currently requested programs are run for a short time each in sequence. The turnover rate is made fast enough so that the effect on the serviced entity is that of a somewhat slower computer. The major advantage of such a system is that conventional languages and coding structures may be used in each user's program.

The major disadvantage of such a technique is that the consequent relationships of sub-pro-

cedures are not explicitly represented in the structure of the conventional language and cannot be implicitly recognized except in rather special cases. As an example, the use of priority overlapped input-output equipment in FORTRAN programs cannot be expressed explicitly and is only sometimes expressed implicitly in the structure of the systems package. The programmer describing a procedure has no direct control over such implicit structures. If the only non-sequential operations to be carried out by the system involve simultaneous reading, writing and computing it is possible to provide implicitly for such operations and some FORTRAN systems do this. When the system is operating under a hierarchy of priorities or priority interrupts the implicit recognition of non-sequential structures lays impressive demands on the system package design. These demands become oppressive if further, non-standard non-sequential operations are required in describing the procedure. Because such structures may be peculiar to a specific task, implicit recognition of such general structures in the system package is neither feasible nor desirable.

Although, in a specific system, the decision to relinquish any part of conventional language and systems structures must be carefully considered, the inherent awkwardness of conventional system structures for general purpose real time systems leads to a hope for a more natural and effective structure which does not place such severe demands on the user, programmer and systems designer.

The use of consequent procedure networks is an alternate and effective approach to a solution of the language and system structure problem. The introduction of non-sequentiality occurs in a natural and flexible manner.

### 1.2 *Operational Requirements.*

The computer system must satisfy the needs of the user, programmer and systems designer. There is no a priori reason for assuming that these needs are identical and can be represented in the same communication languages and techniques. There is little point in re-examining language and systems structure unless these needs are re-examined and the results used in the language and systems study.

In general the user is interested in requesting the application of a specific procedure to a specific set of data to produce certain required responses. Although the nature of the data, procedure, or response may vary widely, the requirement that the system must accept and process such requests for the user is sufficient. The sequences of such user's requests and the data descriptions are the responsibility of the user and he must therefore be given the ability to define these as he sees fit. The computer system must supply a flexible means of communication for such requests and data and a convenient language in which they may be described. The user may also be interested in supplying a priority parameter to be used in scheduling.

At the current level of discussion there is no implied distinction between human and non-human users. The formal language or structure of a request for a non-human user may be quite different from that appropriate to a human user. This does not present any serious problems since the system must be able to accept requests in a "language" suitable for the using entity. Of course, not all interacting entities have the status of a user. Some may merely supply or receive information.

The programmer has a two fold relationship with the computer system. First, he must define the operations which will perform the desired task in the form of a procedure. Second, he has the same requirements as a user while entering the procedure into the system and during the checkout process. The system must provide an appropriate language for the definition of a procedure. There is no reason to assume that this language should be the same as for the user's requirements. Because many sequencing operations are under the control of the user or the system, the language must provide sequence independent structures. In general, the programmer should not care who or what might call his program into action. The programmer is merely defining a procedure. The sequences of such procedures may be dictated by the user or incorporated in some encompassing procedure. In addition, the execution of a procedure may involve external real time interaction of undefined sequence.

The systems designer is responsible for the choice of computer hardware, procedure lan-

guages and operating systems. The design must satisfy the requirements above for users and programmers and should be realizable in an efficient manner. The systems designer must supply translators for the various languages used and must supply an operating system that will accept requests and data. The requested programs must be obtained by the system, supplied with actual parameters, scheduled and executed under system control. Because of the special requirements of such a system program, the user and programmer languages may have to be supplemented with a system language which gives to the systems programmer complete access to all hardware capabilities. The programmer languages do not need such complete access and may be designed accordingly.

### 1.3 *Consequent Procedure Networks.*

The requirements outlined above can be met in a natural manner by the use of consequent procedure networks. The use of procedures in describing a program has a long history of effective use. The network description of processing flow provides a natural way of introducing non-sequential structures of almost any desired type. A consequent procedure network consists of procedures, linked into a network by the programmer, which are executed only when each recognizes the appropriate conditions for its execution. These conditions rather than the sequence of the statements define the network.

In general form, the specifications of the input data entities and the connection network of the procedures involved is sufficient to define the program. The mere existence of the specified data is enough to sequence the various procedures in a network. In the general form, data entities will carry network identity and may have to be stacked in input queues. The manipulation of such entities, in a conventional computer, become somewhat inefficient if full generality must be maintained.

The general form of consequent procedure networks loses many of its advantages in simplicity and efficiency when implemented on a single computer without substantial multi-processing capabilities. The logical complexities in implementing such a system on a conventional processor are severe. A degenerate form of the procedure network is needed in a conventional computer. The general procedure in a pro-

cedure network may represent the operation of a unique piece of hardware or it may represent a body of coding within a single computer. Since the first meaning above is appropriate to a computer complex or to a multiprocessor but not to a single processor we will use the second meaning. A procedure represented by a body of coding may have input queues representing various calls on the procedure or, for each call, the body of coding may be replicated as a distinct entity. For our purposes, the inherent simplicity of the last interpretation is a very promising line of approach to the introduction of non-sequential structures in conventional computers.

We are thus lead to a dynamic tree structure in which branches are executed in non-sequential fashion. The non-sequential characteristics enter at a very fundamental level in the resulting language structure.

## 2.0 TASK TREE STRUCTURES.

A procedure is a description of a body of coding designed to operate upon its formal parameters. When the appropriate activation conditions are met, a task is created. The process of creating a task is carried out by the system in response to a request from an existing task which recognizes the activation conditions. Although the structure of a procedure and of the system is unusual in this system, the creation of a task is essentially that of retrieval of a subprogram from secondary storage, allocation into primary storage, and of replacement of formal parameters by actual parameters. Having created a task, the system will then schedule the task for execution. The completion of the task is recognized within the task itself and the system is requested to erase the task. Note that a task has only a transient existence while it is actually operating upon a specific set of actual parameters. A given procedure may be used simultaneously in several separate tasks since each call for the procedure causes the creation of a unique task. Because of this, recursive structures may be implemented in recursive calls for procedures with no further considerations.

A task tree consists of a task and its associated subtasks. Although a certain task tree may exist implicitly in a set of library proce-

dures, it exists explicitly only in the primary memory as a task and a set of subtasks which may in turn have subtasks. The task tree will grow in response to requests for subtasks and will shrink in response to requests for erasures of tasks.

### 2.1 *The Primary Task.*

A mechanism for the creation of task trees in response to a request from a user is distinct from the mechanism which is built into the system to process requests from existing tasks for new subtasks. The language of such a request from the user must be designed for the convenience of the user. This requires a somewhat more sophisticated analysis by the computer than is required to process internal requests from existing tasks. In addition, the task tree creation process involves certain housekeeping which is not required for extensions of an existing task tree. Consequently a special task, called the primary task, must exist as the head of all user requested task trees. This primary task will create a main task of a task tree in response to each user request.

The services offered to the user by the primary task depend upon the requirements of a specific implementation. Since the primary task is the only place the system interacts with the user (as distinct from the programmer) much thought must be given to user language requirements. Many compromises, due to local conditions, would be expected here. In what follows, only suggestions of possible features to be provided by the primary task are made.

The primary task must accept a user request in the external user language and analyze the meaning of the request. The primary task will keep track of current (or anticipated) hardware requirements and if the current request cannot be satisfied because of hardware limitations, it can be stacked into secondary storage for later response. If a priority user request is permitted, and cannot be satisfied because of hardware limitations, the primary task may request (via a global variable) a lower priority task to terminate its operation and lodge a request for its later continuation. The priority request may then be handled in a normal fashion. The low priority tasks would be programmed in such a manner as to recognize the suspension of operation request and relinquish



its hardware requirements, leaving only a restart task in memory.

If there are hardware facilities which are currently unused, the primary task will examine its pool of unprocessed user requests for suitable tasks. A scheduling algorithm may be used to prevent the more demanding requests from being indefinitely postponed.

The system must be stable to transient overloads in the sense that it must not saturate in a logical blockage of processing. A transient overload may impair, temporarily, the reflex time of the system but the system must continue to work on its backlog. One form of logical blockage which is quite disastrous is to find that none of the task trees can be completed unless a new subtask can be created and there is no more available core space. This might be handled by removing, temporarily, a portion of the task trees in order to finish the remainder. This procedure is fraught with substantial complications. A simpler and more satisfactory procedure is to have the primary task monitor the current core requirements and reject or stack requests which could result in this condition. This may be done by supplying, with each task, a parameter which defines the minimum core space required for completion. This minimum may be much less than the maximum core space required for completion. This minimum may be much less than the maximum because under crowded conditions, some subtasks which might be done in parallel are done in sequence without intervention by the user or programmer.

Once the primary task has determined that the current request can be satisfied by the available hardware, a normal request of the system is made for the creation of a task which will become the main task of the task tree designed to answer the request. The parameters for this task are provided by the user as a part of his original request.

The user may wish to monitor or alter the progress of the program. The primary task will therefore accept requests for display or modification of the parameters of the original user request. Some of these parameters may be programmed to provide the desired information.

The primary task may also provide notification to the user of the real time suspension or completion of the requested task if desired.

## 2.2 *Main Task.*

A main task is a task which has been created by a request from the primary task in response to a request from a user. There will be a main task for each user request currently being serviced. The main tasks may arise from different users, thus providing for sharing of the system among users.

The distinction between a task and a main task is in its position in a task tree and not in its procedure. Any task may be requested explicitly by the user, thus forming a main task, or implicitly by the user through the task tree, thus forming a subtask. Any procedure in the library may be requested as a main task or as a subtask as is currently required by the user. For example, a subtask designed to invert a matrix may be called as a main task by a user with a specific matrix or as a subtask to a main task which is to solve a specific set of equations. This property has important consequences in code debugging and in real time control situations.

## 2.3 *Subtask.*

With the exception of priority tasks (which will be discussed in a later section) all tasks have the same structure. A simple subtask is a task which has no associated subtasks. A task which has potential subtasks but does not in fact request their creation is not a simple subtask.

A procedure body consists of a sequence of statements which are executed in a normal fashion. One additional type of statement, a subtask call, is provided. A subtask call consists of library name, an activation parameter, and a list of actual parameters which provide the information to be used in a request for the creation of a subtask. The activation parameter is a Boolean variable. A true value of the activation parameter implies that the appropriate conditions for creation of the specified subtask do now exist, and that it is appropriate to request the creation of such a subtask.

If, during the execution of a subtask call, the activator is false or if the subtask already ex-

ists, no further action is required and the next statement will be executed. If, however, the activator is true and the subtask does not exist, a request for the creation of the subtask is lodged with the system and the next instruction is then executed.

The system will subsequently request the appropriate procedure from secondary storage and go on about its business. When the specified procedure is available, allocation of storage is made, the procedure is relocated, actual parameters from the call are supplied and a request for execution is made of the task scheduler. At the appropriate time, the scheduler will make an actual call for the task and cause its execution.

When the system scheduler executes a task it does so by turning control over to the coding body of the task. This coding body acquires the references to the actual parameters and executes the set of sequential instructions referred to above. The process described above is then repeated. The set of sequential instructions act primarily as an activation scan of the potential subtasks. Statements to perform other useful operations may also be embedded in the activation scan. Since activation conditions are changing dynamically, further activation scans must be made at later times. In the absence of hardware multiprocessing or associative memories for the storage of activation parameters, efficiency dictates that the system must not continually monitor changes in activation parameters. An effective and economical compromise is to stimulate an activation scan of a parent task each time one of its subtasks either completes or requests an activation scan of its parent.

The use of an associative memory which would eliminate the need for scanning activators would make it possible to reduce the housekeeping and increase the flexibility of the activation concept. An alternative approach would have a second central processing unit carry out the scanning, allocation and other housekeeping. In the later form, queue manipulation becomes feasible and the very powerful general concepts could be implemented effectively.

A task is considered completed if, at the end of the execution of its sequential instructions, no subtasks are currently active. Upon completion of a task the system will make the hardware components involved in that task available for use of other tasks and the task will cease to exist.

#### 2.4 *Simple Tasks.*

If the sequence of statements in the body of a procedure does not contain a subtask call statement, the statements will be executed sequentially and the task is ended. Such a task is called a simple task. The use of simple tasks on any level of a task tree is exactly the same as a conventional subroutine, except for its activation and scheduling. Indeed, most conventional programs could be considered to be simple subtasks to the conventional monitor.

#### 2.5 *Priority Tasks.*

One of the weaknesses of conventional languages is the difficulty of describing hierarchical priority interrupt routines. In the present structure, this is accomplished by a simple task. There is very little reason for explicit non-sequential operation below the priority interrupt level and so the use of a simple task for priority routines is quite appropriate.

A priority task differs from a simple task in that it has three coding bodies associated with it. The first coding body functions just as a simple task whose job it is to create an entrance to the second coding body which defines the appropriate process to follow a priority interrupt. The second coding body then awaits one or more priority interrupts and makes its own decision on when its task is completed. The second coding body, when the task is completed, makes a special request to the scheduler for the execution of its third coding body. The reason for the third coding body is that a task may not relinquish control in the priority mode to anything except the point at which the priority interrupt occurred. The third coding body then releases control to the parent task.

The first and third coding bodies may be used also for initiation and termination respectively of external operations associated with the interrupt.

In the normal mode of operation, the conditions giving rise to an interrupt are not cre-

ated until after the corresponding priority task has been requested and created. Thus the priority reflex time is not dependent on secondary storage or transfer rates. This does imply that all priority tasks must exist whenever their priority interrupts are currently expected. The priority tasks handle the high speed reflex requirements of the system and the non-priority routines handle housekeeping and on-line processing. A given task may, of course, involve subtasks of both types.

A very useful priority task is one which recognizes a clock pulse as its interrupt entrance condition. Control is given to the task in the priority mode and it releases control to the next clock priority routine in the priority mode. The system is responsible for recognizing the clock pulse interrupt and for scheduling the pending clock priority tasks. Such routines may be used to monitor control operations or for periodic data logging.

### 2.6 *Activation.*

We can associate a status with each variable. The status of a variable has two values, active or inactive. For example a matrix  $M$  with inactive status may be given active status when a particular set of values have been assigned to its elements. However, unless we wish to introduce queues, we must not assign a new set of values to the matrix  $M$  until it returns to the inactive state.

The activation parameter for a subtask call may be the state of a variable or a Boolean variable derived by logical manipulations of other variables or states. The concept of the consequent subtask call requires that a request for the subtask be made of the system each time the activator goes active. Again, in order to prevent queuing, we must ignore any activator for a subtask which has already been requested until that subtask completes. This is where we lose the major portion of the generality of consequent networks with queues.

The activator is reset to inactive upon completion of the corresponding subtask and a new call is permitted if the activator should subsequently become active.

The more general concept of activation results in fascinating program simplicity and in

almost prohibitive housekeeping requirements in a conventional computer system. We will consider only the more restricted form which does not generate queues in this paper. Unfortunately, this restricts the programming use of activators almost to that of binary switches.

### 3.0 *Use of Task Trees.*

The use of non-sequential task trees has all of the advantages of subprogram structures with the added flexibility that the programmer need specify sequential relationships only where they are known. Non-sequential relationships may be controlled by the system. Because these relationships are recognized in the structure of the programming language, the programmer is relieved of the responsibility for them and the system is given the information it needs in order to handle them, thus simplifying the job for everyone. Storage allocation and scheduling may be handled by the system. The programmer need not care when his routine is to be used or where.

### 3.1 *Natural Description.*

The consequent structure is well suited to describe what systems do whether they be corporations, job shops or programs. Indeed, this type of structure could be used to describe the operations of a business in recognizable form using programming structures which have their counterparts in administrative structures. The restriction of not permitting queues is similar to abolishing in and out baskets in an office. This restriction is almost unavoidable until we can get effective computer hardware for such "baskets".

The generalized procedure network is an effective model of the way human organizations operate and the degenerate form discussed here is a compromise which retains substantial power. This natural organization suggests the basic power and simplicity which is possible in such a computer system. Many of the burdens of organization and sequencing are assumed by the system. Because the language is designed for this, the system becomes relatively simple.

### 3.2 *Language Structures.*

We will consider only the language appropriate to the programmer. The language in which

a user would make a request is simple but quite dependent on local boundary conditions. The language used in writing the system might be any which offers appropriate flexibility in controlling the specific computer.

The object program structure produced by a source language of the type under discussion is substantially different from normal usage and is closely integrated with the operating system. The source language itself is quite similar to conventional languages and is easily defined.

In FORTRAN, the addition of a subtask statement (similar to sub-routine) and a call subtask statement (similar to call) will suffice. Because of the demand for Boolean variables, a FORTRAN version permitting such variables should be used. The ordinary call and sub-routine statements could be retained if desired. Note that, although the new language is quite similar to the old, the method of use of the language statements is quite different.

In many respects, ALGOL forms an excellent vehicle for such non-sequential structures. In this case the procedure would be modified to include the activator and the subtask would replace both the block and the procedure concepts. This results in a much simpler language which retains the advantages of ALGOL and the power of the non-sequential structures.

### 3.3 Multilevel Control and Debugging.

Since all tasks have the same calling structure, any task may be used as a main task or as a subtask. For example, one might have a task which controls some external equipment through its use of various subtasks. If the current control program is unsatisfactory for some special purpose, the user may assume control on a lower level of the task tree and determine his own sequencing of the more rudimentary control functions. This is similar to the operation of a space satellite in the "fly by wire" mode.

A further advantage of this mode is in debugging. A task tree may be debugged from the end of its branches inward by calling each subtask as a main task and supplying the appropriate test parameters with his request. No special main program need be written and debugged in order to test an individual procedure.

## 4.0 AN EXAMPLE OF CONSEQUENT PROCEDURE USAGE.

In order to give an example of the usage of consequential procedures it is necessary to introduce a language in which to describe them. Although the syntax of the language used does not differ very much from conventional ones, the way in which the language is used is quite different and can best be appreciated by study of an example. The following casual definition of a language based on ALGOL may suffice for illustrative purposes but is certainly not a sufficient definition for the use in an actual system. The object program structures are dependent on a specific computer and are not discussed here. More details as to a language and system structure on a specific computer (the SDS 910) are to be found in the paper by McFarland, Fitzwater and Stewart<sup>4</sup>.

### 4.1 Language for the Description of the Example.

Let a consequent procedure have the form of an ALGOL procedure whose main program is the parent task. The procedure body is a block and contains no further blocks. The procedure statement is replaced by a subtask call with the following syntax:

```
<Subtask Call> ::= <Procedure Identifier>
                    <Activator>
                    (<Actuator> ; <Actual
                    Parameter List>)
<Activator> ::= <Boolean Variable>
                | <State of a Variable>
<State of a Variable> ::= [ <Variable> ].
```

The state of a variable (active or inactive) is a Boolean variable associated with each variable. We will place the further restriction that a subtask call may not be used in a compound statement or in a simple or priority subtask.

The coding body of a simple task may be written in a machine language code which is otherwise unspecified.

Two pseudo blocks which are executed only once each may be included as the first and last statements in the body of a procedure. These are defined by *enter*...*end* or *exit*...*end* with *enter* or *exit* replacing a *begin*. An *enter* block is executed only on first entrance and in a non-

priority mode. An *exit* block is executed just prior to the subtask completion exit and is always executed in a non-priority mode.

In order to simplify the language description given here, we will use descriptions of statements in place of the statement when the explicit syntax for such a statement has not been given. Such descriptions will be placed between parenthesis to prevent confusion with comments.

#### 4.2 Description of Example.

This example involves the control and logging of data from a simple external device and will illustrate the interactions of various priority and non-priority tasks. The same types of interactions, although there might be no need for priority tasks, would occur in some data processing applications.

Let us suppose the external device has three asynchronous processes. Process one sets a value of a variable and holds it during the remaining two processes. We may modify certain device parameters so as to cause the variable value to assume the desired value. The second two processes are very similar but are not sequentially related. We may initiate either process and at a later time (signalled by an interrupt) we may read in a vector of 3 values for each process. Computations using these values then produce a new setpoint for the first process which is set and held dynamically for the remaining two processes. The value of each setpoint is typed out and a wait, if necessary, is made to prevent stacking demands on the typewriter.

In case of failure, for any reason, to complete the above cycle within a specified time limit, we will assume some disaster has occurred, shut down the external device and remove the task tree from storage as an emergency measure. The system will so notify the user.

The operator may change the value of the Boolean variable QUIT to true and cause the task tree to finish normally.

#### 4.3 Coding for Example.

The user would request the task called MEASURE and supply appropriate values for

its parameters. The meaning of the parameters in MEASURE is as follows

PA, PB—define the priority interrupt entrance. If an interrupt associated with PA occurs, the priority subtask associated with PA will be entered in the priority mode. If the subtask does not exist, it will be ignored.

LIMIT—is the integral number of clock interrupts which must be received before disaster is assumed.

I—is a counter for clock interrupts. It may be reset to zero at any time.

QUIT—is a Boolean variable which, if true, requests cessation of operation.

D—is a range bound on the value of the variable C.

NA, NB—are the norms of the vectors A, B. A, B—are vectors whose values are obtained from the external device.

C—is the value to be held in the external device by the first process.

READY—is a dummy Boolean variable denoting that the setpoint has been reached.

The *enter* block will establish the priority subtasks HOLD and TIME which will remain active for the duration of the control task. This block is executed only on first entrance to the task MEASURE.

The subsequent entries to the task, MEASURE, will be made at the first statement following the enter block. Subsequent entries will be made at the completion of any subtasks of MEASURE.

The calling of the evaluate subtask will be considered only if its input data, NA and NB, are available and the last value of C has been logged.

The remaining subtask calls will be made if during execution of the task body (an activation scan) their activators are true. The activator for a subtask will be set false at completion of that subtask. Note that the order is not important since any order of the remaining statements will produce the same results.

Since the scheduling of the subtask calls is a function of currently available storage and of

the occurrence of the priority interrupts the programmer does not know how much overlap of subtasks will actually occur on each measurement cycle. If storage is temporarily restricted, the subtasks would be executed one at a time in sequence. If the response to INPUT1 is very fast, NORM (A; A, NA) may be executed during INPUT2 and while TYPE is operating. The system will automatically take advantage of such overlapping and will give the best response, on each cycle, that is possible under the conditions then prevailing.

*complex* MEASURE (PA, PB, D, LIMIT, QUIT); *priority* PA, PB; *real* D; *integer* LIMIT; *Boolean* QUIT; *begin real* NA, NB, C; *real array* A, B [1:3];

*subtask* EVALUATE, HOLD, TIME, INPUT 1, INPUT 2, TYPE, NORM; *Boolean* P, READY;

*integer* I;

*enter* C := 0.0; [C] := *true*; READY := *false*; HOLD ([C]; C, READY, QUIT, I); TIME ([C]; LIMIT, I, QUIT);

*end* entrance block;

*if* QUIT *then go to* EXIT

*if* [NA] *and* [NB] *and*  $\neg$ [C] *then* P := *true* *else go to* SKIP; EVALUATE (P; NA, NB, C, D);

SKIP: NORM ([A]; A, NA);  
NORM ([B]; B, NB);  
INPUT1 (READY; A, PA);  
INPUT2 (READY; B, PB);  
TYPE ([C]; C);

EXIT: *end* MEASURE

The subtask HOLD is a clock priority routine. This implies that an entrance to the body of the subtask will be made in the priority mode each time a clock interrupt occurs. The *enter* and *exit* blocks are executed only once and in the non-priority mode. The variables declared in hold have the following meaning;

X—is assigned the value at the external address DIAL.

DIAL—is an address which selects the register in the external device holding the value to be assigned to X.

ERROR—is the discrepancy between the setpoint and the current value of DIAL.

The *enter* block initializes the mode switch to hold and requests an entrance on the next clock interrupt. This is executed in the non-priority mode. Subsequent priority mode entrances are made at the first statement after the *enter* block.

If the error is large, HOLD assumes that a new setpoint has been requested and READY is set *false*. When the error has been reduced to an acceptable value, READY is set *true* to initiate the appropriate subtask calls in the parent task. An activation scan of the parent task is requested to give the parent the opportunity to make such calls.

The normal exit from the priority mode section of HOLD is made by requesting another clock interrupt and releasing control.

This subtask will complete only when QUIT is *true* by requesting its removal at a later time in the non-priority mode and releasing control to the interrupted coding. The system will then enter the coding body and execute *exit* block in the non-priority mode. This results in a return to the parent task and the destruction of this subtask.

*clock priority* HOLD (A1, A2, QUIT, I);

*real* A1,; *Boolean* A2, QUIT; *integer* I;

*begin real* X, ERROR; *external address* DIAL := 1234;

*switch* MODE := SET, HOLD; *integer* N;  
*enter* N := 2; (REQUEST CLOCK INTERRUPT) *end if* QUIT *then go to* DONE;

(INPUT VALUE AT DIAL AND ASSIGN TO X);

ERROR := X - A1; *go to* MODE [N];

SETC: *if* ABS (ERROR) < 0.01 *then*

*begin* [A2] := *true*; N := 2; I := 0;

*comment* this resets the time limit clock count;

(REQUEST ACTIVATION SCAN OF PARENT) *end*;

CORRECT: (INITIATE APPROPRIATE CORRECTIVE ACTION TO HOLD ERROR AT A MINIMUM); (REQUEST CLOCK INTERRUPT); (RELEASE PRIORITY);

```
HOLDC: if ABS (ERROR) >0.5 then begin
  [A2]:= false; N:= 1 end; go to CORRECT;
DONE: (REQUEST PRIORITY TASK EXIT); (RELEASE PRIORITY CONTROL); exit (SHUT DOWN HOLDING ACTION) end
end hold
```

The subtask TIME is a clock priority routine which counts clock interrupts. If the limit is reached emergency action is taken. This subtask is active during the operation of MEASURE and does nothing but count unless QUIT is true or LIMIT is reached. If QUIT is true, the subtask completes in the same fashion as hold. If the limit is reached, something is beyond control and the entire device is shut down and a request to eliminate the entire task tree is made of the system. When this is accomplished, the system will so notify the user.

```
clock priority TIME (LIMIT, I, QUIT)
integer LIMIT, I; Boolean QUIT;
begin
  enter I: = 0; (REQUEST CLOCK INTERRUPT) end; if QUIT then go to DONE.
  I: = I + 1; if I = LIMIT then
  begin (SHUT DOWN EXTERNAL EQUIPMENT);
    (REQUEST EMERGENCY ERASURE OF TASK TREE);
  DONE: (REQUEST PRIORITY TASK EXIT)
  end LIMIT REACHED
  else (REQUEST CLOCK INTERRUPT);
    (RELEASE PRIORITY CONTROL)
  end time
```

The INPUT1 and INPUT2 subtasks are identical, at least to the degree of coding appropriate here, and only differ in the external device addresses which are used to obtain information. We will discuss only INPUT1 which uses formal parameters corresponding to actual parameters which have been discussed.

INPUT1 is a priority routine whose body is executed only once. The enter block sets up the priority entrance and initiates the external process which will, at a later time, give the de-

```
sired priority interrupt, P1, which signals that the values of A1 are ready to be read.
priority INPUT1 (A1, P1);
  priority P1; real array A1;
begin
  enter (REQUEST PRIORITY ENTRANCE ON P1);
    (INITIATE EXTERNAL ACTION WHICH CAUSES PRIORITY INTERRUPT P1)
  end entrance set up;
  (INPUT EXTERNAL VALUES OF A1 AND ASSIGN TO A1);
  (REQUEST PRIORITY TASK EXIT);
  [A1] := true;
  (RELEASE PRIORITY)
end INPUT1
```

The remaining subtasks, EVALUATE, NORM and TYPE, are simple non-priority routines that are executed only once per call and return control to the parent task. The actual parameters used in the calls have been discussed and there is little more to say about these subtasks.

```
simple EVALUATE (A1, A2, A3, A4);
  real A1, A2, A3, A4;
  begin A3:= (A1 + A2)/(A1 x A2) + A3 + 0.5;
    if A3 < A4 then A3:= A4;
    [A3] := true; [A1] :=[A2] :=false
  end evaluate
simple NORM (A, N)
  real N; real array A;
  begin N:= (A [1] ↑ 2 + A [2] ↑ 2 + A [3] ↑ 2) ↑ 0.5; [N] := true
  end norm
simple TYPE (A1);
  real A1;
  begin (TYPE VALUE OF A1) end type
```

### 5.0 SYSTEM REQUIREMENTS.

A system of the structure described may be operated on many levels. It might be used as shown in the example within a single processor or it might be used in such a fashion that the activation of a subtask calls into play a whole subset of a man-computer-machine complex.

Some of the subtasks might be performed by people while other people controlled the overall tree structure by making appropriate requests of the system. In a multiprocessor system it is not necessary but it may be desirable to specialize one of the CPU to perform the allocation and scheduling operations. Such specialization could also allow queue control and the subsequent use of the more general activation concept previously discussed. A small associative memory would be advantageous for storage of activator parameters. Indeed, this latter organization is a very intriguing way to consider multi-processor systems. We will, however, confine our subsequent remarks to a single processor system.

We have implicitly considered that the single processor has a secondary (or multi-level) store which has a size sufficient to maintain total library control and transient data storage. We are not so concerned about transfer rates between primary and secondary storage so long as average rates are satisfactory. Critical response loops may be closed through priority routines which have been installed before the high speed loop is initiated.

The most significant demand on core space arises from currently active priority tasks which must occupy storage until their task is done in order to give short reflex times. If longer reflex times are suitable, a small interrupt routine can activate a non-priority task to do the job.

Dynamic allocation of primary storage is a critical factor and can easily become the most time consuming part of the system which otherwise has a very low duty cycle that is optimally phased with the workload. It is still feasible

to relocate subtasks as they are entered but it would be much more desirable if routines could run wherever they happen to be without relocation.

The system itself requires only a small amount of core storage for the control program and tables and may itself extend into subtasks for auxiliary operations or extension of facilities.

#### REFERENCES

1. BROWN, G. W., "A new concept in programming", in *Management and the Computer of the Future*, Martin Greenberger editor, (John Wiley and Sons Inc., New York, 1962), Chap. 7, pp. 250-289.
2. SCHWEPPE, E. J. and FITZWATER, D. R., "Consequent Procedure Networks", to be published.
3. MCFARLAND, D. E., FITZWATER, D. R. and STEWART, R. M., "A Multiprogrammed Real Time Computer Control System", presented at the 1963 Fall SDS Users Conference.
4. MCFARLAND, D. E., FITZWATER, D. R. and STEWART, R. M., "A General Control System for Experiments", to be published.
5. PICKERING, G. E., MUTSCHLER, E. G. and ERICKSON, G. A., "Multicomputer Programming for a Large Scale Real Time Data Processing System", *AFIPS Proc. SJCC* Vol. 25 (1964), pp. 445-461.
6. MCCARTHY, J., BOILEN, S., FREDKIN, B. and LICKLIDER, J. C. R., "A Time Sharing Debugging System for a Small Computer", *AFIPS Proc. SJCC* Vol. 23 (1963), pp. 51-58.



# THE JET PROPULSION LABORATORY

## EPHEMERIS TAPE SYSTEM

*E. G. Orozco*  
*Jet Propulsion Laboratory*  
*Pasadena, California*

### INTRODUCTION

Predictions of the motion of celestial bodies can be presented in tabular form. These tables, called ephemerides, list position as a function of time. Positions at any time point within the range of the table can be obtained by interpolation.

Ephemerides are required in the solution of problems associated with space exploration. The solution of these problems is primarily accomplished with the aid of digital computers. Ephemerides are generally tabulated on magnetic tape to make them acceptable for use by computers.

The Jet Propulsion Laboratory has, in the past, developed tape ephemerides. Figure 1 summarizes the contents of these tapes and the programs used to read and interpolate the data. Because of increased accuracy demands, these tapes are no longer adequate.

The Ephemeris Tape System was established solely to meet JPL's needs, but the ephemeris tapes generated and the program used for reading and interpolating the ephemerides will be useful to anyone doing interplanetary trajectory work.

### GENERAL SYSTEM DESCRIPTION

The ephemeris data generated by the JPL Ephemeris Tape System covers the period from 1950 to 2000. They include tabulations in

rectangular coordinates of position and velocity components of the Moon and the nine planets. In addition, nutations in longitude and obliquity are included. Modified second and fourth differences of these quantities are tabulated for purposes of interpolation.

Numerical differentiation of tabulated positions is too inaccurate for problems in which planetary velocities are critical. For these critical cases, planetary position and velocity are simultaneously obtained from a solution of the equations of motion for the planet. The position and velocity components thus generated are mutually consistent with gravitational theory to high precision and are the best fit in the least squares sense to source position predictions.

No set of predicted motions now available can be considered final. Accordingly, the JPL Ephemeris Tape System is designed to permit the easy updating of data for any body or bodies.

Positions and velocities are carried as double precision floating point numbers (that is, to about 16 decimal places). This allows for predictions more accurate than those now available to be readily assimilated into the Ephemeris Tape System.

The JPL Ephemeris Tape System is designed to prevent degradation of the accuracy of source data during the data processing. The use of

Tape Number	Period Covered	Interpolation Sub-routine	Data Record Length (20-Day Record)	Bodies on Tape and Data Interval (days)											
				Geocentric Sun	Geocentric Moon	Venus Position	Mars Position	Jupiter Position	Earth-Moon Position	Venus Velocity	Earth-Moon Velocity	Saturn Position	Mars Velocity		
I	1 JAN 60 -19 DEC 69	INTR	596	1	1	4	4	4							
II	28 AUG 60 -19 DEC 69	INTR	515		1	4	4	4	4	4	4	4			
III	11 JAN 50 -27 JUL 81	INTR	569		1	4	4	4	4	4	4	4	4		
IV	23 JAN 50 -29 JUN 81	INTR	623		1	4	4	4	4	4	4	4	4	4	4

Figure 1.

double precision contributes toward this end. Intervals of tabulation were chosen so that interpolation yields accuracy consistent with tabulated values. Formal procedures for checking each step of the processing are included to insure that the published ephemerides are free from error.

An interpolation program has been written to read and interpolate the JPL Ephemeris Tapes. This program includes a number of useful options along with the interpolation capability. Coordinates referenced to any of the tabulated bodies as center may be obtained. Units may be AU's and AU's per day or Kilometers and Kilometers per second for the planets. For the Moon, coordinates may be obtained in Earth radii and Earth radii per day or Kilometers and Kilometers per second. Position data only, velocity data only, or position and velocity data together may be obtained. Data for all the bodies on the tape or any subset of these are available as desired. Any date between 1950 and 2000 is available for the ten bodies on the tape.

#### SOURCE THEORY EVALUATION

Source position data as generated from the theory are kept in a source tape library. The data included in the current JPL Ephemeris Tape are the most accurate predictions of lunar and planetary motion available. At the present time the best source theory for the Moon is the Brown Improved Lunar Theory. The New-

comb theories are used for Mercury, for Venus and for the Earth-Moon barycenter, with corrections to the mean elements deduced for Mercury by Clemence and for Venus and the Earth-Moon barycenter by Duncombe. The provisional Hansen-Clemence theory is used for Mars. For the outer planets, the source data are obtained from the numerical integration by Eckert, Brouwer and Clemence, with corrections deduced by Clemence to transform the data to heliocentric coordinates.

Velocities are generated by numerical differentiation for the Moon, Mercury and Neptune and by solution of the equations of motion for the other planets. The solution of the equations of motion for each body is done by a program that includes a linear regression scheme that generates and applies differential corrections to the initial position and velocity until satisfactory least squares fit to the source data is obtained. The perturbing attractions of the other eight planets are computed from positions obtained from the current Ephemeris Tape. The integration step size is chosen so as to insure twelve-figure accuracy in the calculated positions. The final corrected values of initial position and velocity are used to initiate a final numerical integration. The position-velocity output tape is written during this final iteration.

#### OPERATION

The position-velocity tapes (source and fitted) are processed by a program which cal-

culates the sixth order central differences of the data. These differences are plotted. Any missing point or inconsistent data cause large discontinuities in the plots of differences. Any such discontinuities cause rejection of the data.

Accepted data tapes are processed to calculate second and fourth modified differences of the position and velocity coordinates. A new tape is then written for each planet containing the position-velocity coordinates along with their associated modified differences.

These tapes are then merged onto a JPL Ephemeris Tape containing data for all ten bodies. The data for ten bodies are buffered and overlapped for convenience of interpolation. Due to the amount of data involved, three tapes are required to contain the fifty year span from 1950-2000. Figure 2 summarizes the contents of these tapes.

A consistency check is made between the JPL Ephemeris Tapes and the data on the pertinent position-velocity tapes. This assures that no points have been lost or displaced.

Using the modified differences to interpolate, a tape is written carrying the positions and velocities of a given body at the midpoint of its tabulated dates. These tapes are processed as source tapes and plots of sixth differences from

these tapes are checked to insure consistency of the modified differences.

DOCUMENTATION

A document describing the data on JPL Ephemeris Tapes is prepared when a tape is ready for distribution. This document includes a description of the theory from which each body's data are generated. A statement as to the accuracy of the data is included for each body. Checks performed on the data are mentioned. If the data was fitted using the integration program, this is pointed out. And, finally, the plots used in checking the data are included in this document. The Stromberg Carlson 4020 plotter is used for this purpose. An example is given in Figure 3.

DISTRIBUTION

The first JPL Ephemeris computer package is currently available for distribution. The distributed package includes a user's guide, an interpolation program, a document describing the tape contents and three magnetic tapes containing the ephemerides information.

Distribution of these tapes is being handled by the Jet Propulsion Laboratory with the exception of those involved in the Apollo project. Distribution for Apollo is being done directly by NASA.

DATA TYPE	STEP SIZE (DAYS)	NUMBER OF WORDS	FORMAT	
MOON	0.5	612	General format for all body data is double precision: 1. X Component of Position 2. Second Modified Difference X Component of Position 3. Fourth Modified Difference X Component of Position 4. Y Component of Position 5. Second Modified Difference Y Component of Position 6. Fourth Modified Difference Y Component of Position 7. Z Component of Position 8. Second Modified Difference Z Component of Position 9. Fourth Modified Difference Z Component of Position 10. X Component of Velocity 11. Second Modified Difference X Component of Velocity 12. Fourth Modified Difference X Component of Velocity 13. Y Component of Velocity 14. Second Modified Difference Y Component of Velocity 15. Fourth Modified Difference Y Component of Velocity 16. Z Component of Velocity 17. Second Modified Difference Z Component of Velocity 18. Fourth Modified Difference Z Component of Velocity	
MERCURY	2.0	180		
VENUS	4.0	108		
EARTH-MOON BARYCENTER	4.0	108		
MARS	4.0	108		
JUPITER	4.0	108		
SATURN	4.0	108		
URANUS	4.0	108		
NEPTUNE	4.0	108		
PLUTO	4.0	108		
MUTATIONS	0.5	204		Mutations are expressed as single precision values and include mutation in longitude, mutation in obliquity, mutation rates, and second and fourth modified differences for each.

- (1) Each record contains a double precision Julian date.
- (2) Each record contains a check sum of the 1862 data words.

Figure 2.

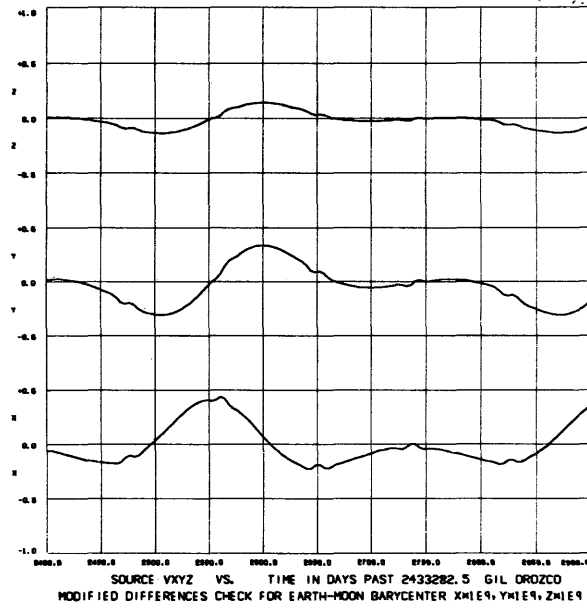


Figure 3.

# JPTRAJ

## (THE NEW JPL TRAJECTORY MONITOR)

*Nicholas S. Newhall*  
*Jet Propulsion Laboratory*  
*Pasadena, California*

### INTRODUCTION

The Jet Propulsion Laboratory of the California Institute of Technology is under contract to the National Aeronautics and Space Administration to conduct space projects such as *Ranger*, *Mariner*, and *Surveyor*. These projects require an enormous amount of computer support in both trajectory design and spacecraft tracking. To this end, three IBM 7094 systems are always available, and other equipment is operational during a mission.

A multitude of computer programs must be run for adequate mission work. These programs entail every facet of space flight—from preliminary conic studies to real-time orbit determination and midcourse maneuver—and represent the results of years of effort on the part of programmers. Each is structurally self-contained and operates alone on the 7094.

In the past, trajectory design was accomplished by running a series of individual programs one at a time and hand-carrying the results between them. The mission programs had either to operate their links individually or to run them successively and share blocks of common data. The Laboratory needed a system which would incorporate the execution and communication between (perhaps) independently written programs. A means of providing a flexible modification and checkout scheme was required. The system developed to meet these needs is JPTRAJ, the JPL Trajectory Monitor.

The use of JPTRAJ for design and mission work is illustrated in Section I. Through an example of the design of an Earth-Moon trajectory, it is shown how JPTRAJ is used to run a single sequence of programs. Real-time operation demands the interruptible execution of many such sequences. This feature along with the internal operation of JPTRAJ and the debug capability is described in Section II.

### I. USER CONTROL OF JPTRAJ

Much computer time at JPL is devoted to the design of *Ranger* lunar trajectories. The design consists of five steps on the IBM 7094:

1. The *Near-Earth Conic* program selects analytically approximate launch time and burnout energy of the optimum trajectory (Fig. 1).
2. The *Powered Flight* program simulates a rocket burn to compute position and velocity at injection (burnout) as shown in Fig. 2.
3. The *Space Trajectories* program from this point integrates numerically the flight path to the Moon (Fig. 3).
4. The *Search* program recomputes launch data, if the lunar impact point differs from the desired one, and reruns steps 2 and 3 until desired impact has been attained (Fig. 4).

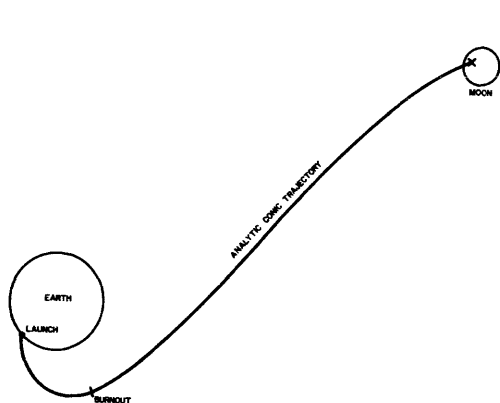


Figure 1. Approximate trajectory computed by Near Earth Conic.

5. The *TV Constraints* program uses terminal conditions (outlined in Fig. 5) from steps 3 and 4 to produce quality information about lunar television pictures.

The following set of figures illustrates the way this sequence can be set up for JPTRAJ. Rather than run the programs singly, mnemonic names for them are adopted and listed in Fig. 6 in the order in which they are to be executed.

Input data, such as initial launch azimuth (direction), are needed. In Fig. 7, each program name is followed by the data intended for it.

The transfer of computed data from one program to the next can be effected by the use of cards, as shown in Fig. 8. Symbols to the left of the program names may be used for reference to a specific word or line in the example.

The data transfer (or WANT) instructions are placed after the program that is to receive the data. They denote symbolically which pro-

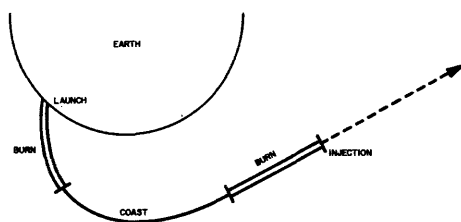


Figure 2. Trajectory segments integrated by Powered Flight program.

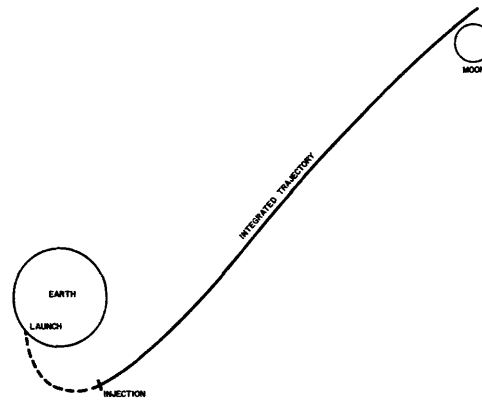


Figure 3. Numerically integrated trajectory computed from injection conditions supplied in step 2.

gram is to generate the data. The WANT after POWER may be read: "Launch time (LT) and energy at burnout (EBO) are needed from the program at symbolic location A" (NECON in this example). Step 4 in the example indicates that POWER and SPACE will be rerun as often as necessary. SEARCH is the driver and must be reentered each time SPACE completes execution. It may be denoted symbolically by the GO card shown in Fig. 9.

At this point, if compared with the five steps above, Fig. 9 will be seen to represent the logic flow. One may question the fact that when SEARCH first executes after NECON, it will attempt to read computed data from SPACE, which has not executed before. However,

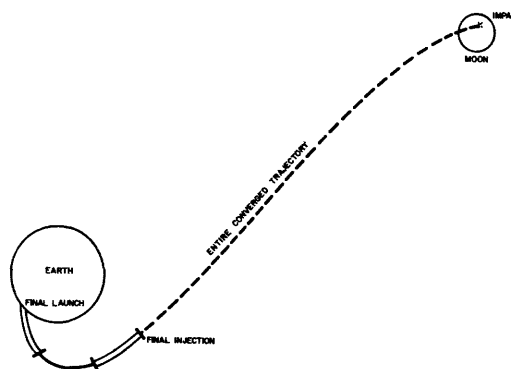


Figure 4. Converged trajectory resulting from Newton-Raphson Search.

SEARCH ignores any transferred data on the initial entry, using them only on subsequent entries. A more serious problem is that SPACE will eventually indicate that desired impact has been attained. The GO statement must be bypassed so that the TV program may execute. Under JPTRAJ, SEARCH (and, in fact, any

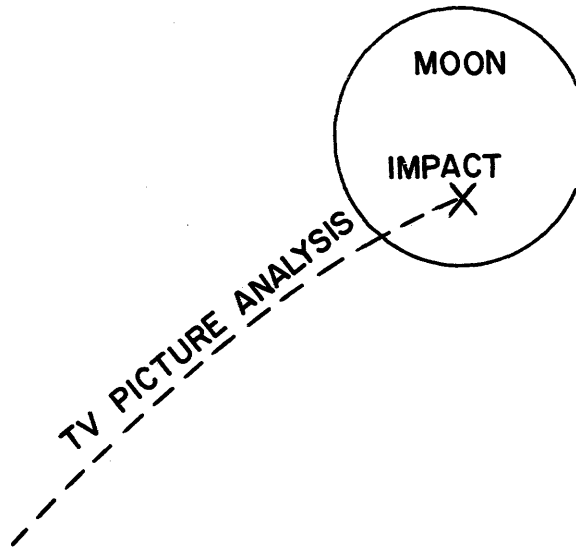


Figure 5. Phase of trajectory used for television picture analysis.

NECON (NEAR EARTH  
CONIC)  
SEARCH (SEARCH)  
POWER (POWERED  
FLIGHT)  
SPACE (SPACE TRA-  
JECTORIES)  
TV (LUNAR TV  
CONSTRAINTS)

Figure 6. Mnemonic names for programs.

NECON  
AZ = 102.0  
SEARCH  
POWER  
SPACE  
TV

Figure 7. Specifying input data for a program.

A NECON  
AZ=102.0  
B SEARCH  
WANT D,EC  
C POWER  
WANT A,LT,EBO  
D SPACE  
WANT C,R,V  
E TV  
WANT D,EC

Figure 8. Transferring computed data between programs.

```

A  NECON
   AZ=102.0
B  SEARCH
   WANT  D,EC
C  POWER
   WANT  A,LT,EBO
D  SPACE
   WANT  C,R,V
   GO  B
E  TV
   WANT  D,EC

```

Figure 9. The use of a GO card to direct program execution.

program) is capable of selecting a *computed* branch or transfer after execution. The symbol E to the right of SEARCH in Fig. 10 means that the Search program will exit to the program at location E (the TV program) when desired impact is reached.

The configuration of symbols and numbers shown in Fig. 10 is precisely that punched on cards and used as input to the computer. It is seen to resemble a limited programming language, treating whole programs as built-in functions. Such a deck of cards is called a *source deck*. Most JPTRAJ source decks will contain a much larger percentage of data cards,

```

A  NECON
   AZ=102.0
B  SEARCH E
   WANT  D,EC { EC=END CONDITIONS }
C  POWER
   WANT  A,LT,EBO { LT=LAUNCH TIME
                   EBO=ENERGY AT BURNOUT }
D  SPACE
   WANT  C,R,V { R=POSITION
                 V=VELOCITY }
   GO  B
E  TV
   WANT  D,EC

```

Figure 10

which were omitted here for clarity. When the actual cards are punched, an asterisk (\*) must occupy column 1 of any card that is not a data card, and an END card (shown in Fig. 11) must be the terminal card of every source deck.

Such is the type of input to JPTRAJ on the standard system input tape. This example has shown how a user can create a single sequence of programs to execute. In the following section, we shall see the treatment of a source deck by JPTRAJ, and how many such sequences can operate intermittently on a time-sharing basis during a mission.

## II. INTERNAL OPERATION OF JPTRAJ

All programs in use by JPTRAJ reside from day to day in permanent storage on the disk file. When a program is requested in a source deck, it is read from the disk file into 7094 core storage. It is never overwritten on disk during execution. If it is called later, it will again be read in fresh from the disk file.

JPTRAJ may be broken internally into four sections:

1. Compiler
2. Monitor
3. Loader
4. Debug

```

*  A  NECON
    AZ=102.0
*  B  SEARCH E
    WANT  D,EC
*  C  POWER
    WANT  A,LT,EBO
*  D  SPACE
    WANT  C,R,V
*  GO  B
*  E  TV
    WANT  D,EC
*  END

```

Figure 11. An entire source deck.



*Compiler and Monitor*

When a source deck is submitted as input, the Compiler makes two complete passes over it and translates it into a block of specially constructed words called an object string. (This is analogous to the operation of a standard compiler. The object string is merely an encoded form of the source deck.) The object string is stored in a reserved area on the disk file. At this point, the Compiler is no longer needed and will not be referenced again. The Monitor, a small, 640-word program, is now read from disk into core. Its sole task is to *interpret the object string* generated by the Compiler. By proper interpretation, the Monitor performs all reading of programs, saving and transferring of data, and executing of programs. It is in core storage at all times during execution. Figure 12 shows a simple flow of source deck processing.

This approach was adopted in lieu of a simple card interpreter because excessive machine time would be required for processing a source deck. The number of disk accesses needed for such an operation would be more than twice that for the existing system. This method also is a powerful tool for real-time mission operation. A mission can be broken into three categories: orbit determination, midcourse maneuver, and telemetry processing. Requirements of incoming data require frequent operation of the Data Editing program, not part of JPTRAJ. This demands interrupt and time-sharing capability of the three mission categories. Orbit determination may be in operation when interrupted for data editing and then the telemetry section may be started. Each of these categories is represented by a JPTRAJ source deck naming the appropriate programs. For mission work the JPTRAJ Compiler may be called several successive times, each time to compile a

separate source deck. By means of the DECK card shown in Fig. 13, a source deck may be assigned to a particular task. The resulting object string is stored in the correct one of several pre-specified areas on the disk file. Then, when the midcourse maneuver, for instance, is to be performed, the associated object string is interpreted by the Monitor. If interrupted, interpretation will resume later at the proper point.

The passage of data between programs is accomplished during execution as follows: When the program generating the data has completed execution, the data specified on the WANT cards are set aside in a buffer within the Monitor. Then, when the program needing the data is about to execute, the data are recalled from their "saved" area and inserted into the specified locations. The Monitor's buffer contains 225 locations, and ample disk storage is available for overflow. A total of 9,000 words may be saved on WANT cards per source deck.

Note that data passage between programs within a single source deck is completely free, but that there is no facility to pass data between *different* source decks. If programs in separate source decks are to share data, such communication must be programmed into them.

Items on data cards are converted and saved within the object string itself. A GO card merely transfers interpretation to the appropriate object string word.

*Loader*

The Compiler and Monitor are employed only for the execution of programs, and operate for real-time or production work.

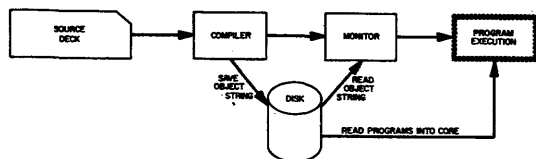


Figure 12. Source deck processing.

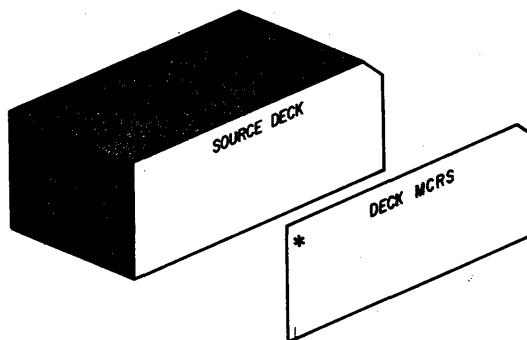


Figure 13. The DECK card and its position.

The remaining section of JPTRAJ, the Loader, is used for program editing and can be operated as a non-real-time feature only. The Loader enables a programmer to add new programs to disk storage, delete unwanted ones, and change existing versions. Any number of programs may be loaded simultaneously, and

compilations, assemblies, and binary object decks may be included with each program. A library search is performed for any requested subroutines not included with the submitted decks. Upon completion, the Loader prints a core storage map, as illustrated in Fig. 14, and writes the program on the disk file. If a source

ENTRY POINTS TO DEFINE	SUBROUTINES ATTACH	REQUESTED FROM LIBRARY OPEN	READD	PROUT	CLOSE			
THE NAME OF THIS PROGRAM IS 'JPTAP '				8/01/64				
ENTRY NAME .....	ENTRY ADD. 22331	TRANSFER VECTORS DEFINE	LOAD ADD. 22300	OCTAL LENGTH 36616	DECIMAL LENGTH 15758	COMMON BREAK 77461		
		ATTACH						
		OPEN						
		READD						
		PROUT						
		CLOSE						
PROUT	61132	OUTUS	61116	02637	01439			
FGDOUT	62301	ACTIND						
PRCNV	61217	BFLG						
PROUT2	61204	RESTKA						
PROUT3	61210	REQIND						
TSXA	63200	WRITE						
		PRCON						
		RGGSVAV						
		RGGSTR						
		CKIND						
		CKACT						
READD	63756	(IOU)	63755	00365	00245			
READR	63760							
WRITED	63763							
WRITEB	63765							
BSREC	64174							
BSFILE	64177							
REWIND	64202							
UNLOAD	64205							
ENDFIL	64210							
SETLOW	64166							
SETHI	64171							
(UNIT)	64336							
TAPEIO	64336							
(IOU)	64345	'NONE'	64342	00030	00024			
OUTUS	64374	RGGSVAV	64372	02320	01232			
BFLS	65227	RGGSTR						
ENDOUT	64760							
CKIND	66622							
CKACT	66665							
REQIND	66621							
ACTIND	66620							
RESTKA	66616							
PL1CON	66701							
PL2CON	66702							
PL3CON	66703							
PRCON	66704							
RGGSVAV	66712	'NONE'	66712	00133	00091			
RGGSTR	66774							
IOCS	67046	CLOCK	67045	04335	02269			
DEFINE	67047							
JOIN	67052							
ATTACH	67055							
CLOSE	67060							
OPEN	67063							
READ	67066							
WRITE	67071							
COPY	67074							
REW	67077							
WEF	67102							
BSR	67105							
BSF	67110							
STASH	67113							
CLOCK	73405	'NONE'	73402	00106	00070			
MINUTE	73402							
XMIN	73402							
LOADING ACCOMPLISHED								
UNUSED CORE LIES FROM 73510 THROUGH 77461, LEAVING 03752 OCTAL OR 02026 DECIMAL LOCATIONS.								

Figure 14. A typical core storage load generated by the Loader.

deck follows a program deck, it is executed after the program has been loaded, thus providing a "load-and-go" operation.

The Loader can accept symbolic patches to individual binary object decks. Patches must immediately follow the intended subroutine, and facility is included for increasing its length if necessary. The patches may be in either fixed or floating point decimal or in octal form. Relocation will be performed if specified. Figure 15 shows a card which will insert a TRA \*+3 at location 220<sub>8</sub> within a subroutine.

*Debug*

Undoubtedly one of the most powerful features of JPTRAJ is Debug. It offers the user a means of obtaining selective core snapshots in any mode and at any desired frequency, and does not decrease the amount of core storage provided a program. The debug cards are placed in a source deck much like data for the intended program. They do not accompany the binary deck, so the program need not be reloaded if snaps are desired.

Debugging is done on the basis of locations relative to a particular subroutine. A typical debug card is illustrated in Fig. 16. The card may be interpreted from left to right to read: in subroutine A, place the snap at location 41. Print in floating point mode the contents of subroutine B, relative locations 155, through 267. This is shown schematically in Fig. 17. Whenever the instruction on the dotted line is executed by the computer, the hatched area is printed in floating point mode.

The debug cards are processed by the Compiler, with the aid of the core storage map generated by the Loader. This map cross-references subroutine entry names and absolute storage

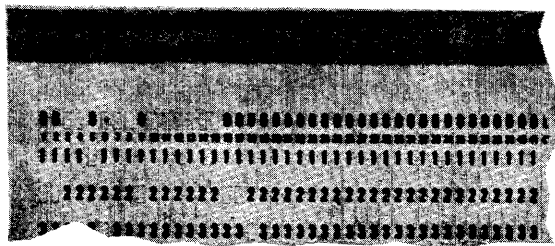


Figure 15. A symbolic patch for a subroutine binary deck.

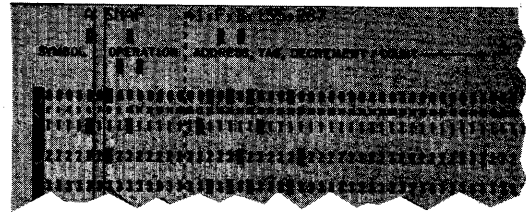


Figure 16. A typical debug card.

addresses. The Compiler translates the debug cards into part of the object string and overlays them in the program after it is read into core storage. The debug instructions are not put on the disk and do not affect a program's permanent operation.

As many as twenty debug cards may appear in a source deck. The modes of snapping on any card include floating point and octal with or without instruction mnemonics. The contents of the panel (all registers and indicators) may be included on option.

The one restriction on the use of Debug is that it may be used during checkout only. The software necessary for real-time data processing does not provide sufficient space for the debug machinery.

III. CONCLUSION

The New JPL Trajectory Monitor has proved to be an invaluable asset to the Laboratory's

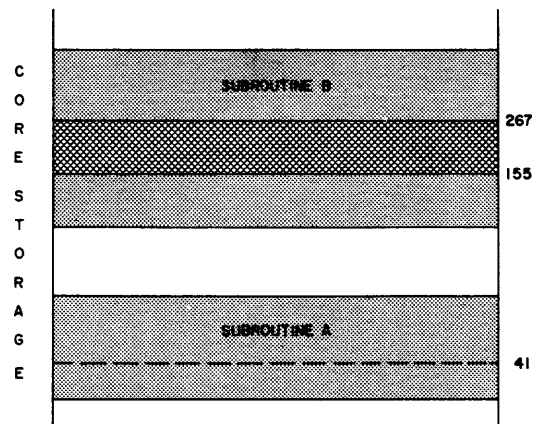


Figure 17. The meaning of the debug card shown in Figure 16.

space flight operations. It has been working quite smoothly since the end of 1963. At the moment, it can operate programs written in Fortran II, Version 3 system only, but modification is under way to interface with the IBJOB programming system. Any program that operates under JPTRAJ may be operated in real-time, so JPTRAJ has, in fact, been JLP's only way to debug mission programs.

It should be evident that nothing limits JPTRAJ to space flight programs alone. The designation "Trajectory Monitor" is of historical origin, and, in fact, many non-trajectory type programs are operational under JPTRAJ. Any IBM 7094 system with disk file storage may use JPTRAJ, and modification will be completed soon which will allow JPTRAJ to operate on a "Direct-Couple" system.

# ACE-S/C ACCEPTANCE CHECKOUT EQUIPMENT

*R. W. Lanzkron*  
NASA

*Manned Spacecraft Center, Houston, Texas*

## INTRODUCTION

Since the beginning of the space age, spacecraft have increased in complexity—progressing from such relatively simple vehicles as the V-2 to the sophisticated Saturn, and from unmanned satellites to the Apollo three-man spacecraft. With the entry of man into the system, the complexity increased an order of magnitude.

Two aspects of these manned systems demand complexity: first, those problems involved in the life-support requirements for each system and, second, the increased requirements for reliability and crew safety. Systems are more sophisticated; at least one backup and usually more are required. The engineer is confronted with the difficult task of insuring that the complex system is ready for launch and that it will meet mission and crew safety requirements.

Final countdown for early missiles took about a day. Although countdown in some of the small, modern missiles has been reduced to one or two hours, the newer and more complex spacecraft still require about a day. Moreover, to insure mission success and crew safety, great quantities of data must be monitored in real time before launch. This was not previously the case when real-time monitoring involved only a small percentage of the instrumentation.

To perform checkout and monitor large numbers of parameters in real time, automatic checkout equipment had to be developed. The Manned Spacecraft Center had two other mo-

tives in developing its automatic checkout equipment. First, due to the complex checkout procedure and the large booster hazard, checkout by remote control became essential. Current plans call for the checkout equipment to be located ten miles from the vehicle. (Automatic checkout lends itself extremely well to remote control.) Second, the cost of the space program has to be reduced. One of the most expensive parts of the space program is its associated Ground Support Equipment (GSE). Therefore, by standardizing the checkout equipment while it is being automated, the period of its use can be prolonged indefinitely. This is one of the basic requirements placed on MSC's automatic checkout equipment or, as it is called, ACE-S/C (Acceptance Checkout Equipment).

The ACE-S/N System has one other unique feature. It combines the capabilities of manual and automatic checkout into one system. This dual capability was incorporated at the request of the system engineers who are still not accustomed to completely automatic systems, and for whom automation is still in the learning process.

## II. ACE-S/C DEFINITIONS

The ACE-S/C System is composed of two main parts: the Spacecraft Unique System (carry-on and facility equipment) and the Ground System (as shown in Figure 1).

The carry-on and facility equipment is associated with the spacecraft, for example, the

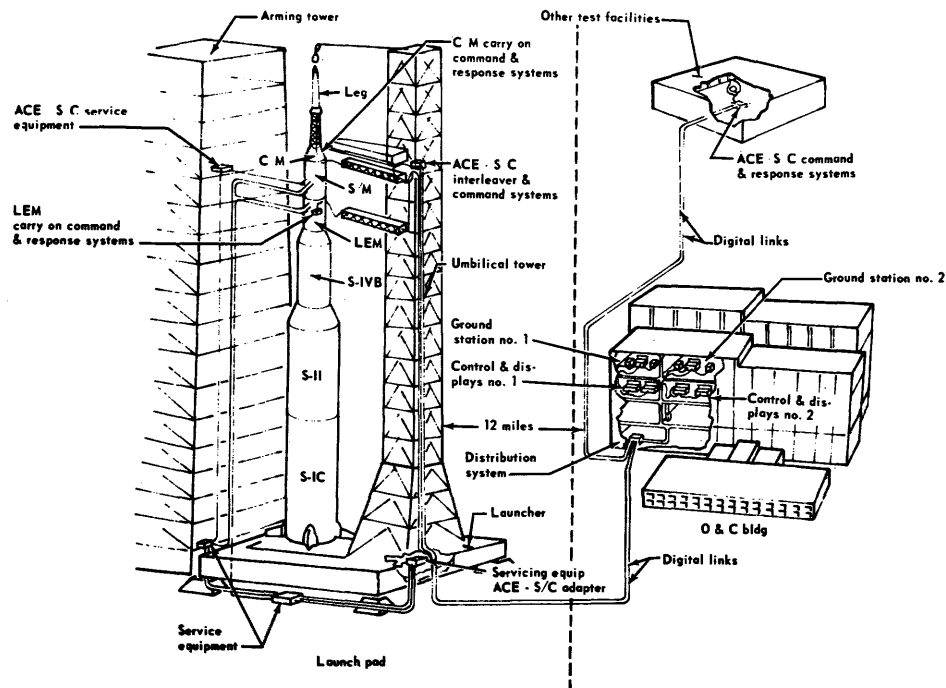


Figure 1. ACE-S/C carry-on and facility

Command and Service Module (C & SM) or the Lunar Excursion Module (LEM). It consists basically of (1) signal-conditioning equipment which conditions signals transmitted from the spacecraft, commutates and digitizes them, and finally converts them into serial pulse form, and (2) the command system, which takes computer outputs and presents them to the spacecraft system. These parts are modular and, by using the building block concept, can be used with any spacecraft or flight vehicle. Signal-conditioning is made part of the ACE-S/C system to minimize fly-away weight. The carry-on equipment is removed from the spacecraft 5 hours before launch so that no scar and tear weight associated with checkout is flown; thus measurements can be taken up to the time the spacecraft is sealed.

The second part, the Ground System, takes the serialized data, decommutates it, and displays it on meters, lights, oscillographs, or alphanumeric displays. The Ground System also takes system engineering inputs and translates them into computer language used for command of the spacecraft. The ground station is the same for both C & SM checkout and LEM checkout.

### III. CARRY-ON AND FACILITY EQUIPMENT

The carry-on equipment is divided into two parts: the Uplink (the Digital Test Command System), and the Downlink (the Digital Test Monitoring System), as shown in Figure 2.

#### A. The Uplink or Digital Test Command System (DTCS)

The DTCS, in turn, consists of two parts: internal and external to the spacecraft. The DTCS consists of a receiver decoder, base plates, and pluggable modules. The pluggable modules are relay and Silicon Controlled Rectifier (SCR) modules and Digital-to-Analog Converters (DAC). The receiver decoder, the first portion of the system to receive data in a serial stream in redundant form, compares the data and checks for correct codes. The receiver decoder stops incorrect information, signals the Ground Stations that it has received erroneous data, and indicates the type of error. Information may be incorrect because of transmission or because of errors in the original data sent to the spacecraft.

From the receiver decoder, the information is routed to the base plate, which stores the power

and the pluggable modules discussed above. Each base plate contains 4 pluggable modules. The system is modular; any module (relay module, SCR, and DAC) can be selected and plugged into the base plate. The receiver decoder can drive 32 base plates which are grouped into 4 groups of 8 plates each. The modules again verify that information received from the receiver decoder is correct and, by looking at the address, that the correct base plate and module have been selected. Notification of incorrect information and the type of error involved is sent to the ground station, which may not then execute a command. Should there be no error, verification is sent to the

ground station, and the System Engineer may then execute the command.

The relay module is used for turning specific signals to the spacecraft on or off. The SCR module is used for high-speed switching and for high current capacity. The DAC modules is used for changing the stream of digital data into an analog signal. This conversion is required, for example, in torquing a Gyro, where a series of digital commands are sent to the vehicle, converted to voltage levels, and in turn filtered so that the output is smoothed and looks like an analog sine wave. The system can transmit data in bursts at a rate of 500 k bits per second. The basic information is

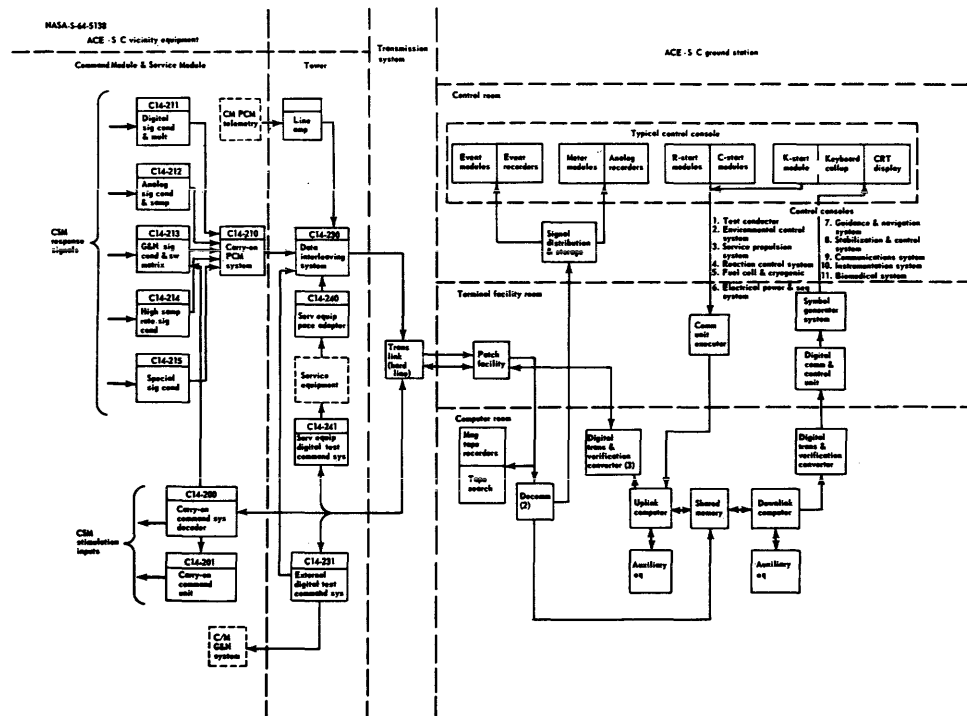


Figure 2. ACE-S C block diagram

grouped into 24-bit messages. Each message consists of two 12-bit redundant words, which are checked at the receiver decoder. It contains the information needed to exercise any one of the modules.

In addition to the base plate and its three types of modules, there is a special unit, the Guidance and Navigation (G & N) Buffer. This unit acts as a buffer between the Ground Digital System and the onboard G & N computer. This unit is specialized in the sense that its output is tailored to a unique digital computer. The G & N buffer unit is used for storage verification and shift out of the G & N digital data. Again, data is verified at the receiver decoder and in the G & N unit to ensure its correctness. Additional precautions have been taken with this unit as the G & N is the heart of the system. The G & N command uses 15 bits of the 24 for transmitting information. These 15 bits represent the character, its complement, and the character again. The internal portion of the DTCS is stored under the left couch in the Command Module as shown in Figure 3 and Figure 4. The function of the DTCS is to exercise the systems aboard the Command Module. This part of the system is removed 5 hours before launch to minimize flight weight. The external DTCS is stored on the Launch Umbilical Tower (as shown in Figures 1 and 2) since its prime function is to exercise the G & N system and the servicing

equipment. Because the G & N has to be exercised until the last moment before launch, this unit operates through the umbilical.

An equivalent type of equipment exists for the system associated with the LEM. In the LEM, most of the DTCS functions are available outside the LEM.

#### B. *The Digital Test Monitoring System (DTMS)*

The DTMS consists of the following parts: the signal-conditioner, the sampling unit, the PCM system, the flight PCM system, and the interleaver. Because minimizing fly-away checkout weight is a requirement, the signal-conditioning package was made part of the carry-on equipment. Again, the package is removed at T-5 hours.

Signal-conditioning consists of converting analog signals from their level to a 0-volt to 5-volt level. It is also used for frequency count, glitch detection (detection of unpredicted spikes), phase comparison, counting, and other required functions.

These signal-conditioners are modular and of the same size. They can be plugged into the main container in any position. If a change in the signal type or valve is needed, a different signal-conditioner can be plugged into the same position in the container. After these signals have been conditioned, they are sampled and

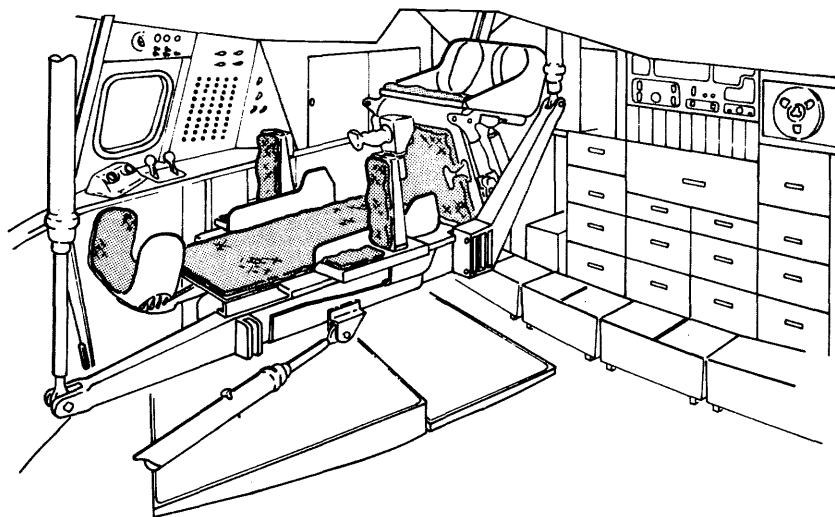


Figure 3. Left hand couch



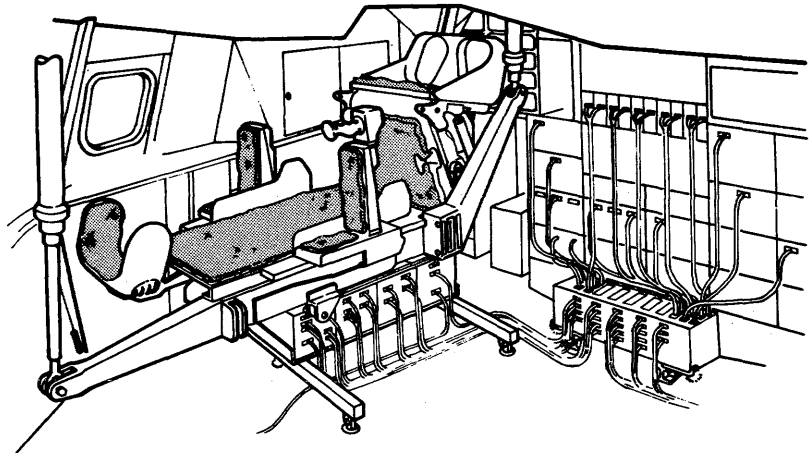


Figure 4. Left hand couch and DTCS

transmitted to a PCM system, which converts them into a serial form. The PCM train, at that point, is at the rate of 51.2 k bits per second. This information is then combined with the information available from the flight PCM which is also 51.2 k bits per second, and the servicing equipment PCM, also at 51.2 k bits per second. Note that the servicing equipment also has associated with it a signal-conditioning and PCM system, which are housed in slightly different containers; the same type of modular units are used inside the spacecraft. The system used to combine the three 51.2 k-bit trains is called the interleaver. The interleaver is set up to receive four streams of inputs, each at a rate of 51.2 k bits per second, to a total stream

of 204.8 k bits. The interleaver thus has a reserve capacity for accepting another 51.2 k bits per second.

The DTMS is divided into two parts: (1) the signal-conditioner, the sampling unit, and the PCM system, which are located in the spacecraft (as shown in Figures 5 and 6), and (2) the interleaver, which is located in the Launch Umbilical Tower (as shown in Figure 1). The interleaver is part of the facility equipment.

Because of its shape, the same arrangement generally holds true for the LEM system. The containers for the signal-conditioners and the PCM systems are slightly different, but the modular pluggable components are the same.

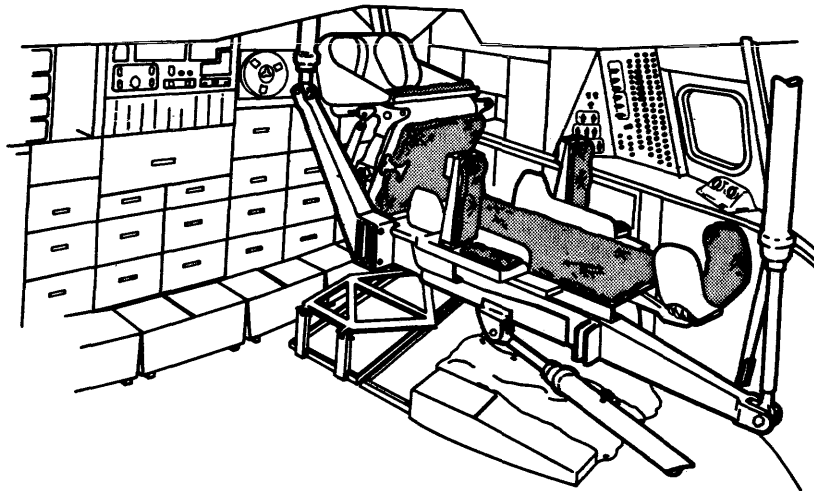


Figure 5. Right hand couch

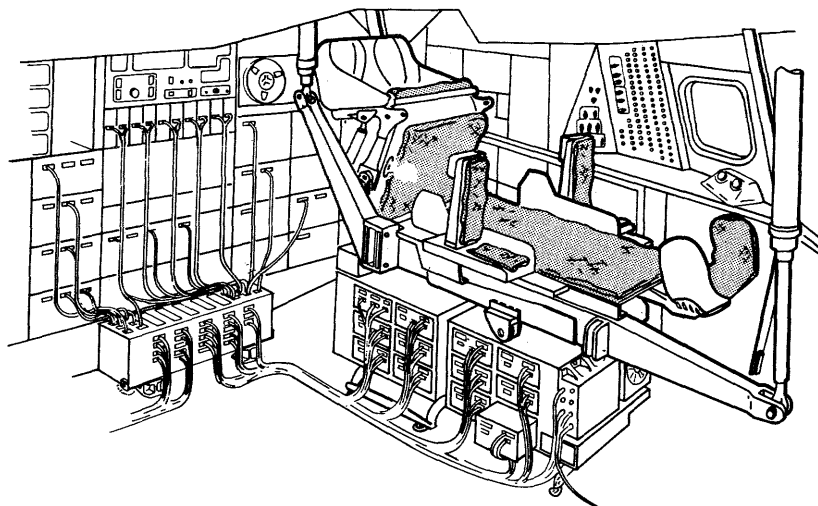


Figure 6. Right hand and DTMS

#### IV. THE GROUND STATION

##### A. General

The Ground Station consists of three rooms: the control room, the computer room, and the terminal room (as shown in the artist's conceptions included as Figures 7, 8 and 9).

The terminal room contains the timing system, the ground portion of the DTCS, and the formatting equipment for the control room.

The computer room contains the computers and associated peripheral equipment, the decommutator, which is used to decommutate the information from the interleaver, and the transmission equipment for the DTCS.

The control room contains the control consoles at which the systems engineers sit to control and command the spacecraft systems as required.

##### B. Downlink

Figure 10 shows the basic flow for the downlink. In the figure, information from the interleaver is directly recorded and at the same time flows into a decommutator. The system has two decommutators which provides redundancy and also allows the airborne telemetry to be routed through one of the decoms and the rest of the information (including the airborne telemetry) through the second decom. From the decommutator, the information

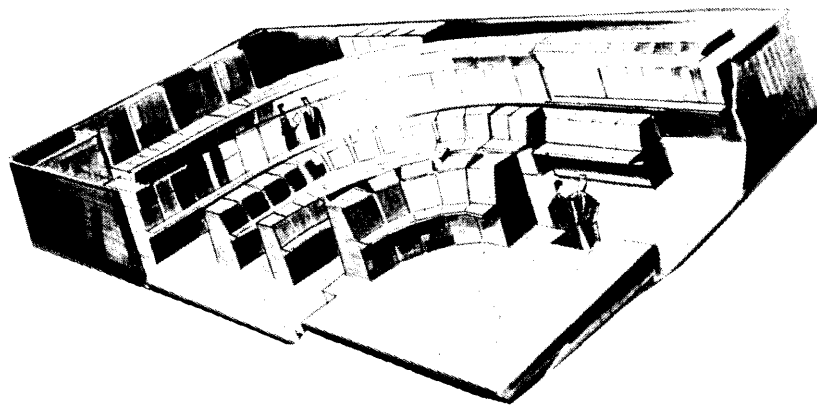


Figure 7. Control Room

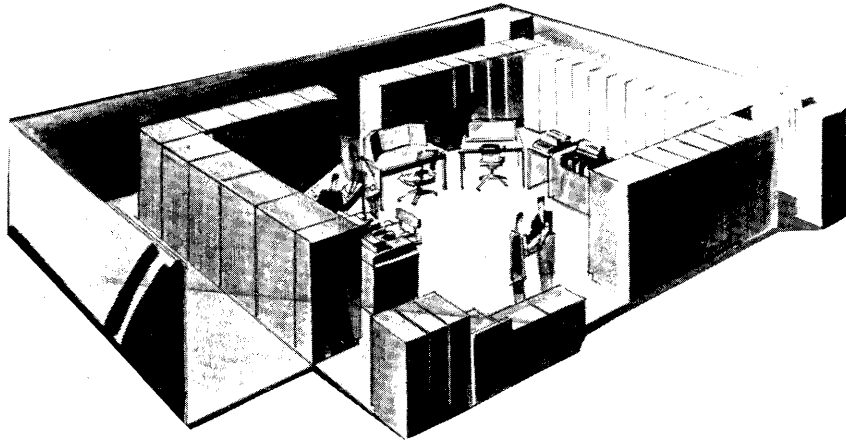


Figure 8. Computer Room

flows to the downlink computer, to event storage and distribution units (ESDU), and to the control consoles. The decommutator can drive the ESDU, which, in turn, drives the event modules (Figure 16) on the control consoles. The decommutator also drives some 200 meter modules (Figure 15) and oscillographs located on the control consoles, thus allowing the engineers to watch some of the measurements on meters and oscillographs as required. Decommutator output presented to the downlink computer is manipulated by introducing the calibration information, comparing it to prestored information, and changing the information into engineering units. The downlink computer then transmits this information to the Symbol Generator and Storage Unit (SGS) via the data transmission and verification converters (DTVC), and the digital communicator control unit (DCCU). (These are transmission lines and verification units for the digital data.)

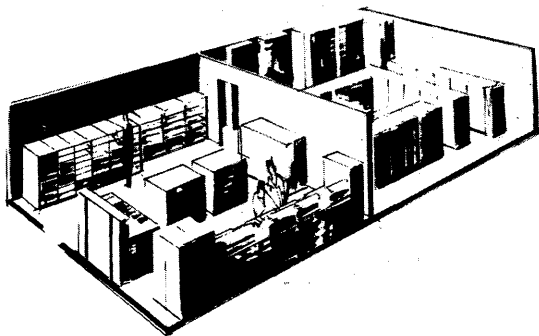


Figure 9. Terminal Facility Room

The SGS transmits information, when requested, to the Cathode Ray Tube (CRT) which displays the alphanumeric information for the ACE-C/C.

On receipt of the coded words from the DCCU, the SGS decodes the 12-bit words, strips the data out, and stores it in memory locations according to received instructions and addresses. The entire content of the SGS memory is updated once per second and is scanned at a rate sufficient to update all alphanumeric displays at least 30 times per second.

The character repertoire of the SGS enables all alphanumeric characters to be displayed together with a set of special symbols. The SGS can also cause characters to blink on the CRT screen when so instructed by the computer. This blinking indicates an out-of-tolerance condition. Memory allocations in the SGS are sufficient to display 20 "pages" of data; a "page" consisting of 24 lines of 40 characters each, plus two 32-character lines, one at the top and one at the bottom of the "page." On demand by the System Engineer, the SGS provides the proper signals to cause the contents of any top half-page and any bottom half-page to appear on the screen of the requesting CRT. Any given half-page can be selected by as many as 20 CRT's simultaneously.

The CRT's utilized for the Alphanumeric Display System are 10-inch electrostatically deflected devices. Besides controls for selecting the desired half-pages, the CRT module has controls for focusing, brightness, horizontal and

vertical centering, and an on/off switch. Any given half-page display contains 12 lines of identified data in decimal numbers and engineering units. In addition, a single top (or bottom) line identifies the page. The CRT module, by a special means, can indicate other pages that have information in them that is out of tolerance. Thus, the display portion of the display console consists of event lights driven by the ESDU, the meters and oscillographs driven directly by the decommutator, and alphanumeric displays driven by the computer.

C. The Uplink

Figure 11 shows the basic flow for the uplink (or command link). The uplink starts with the systems engineers who have the option to exercise R-START, C-START, and K-START. As shown in Figure 12, an R-START module consists of four function switches and an execute (XEQ) switch; Figure 12A shows the R-START panel configuration. Each function switch is a backlighted, split-window, pushbutton switch. The two halves of the window are independently illuminated. Successively pressing a given switch will cause its lower half to alternately illuminate and extinguish. The upper half is illuminated or extinguished under program control by uplink verification as discussed earlier. The execute switch is an independently backlighted, split-legend (XEQ/SEAL), pushbutton switch. The XEQ portion is illuminated when the switch is depressed. The XEQ portion is extinguished, and the SEAL portion illuminated and extinguished, under computer control.

The Systems Engineer sets the desired status of each of the four relays (at the spacecraft) controlled by the R-START module by setting the condition of the lower half of each function switch; illuminated for a latch command and extinguished for an unlatch command.

Execution of the selected commands is initiated by depressing the XEQ switch. This illuminated the XEQ portion of the switch. When an uplink verification is received, indicating delivery of the command message, the XEQ portion will extinguish, and the upper halves of the function switches will illuminate in conformity with the lower halves.

Normally, the data transfer will be so rapid that the XEQ light will appear to remain extinguished. However, status identity of the upper and lower half of each function switch indicates receipt of the uplink verification. It should be noted that the uplink verification only confirms proper delivery of the command message; it does not confirm actual relay latching or unlatching. The Systems Engineer will now observe the proper event display meter module or oscillograph to obtain downlink verification of the relay status.

The SEAL portion of the execute switch illuminates to indicate that the computer has SEALed this particular R-START module, which prevents it from initiating commands. Extinction of the SEAL light indicates that the computer has released the SEAL, thus returning the module to normal operation. The computer will SEAL an R-START module when command transmission difficulty is ex-

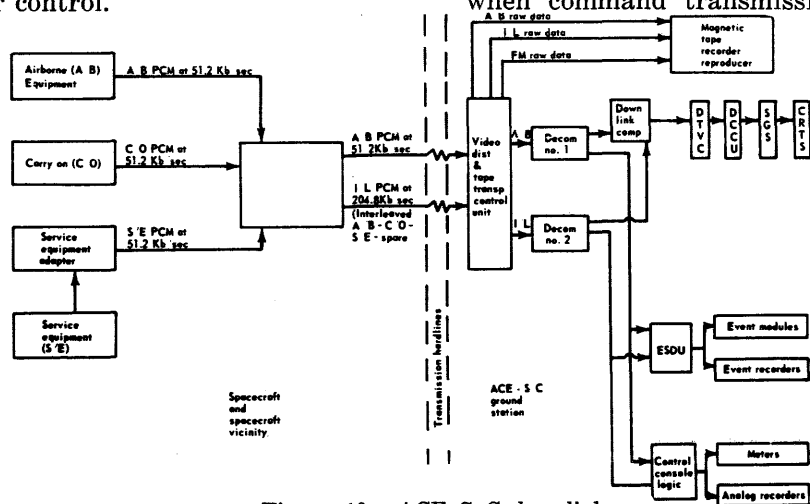


Figure 10. ACE-S C downlink

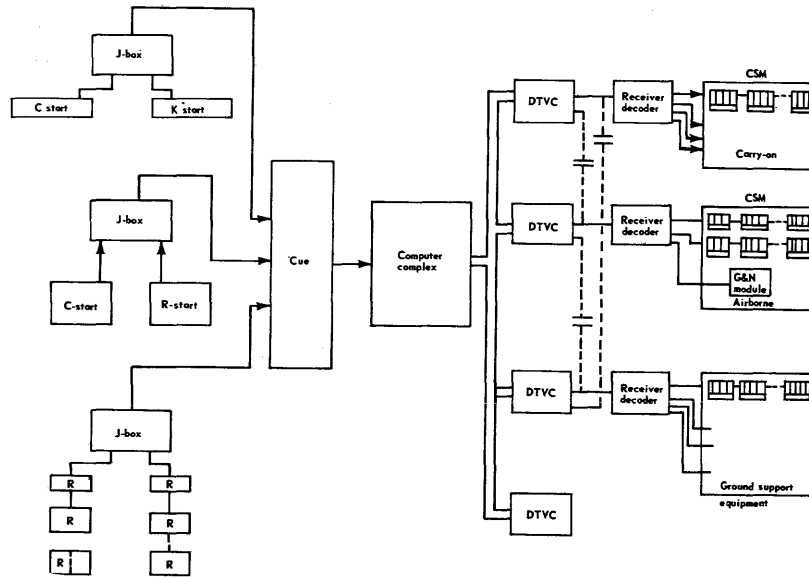


Figure 11. ACE-S C uplink (DTSC)

perienced or when under program control, required by a particular test procedure.

A C-START module panel (see Figure 13) presents ten 12-position rotary selector switches, each having an associated relay display, an execute (XEQ) switch, and an XEQ verify light.

The 12 positions of each selector switch are identified by the 10 decimal digits, 0 through 9, and the two signs (+) and (-). The readout display above each selector switch indicates the switch setting by displaying the appropriate decimal digit or sign. The execute switch is an independently backlighted, split-legend (XEQ/SEAL), pushbutton switch. The XEQ portion

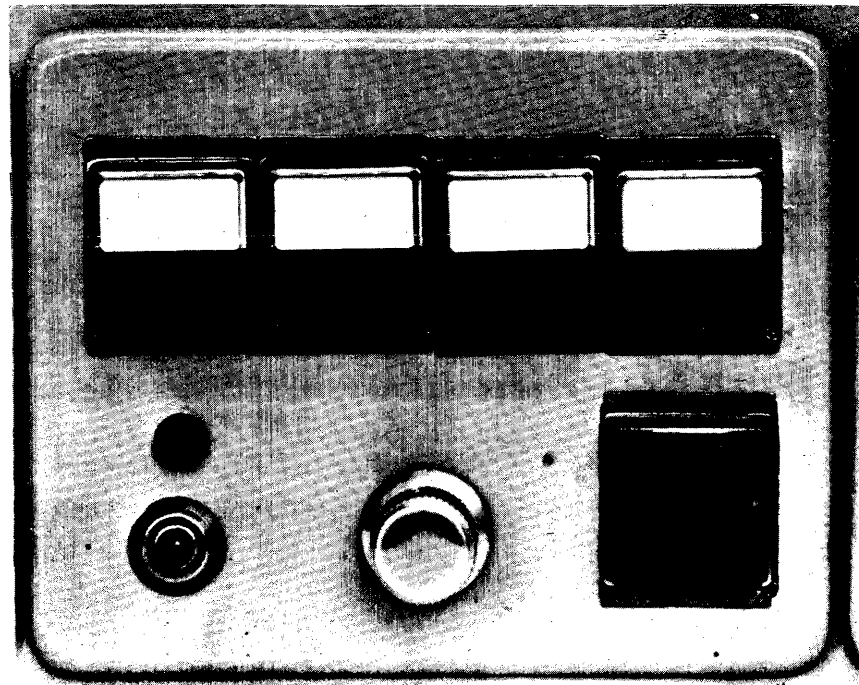


Figure 12. R-START MODULE

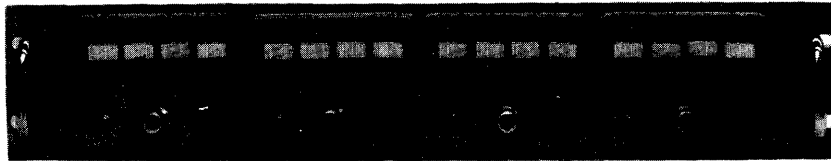


Figure 12a. R—START PANEL CONFIGURATION

is illuminated when the switch is depressed and extinguished, under program control, by the computer following delivery of the command message. Module SEAL is not presently applicable to the C—START, but the SEAL portion of the execute switch has been provided for possible future use.

The execute verify light is illuminated when a valid execute signal is sensed, and remains illuminated for 1 to 1½ seconds.

The Systems Engineer sets up the desired input command to the computer by appropriately positioning each of the ten selector switches. Execution of the selected command is initiated by depressing the XEQ switch. This illuminates the XEQ portion of the switch. The next action will be illumination of the XEQ verify light and extinction of the XEQ light switch. Normally, the process is so rapid that the XEQ switch light will appear to remain extinguished; hence, the extended illumination of the XEQ verify light provides the Systems Engineer with an observable indication of valid execution.

The K—START module panel (shown in Figure 14) consists of a pushbutton keyboard, six tape control switches, a tape input display, three status displays, and a bank of 24 Apollo Guidance Computer (AGC) event displays.

The keyboard contains 18 pushbutton switches, as shown in Figure 14, for manual insertion of binary-coded command messages to the AGC. The associated perforated tape reader, used for automatic insertion of command messages, is controlled by six backlighted, pushbutton switches. Four of the switches

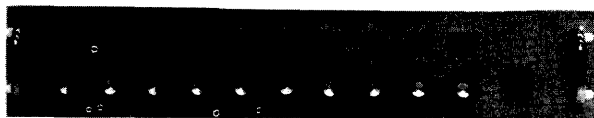


Figure 13. C—START MODULE PANEL

have split legends. Successive depressions of any of these will alternately illuminate one-half and extinguish the other.

The tape input display consists of a row of eight lights, which represent the eight digit positions of a single perforated tape character. This permits the Systems Engineer to visually read a tape character. The illumination of a given light indicates the presence of a binary one (perforation) at that digit position. No illumination indicates a zero. The three status displays operate in both manual and automatic modes as follows:

**XEQ/SEAL:** Upon depression of a keyboard pushbutton or read-in from the tape, the XEQ will light, thus indicating an execute request. Receipt of an uplink verification will extinguish XEQ and illuminate SEAL. No commands can be inserted while the module is SEALED. Receipt of a downlink reply will result in release of the SEAL and extinction of the SEAL display.

**VERIFY:** This is a display only. Its illumination signifies that the downlink reply compared correctly with the character transmitted via the uplink.

**NONVERIFY:** This is a combination display and pushbutton switch. Its illumination signifies that the downlink reply did not compare correctly with the character transmitted via the uplink. In the automatic mode, this

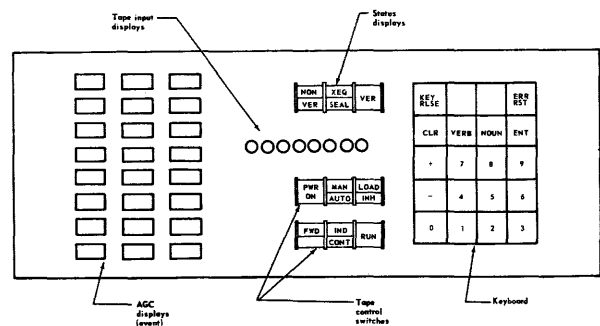


Figure 14. K—START MODULE PANEL

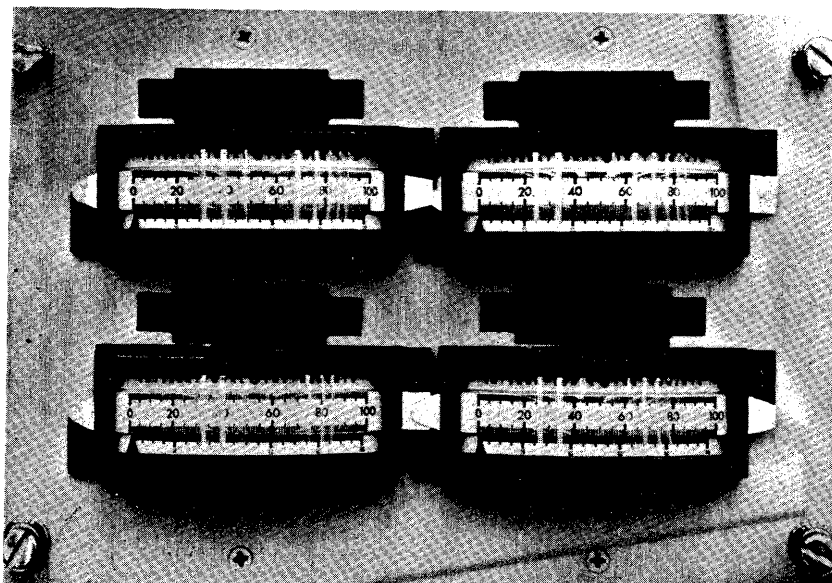


Figure 15. METER MODULE

condition will prevent character entry from the tape. Depression of the NONVERIFL switch will extinguish its light and permit a new attempt at command insertion.

The AGC displays provide the Systems Engineer a convenient indication of 24 discrete events within the Apollo Guidance Computer.

When any of the START modules are depressed, the scanning mechanism checks on the R-STARTS, C-STARTS, and K-STARTS. The scanning device, called the Communication Unit Executor (CUE) notifies the uplink computer as soon as it notices an R-START, C-START,

or K-START. The uplink computer checks the CUE before accepting the information. The CUE compares the address, located in the START module, with what it thinks has been exorcized and transferred. If the comparison is affirmative, the information is transmitted to the computers, which interprets the codes indicated by the R-START, C-START, or K-START, and transmits the information to the spacecraft where the DTCS described earlier checks it and exercises the spacecraft accordingly. The system can interpret some 1200 measurements in the downlink and some 200 in the uplink area.

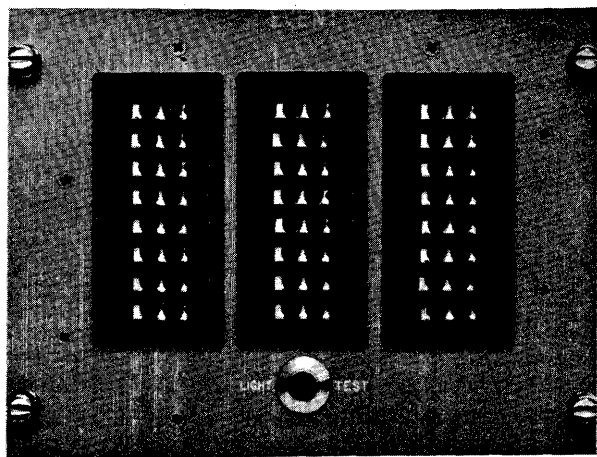


Figure 16. EVENT MODULE

## SUMMARY

The main features of the system are as follows:

1. Rapid checkout
2. Use of the flight telemetry and carry-on concept. The carry-on equipment for checkout measurements does not penalize the fly-away weight.
3. Capability to operate manually as well as automatically at close-by or 10-mile remote positions.
4. Use of the same equipment for checking out both the C & SM and the LEM at both the factory and the launch site.

5. Modularity and digital building blocks to achieve turnaround of about 30 minutes. This allows hook-up to any spacecraft on 30 minutes' notice, assuming that the carry-on equipment is in position.
6. Factory connection of carry-on equipment and removal at T-5 hours (except for weight and balance). This minimizes connecting and disconnecting the spacecraft cabling.
7. Restriction of the total number of cables through the vehicle hatch to two, each containing three co-axes, thus allowing undisturbed ingress and egress to the spacecraft. (Two cables in Apollo compare to about two dozens in Mercury.

The system is quite versatile—so versatile, in fact, that the vehicle on which it is to be exercised does not have to be defined until late in the design.

Selecting the signal-conditioners and the modules associated with the base plates, and mounting the name plates associated with the

control consoles are the only time-limiting actions. Once the system engineers have established a general configuration of the control room the SGS, the decommutator, and the computers are programmable. Thus, after hardware definition, the only problem remaining is to define the digital programs or the software associated with the system. But these can be phased in as soon as the test people and the system engineers have decided the test detail. Preliminary specification shows that the hardware design and preliminary software require something on the order of 8 to 10 months, whereas the final software (because the preliminary subroutines and executive routines have been prepared) can be completed within two or three months.

The first system is operational at North American's plant in Downey, California. Follow-on stations will be located at the Grumman Aircraft Engineering Corporation, Bethpage, L.I., New York; the Manned Spacecraft Center, Houston, Texas; and the Kennedy Space Center, Cape Kennedy, Florida.



# SATURN V LAUNCH VEHICLE

## DIGITAL COMPUTER AND DATA ADAPTER

*M. M. Dickinson*

*J. B. Jackson*

*G. C. Randa*

*International Business Machines Space Guidance Center  
Owego, New York*

### INTRODUCTION

This paper describes the IBM Space Guidance Center's part in the Saturn V Program and the digital computer and data adapter being developed for the Saturn V booster. This work is being performed under contract to NASA under direction of the Marshall Space Flight Center, Huntsville, Alabama.

The computer and data adapter are located in the Saturn V Instrument Unit and integrated into the total guidance system of the booster (Figure 1). The computer interfaces only with the data adapter, which in turn presents the interface to the rest of the system. Basically, during boost guidance, the computer evaluates in-flight changes in booster speed and position derived from an inertial platform and develops signals to control the rocket engines so as to keep the booster on course. The data adapter takes analog inputs from sensors and converts them to digital form for the computer; it also takes the computer digital outputs, converts some of them to analog form, and sends corrections to the appropriate controls.

### MODES OF OPERATION

The system operates in four basic modes (Figure 2): (1) Pre-launch Check-out; (2) Boost Guidance; (3) Orbital Check-out; and (4) Lunar Trajectory Injection.

During the Pre-launch Check-out mode, a test program is loaded into the computer to ensure that all guidance system interfaces operate properly prior to flight. The program includes a computer self-test, complete mission simulation, and a system test, among others.

In the Boost Guidance mode, which starts at lift-off and lasts until the final booster stage burns out, the computer navigates and steers the booster, computing stage cutoff. During this initial phase of operation it receives data on booster speed and attitude through the data adapter. The computer processes these data and, through the data adapter, controls the

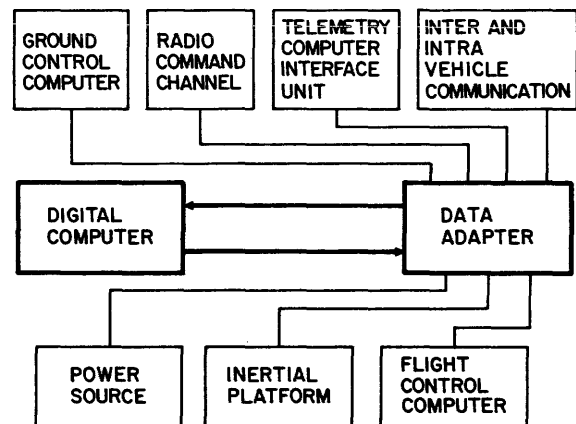


Figure 1. Saturn V Guidance System.

- PRE-LAUNCH CHECKOUT
  - PROGRAM CHECK
  - COEFFICIENT LOAD
- BOOST GUIDANCE
  - STEERING COMMANDS
  - ENGINE CONTROL
- ORBITAL CHECKOUT
  - INSTRUMENT UNIT CHECKOUT
- LUNAR TRAJECTORY INJECTION
  - STEERING COMMANDS
  - ENGINE CONTROL

Figure 2. Saturn V Mission.

direction of thrust of the gimballed rocket engines to keep the vehicle on the desired course.

When the vehicle is in orbit about the earth, the computer checks out the propulsion system, the mid-course guidance and control system, and other related Instrument Unit systems and sends the test results to the ground for analysis. If all tests are satisfactory, the Lunar Trajectory Injection mode is initiated.

The Lunar Trajectory Injection mode follows the same sequence as the Boost Guidance mode in that the computer navigates, controls vehicle steering, controls booster cut-off, and directs booster separation.

**EQUIPMENT ORGANIZATION AND CHARACTERISTICS**

Microminiature packaging technology and redundant logical configurations are used in the computer and data adapter to meet the stringent weight and reliability requirements of the Saturn V Program. Whereas the Saturn I launch vehicle guidance system was developed from a previously proved military system to play a limited role on early space booster missions, the Saturn V equipment must have much greater sophistication to function with greatly increased reliability over a much longer operational span in a more demanding environment (Figure 3). Thus, the Saturn V computer and data adapter share new electronic and mechanical design features that provide significantly greater capabilities than are found in the

	SATURN I	SATURN V
NUMBER OF COMPONENTS	12,000	80,000
WEIGHT - LBS.	210	253
VOLUME - CU. FT.	3.9	5.5
POWER - WATTS	540	438
OPERATIONS CAPACITY- OPS/SEC	3,200	9,600
STORAGE CAPACITY - BITS	100,000	460,000
RELIABILITY - MEAN TIME BETWEEN FAILURE IN HOURS	750	45,000

Figure 3. Equipment Comparisons.

Saturn I spaceborne computer system, including:

- (1) A speed approximately three times greater than is found in either the Saturn I or IBM's commercial 1401 system.
- (2) A modular memory that can be expanded to 920,000 bits for later Saturn V missions by simply plugging in additional memory modules.
- (3) Seven times the number of components with only a slight increase in weight and volume.
- (4) A reliability increase of more than 6000 percent.

**COMPUTER**

The Saturn V computer is serially organized and operates at a rate of 512 kilobits per second (Figure 4). Using two's complement arithmetic, it multiplies four bits at a time and divides two bits at a time. Glass delay lines

TYPE	GR STORED PROGRAM, SERIAL, FIXED POINT, BINARY
CLOCK	512 KILOBITS PER SECOND
SPEED	ADD-SUBTRACT, MULTIPLY-DIVIDE SIMULTANEOUSLY:
	ADD - 82 μSEC, 26 BIT
	MULTIPLY - 328 μSEC, 24 BIT
	DIVIDE - 656 μSEC, 24 BIT
STORAGE	16,384 28-BIT WORDS, EXPANDABLE
RELIABILITY	0.99 GOAL FOR 250 HOURS
WEIGHT	77 LBS.
VOLUME	2.2 CU. FT.
POWER	131 WATTS

Figure 4. Digital Computer Characteristics.

are used for internal registers to improve reliability. AND-OR-INVERT logic, operated in saturated and cut-off modes, is employed in both the computer and data adapter. Four clock pulses per bit are employed, with 6-volt clock signals applied to AND resistors; this arrangement permits inverter loads to be time-shared and obviates the need for an AND diode for each clocked AND.

Each instruction is comprised of a four-bit operation code and a nine-bit operand address; the nine-bit address allows 512 locations to be directly addressed. The memory is divided into sectors of 256 words, and contains a residual memory for 256 common data words. The nine-bit address specifies a location in either the previously selected sector (data sector latches) or in the residual memory.

Instructions are addressed from an eight-bit instruction counter augmented by a four-bit instruction sector register (Figure 5). Instruction memory sector selection is changed by special instructions, but sector size is sufficiently large that this is not a frequent operation.

Data words consist of 26 bits (25 magnitude bits plus sign). Instruction words consist of 13 bits and two instructions are stored in each

memory data word. Hence, instructions are described as being stored in syllable 0 or syllable 1 of a memory word. Two additional bits are used in the memory for parity checking each of the two syllables.

The computer is programmed by means of single-address instructions. Each instruction specifies an operation and an operand address. Instructions are addressed sequentially from the memory under control of the instruction counter. Each time the instruction counter is used, it is advanced one increment to develop the address of the next instruction. After the instruction is read from the memory and its parity checked, the operation code is sent from the transfer register to the operation code register, a static register that stores the operation code for the duration of the execution cycle.

The operand address portion of the instruction is transferred in parallel (nine bits) from the transfer register to the memory address register. The transfer register is then cleared.

If the operation code requires reading the memory, the contents of the operand address are read, 14 bits at a time (including parity), from the memory into the buffer register where

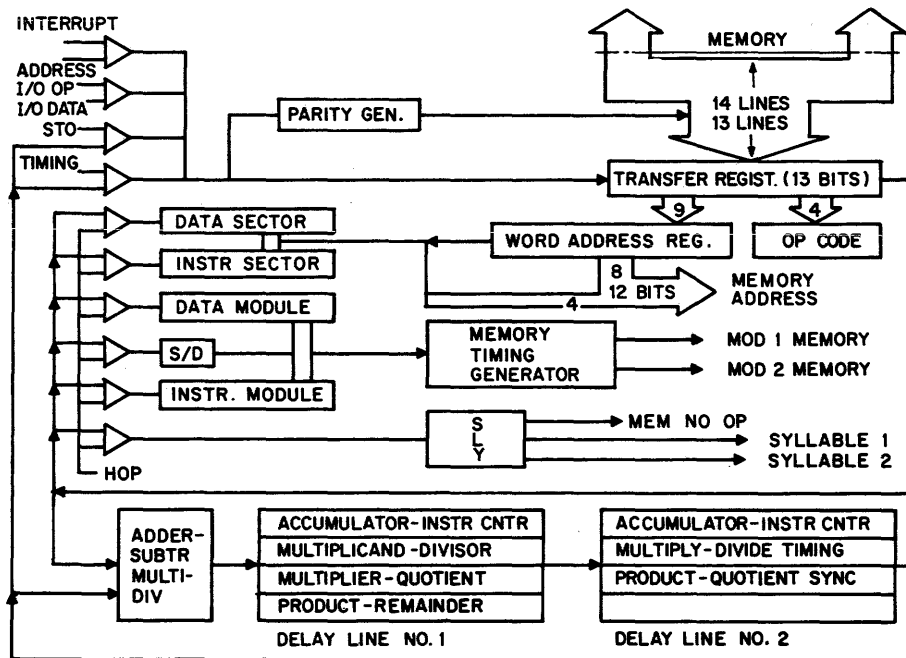


Figure 5. Digital Computer Block Diagram.

a parity check is made. Data bits are then sent in parallel to the transfer register. This information is then serially transferred to the arithmetic section of the computer. If the operation code is a store (STO), the contents of the accumulator are transferred serially into the transfer register and stored in two 14-bit bytes. A parity bit is generated for each byte.

Upon completion of the arithmetic operation, the contents of the instruction counter are transferred serially into the transfer register. This information is then transferred in parallel (just as the operand address had previously been transferred) into the memory address register. The transfer register is then cleared and the next instruction is read, thus completing one computer cycle.

The data word is read from the memory address specified by the memory address register and from the sector specified by the sector register. Data from the memory go directly to the arithmetic section of the computer where they are operated on as directed by the operation code.

The arithmetic section contains an add-subtract element, a multiply-divide element, and storage registers for the operands. Registers are required for the accumulator, product, quotient, multiplicand, multiplier, positive remainder, and negative remainder. The add-subtract and the multiply-divide elements operate independently of each other, so they can be programmed to operate concurrently (i.e., the add-subtract element can perform several short operations while the multiply-divide element is in operation).

No dividend register is shown in Figure 5 because it is considered to be the first remainder. The divisor is read from the accumulator during the first cycle time and can be regenerated from the two remainders on subsequent cycles. Both multiply and divide require more time for execution than the rest of the computer operations. A special counter is used to keep track of the multiply-divide progress and stop the operation when it is completed. The product-quotient (PQ) register can be addressed from the operand address of any instruction. The answer will remain in the PQ register until another multiply-divide is initiated.

CLA	CLEAR AND ADD	CDS	CHANGE DATA LOCATION
ADD	ADD	EXM	EXECUTE MODIFIED INST.
SUB	SUBTRACT	STO	STORE
MPY	MULTIPLY (CONCURRENT)	HOP	CHANGE REGISTER CONTENTS
MPH	MULTIPLY (NON-CONCURRENT)	TNZ	TRANSFER NON-ZERO
DIV	DIVIDE	TRA	UNCONDITIONAL TRANSFER
RSU	REVERSE SUBTRACT	TMI	TRANSFER ON MINUS
AND	LOGICAL AND	PIO	PROCESS INPUT/OUTPUT
XOR	EXCLUSIVE OR		
SHF	SHIFT		

Figure 6. Operation Codes.

Two multiply instructions (Figure 6) are used in the computer. MPY requires that one-word-time operations be performed in the adder unit during the multiplication operation because the instruction counter advances each word time. Thus, simultaneous MPY and one-word-time operations are carried out. When the program is multiply-limited, and a sufficient number of useful one-word operations cannot be found in the portion of the flow diagram being executed, the MPH instruction is used. This instruction inhibits the advance of the instruction counter and no new instructions are read from the memory until the operation is completed.

Since only partial addresses for instructions and data are provided in the instruction counter and the data address of the instruction word, a HOP instruction permits transfer to various sections of the memory; static registers provide the additional address bits required. An Execute Modified Instruction (EXM) operation permits the execution of selected instructions from memory after they have been modified by the contents of the instruction word.

The computer communicates with the data adapter through use of the Process Input-Output (PIO) instruction, which transfers data words from the data adapter to the accumulator or from the accumulator or memory to the data adapter.

The computer processes accelerometer and gimbal angle information and computes attitude corrections 25 times each second. The slower major loop, which contains navigation and other computations, is performed once or twice each second.

A program assembler and a mission simulator are being developed as part of a software package. Laboratory check-out and acceptance test programs are also being prepared. IBM is using special simulators to perform delay simulation on detailed logical designs, to compute the reliability of redundant and duplex logical organizations using Monte Carlo techniques, and to simulate the effects of component malfunctions. Operational flight programs for Saturn V missions are being prepared by IBM under a separate contract.

### DATA ADAPTER

The data adapter provides input-output communication with telemetry equipment, ground launch computer, and discrete inputs and outputs (Figure 7). It stores interrupt signals and buffers real-time, accelerometer data monitoring, and other counting signals. Angular information from two-speed resolvers is read in through a crossover detector system (Figure 8). Resolver outputs are passed through RC networks and the zero crossover of the phase-shifted sine wave outputs is detected by precision crossover detectors. A high-speed binary counter converts this time interval into a binary number. Analog voltage outputs are provided through a resistive ladder network and suitable capacitor sample-and-hold circuits.

A significant feature of the data adapter design is the use of glass ultrasonic delay lines to store digital information required by the

CLOCK	512 KILOBITS PER SECOND
DISCRETES	68 BITS
REGISTERS	45 BITS
D/A CONVERTER	8 BITS + SIGN
A/D CONVERTER	2-SPEED RESOLVER, 17-BIT EQUIVALENT
DELAY LINES	16 CHANNELS
POWER SUPPLIES	6 PAIRS, DUPLEXED
RELIABILITY	0.99 GOAL FOR 250 HOURS
WEIGHT	176 LBS.
VOLUME	3.30 CU. FT.
POWER	438 WATTS INPUT

Figure 7. Data Adapter Characteristics.

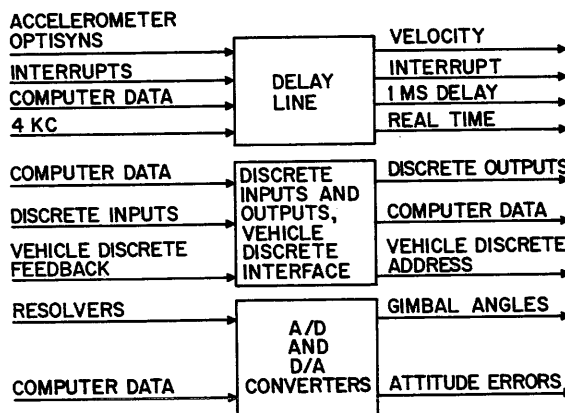


Figure 8. Data Adapter Block Diagram.

computer. Four-megacycle lines are used and four channels of information are multiplexed into each line.

A set of redundant delay lines is divided into three 14-bit phase times, providing 12 syllables of storage in each line. This set is used to store: (1) interrupt signals until the interrupts are acted upon by the computer, with one of the syllables being reserved to prohibit multiple computer interrupts from the same external interrupt pulse; (2) interrupt signals which are inhibited by the computer. The set also: (1) times the activation of the switch selector used to control the vehicle staging; (2) times the interval between processing of minor loop platform attitude inputs; and (3) stores real time as it is accumulated from an oscillator input.

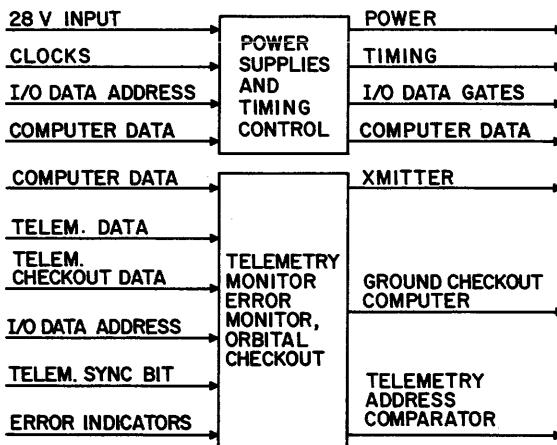


Figure 9. Data Adapter Block Diagram (Cont.).

A non-redundant delay line is used to buffer non-mission-critical telemetry information to the ground (Figure 9). The PCM telemetry system will monitor digital data from the computer/data adapter system at a constant rate of 240 forty-bit words per second. Approximately 100 of these words will be direct outputs from the computer to the telemetry. The remaining 140 words will be supplied by the Digital Output Monitor (DOM). This data adapter system serves two purposes: (1) it allows the telemetry channel to be used with maximum efficiency without burdening the computer program with intricate timing problems, and (2) provides a means of monitoring the digital data entering and leaving the data adapter at a point close to the interface.

The information that is sent to telemetry via the DOM is derived from computer input-output operations. Since these operations occur at widely varying rates, the DOM provides a buffering capability to optimize data flow. Input-output operations that occur too frequently and convey too little information are ignored by the DOM.

## RELIABILITY

Very high reliability is required of the Saturn V digital computer and data adapter—the design goal for each unit is 0.99 for a 250-hour mission. Conventional techniques for achieving ultra-high reliability include (Figure 10) high reliability parts, conservative design practices (simplicity and wide tolerances), 100-percent screening of parts and assemblies, thorough qualification of parts and manufacturing processes, and detailed laboratory analysis and corrective action for all “failed” items. Over and above these conventional requirements, the equipment has been designed to continue accurate operation, even after a tran-

### CONSERVATIVE DESIGN REDUNDANCY COMPONENT SCREENING IN-PROCESS INSPECTION SYSTEM ENVIRONMENTAL ACCEPTANCE

Figure 10. Approaches to Reliability.

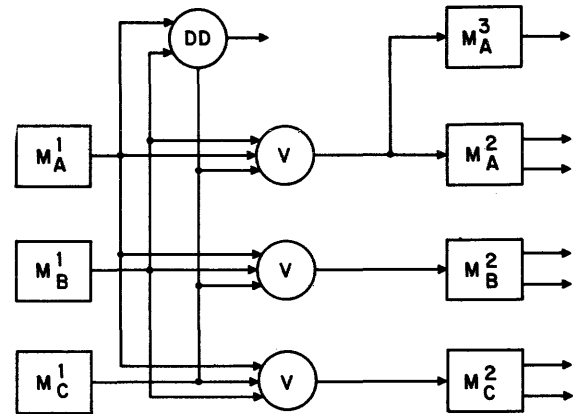


Figure 11. Triple Modular Redundancy (TMR).

sient or catastrophic failure, through the use of redundant elements.

The redundancy approach employed by IBM in the computer logic is “Triple Modular Redundancy” (TMR), which realizes a 20-fold increase in reliability with only  $3\frac{1}{2}$  times more components than a non-redundant system.<sup>(1,2,3,4)</sup> Figure 11 shows part of the Saturn V logic divided into sections called modules, (M). Each module is identical, receiving the same problem at the same time. The outputs of the three modules are transmitted to another circuit called a “majority rule voter” circuit (V), which checks the inputs to see whether they agree. If one input differs from the other two, it is disregarded, so a component failure will not cause a system malfunction.

A third circuit, called a disagreement detector (DD), monitors system performance by signalling the ground equipment whenever voter inputs are not all identical. Computer logic is divided into seven modules, each with an average of 10 voted outputs. Disagreement detector outputs are OR’ed together such that malfunctions can be isolated to one, two, or three replaceable subassemblies.

The memory in the Saturn V computer uses conventional toroidal cores in a unique self-correcting duplex system. The memory consists of up to eight identical 4096-word memory modules that may be operated in simplex for increased storage capability or in duplex pairs for high reliability. The basic computer program can be loaded into the instruction and constants sectors of the memory at electronic speeds on the ground or just prior to launch.

Thereafter, the information content of constants and data can be electrically altered but only under control of the computer program.

The self-correcting duplex system uses an odd parity bit for malfunction indication and correction. In conjunction with the parity bit, error-detection circuitry also monitors memory drive current. Unlike conventional toroid random-access memories, the self-correcting extension of the basic duplex approach permits correct information to be regenerated after transients or intermittent failures.

Figure 12 is a simplified block diagram of the computer memory system. The basic configuration consists of a pair of memories providing storage for 8,192 fourteen-bit memory words for duplex operation, or 16,384 fourteen-bit memory words for simplex operation. Each of the simplex memories includes independent peripheral instrumentation consisting of timing, control, address drivers, inhibit drivers, sense amplifiers, error-detection circuitry, and input/output connections to facilitate failure isolation.

The computer functions, which are separate for each simplex memory, consist of synchronizing gates which provide the serial data rate of 512 kilobits per second. This data rate is required by the computer to generate a "start memory unit" command at 128 kilobits per second. These gates also provide the selection of multiple simplex memory units for storage flexibility and permit partial or total duplex operation throughout the mission profile to extend the mean-time-before-failure for long mis-

sion times. Each of the simplex units can operate independently of the others or in a duplex manner. The memory modules are divided into two groups: one group consisting of even numbered modules (0-6) and the other consisting of odd numbered modules (1-7). The buffer register associated with each group is set by the selected modules.

For duplex operation, as shown in Figure 12, each memory is controlled by independent buffer registers when both memories are operating without failure. Both memories are simultaneously read and updated, 14 bits in parallel. A single cycle is required for reading instructions (13 bits plus 1 parity bit per instruction word). Two memory cycles are required for reading and updating data (26 bits plus 2 parity bits). The parallel outputs of the memory buffer registers are serialized at a 512-kilobit rate by the memory transfer register under control of the memory select logic. Initially, only one buffer register output is used, but both buffer register outputs are simultaneously parity checked. When an error is detected in the memory being used, operation immediately transfers to the other memory. Both memories are then regenerated by the buffer register of the "good" memory, thus correcting transient errors. After the parity-checking and error-detection circuits have verified that the erroneous memory has been corrected, each memory is again controlled by its own buffer register. Operation is not transferred to the previously erroneous memory until the "good" memory develops its first error. Consequently, instantaneous switching from one memory output to another permits uninterrupted computer operation until simultaneous failures at the same storage location in both memories cause complete system failure.

The data adapter uses duplex and redundant-component circuits in conjunction with TMR logic to meet its reliability goals.

The data adapter logic is divided into four parts for the purpose of reliability computation. The analog-to-digital converter section employs coarse and fine resolver windings in a two-speed system for duplex operation. The fine inputs are selected by the computer for use unless the error indicator circuits that monitor

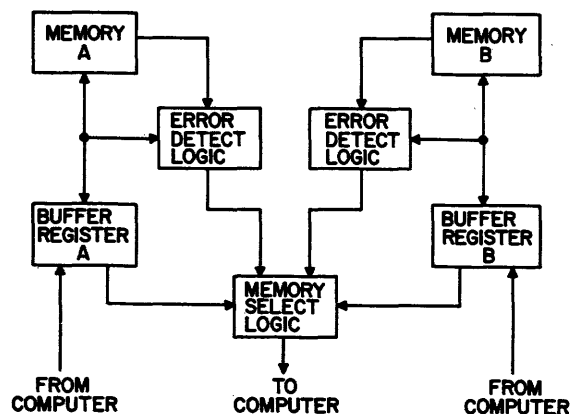


Figure 12. Duplex Memory Flow Diagram.

signal levels indicate a failure; then the coarse input will be used, with degraded system accuracy. The selected input is fed to duplexed logic sections that can be tested against each other and for output reasonableness. Here the computational capability of the computer can insert test problems into the duplex logic of the data adapter to determine which half is operating properly.

Duplex digital-to-analog ladder converters are used for outputs to the control computer. One of the duplex channels is selected to derive the output voltages, and this output is compared with a reference channel. Any significant difference causes the other duplex channel to be switched in.

TMR logic is used in conventional buffer registers that store discrete information and provide communication with telemetry and the ground control computer. Where TMR outputs must be used to develop a single highly reliable output signal, redundant-component voter/driver circuits are used in the final stage.

To ensure that failures in the power supplies, which are located in the data adapter and supply power to that unit and to the computer, do not cause system failure, duplexing is used (Figure 13). Pulse-width-modulated dc-to-dc converters provide high power conversion efficiency. Feedback amplifiers sense any variations in the average value of the output voltage,

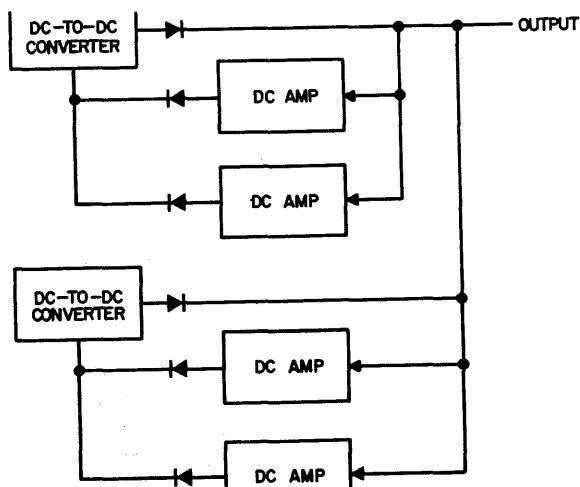


Figure 13. Duplex Power Supply.

and the error signal is used to control the power inverter pulse width. Duplexed feedback amplifiers are employed and the supplies themselves are duplexed. Either supply can provide the full current required for that supply voltage.

Table I compares the reliability of the redundant and non-redundant portions of the computer and data adapter and shows the dramatic results obtained when TMR techniques are used in the control and arithmetic logic of the computer. In applications where the logic could be divided into modular parts and made duplex, it would be difficult to determine which duplex part was operating correctly. TMR, with majority rule voters re-establishing the correctness of module outputs at critical points, overcomes this difficulty and makes malfunction correction automatic.

The improvement in memory reliability derived from duplexing is not as great as in other duplex logic applications because all possible memory failures cannot be detected by the checking circuits. As shown in Table I, overall computer reliability in the redundant configuration is gated by the memory system.

TABLE I. EQUIPMENT RELIABILITY FOR 250 HOURS

	Non-Redundant	Redundant
Digital Computer		
TMR Logic	0.973	0.9992
Duplex Memory	0.969	0.9980
Unit	0.943	0.9972
Equivalent Mean		
Time to		
Failure	4,250 Hrs.	89,500 Hrs.
Data Adapter		
Duplex I/O	0.988	0.9982
Duplex Logic	0.994	0.99995
TMR Logic	0.979	0.9997
Duplex Power		
Supply	0.995	0.99999
Unit	0.957	0.9978
Equivalent Mean		
Time to		
Failure	5,680 Hrs.	113,000 Hrs.



The mean-time-to-system-failure (MTF) shown in Table I for a non-redundant equivalent computer was computed assuming the conventional exponential (constant failure rate) case.

### MICROMINIATURE PACKAGING

To implement redundancy and still meet NASA's weight and volume requirements, IBM chose a microminiature design approach<sup>(5)</sup> proven feasible in 1960 with a small-scale computer. Since 1961, the technology has been subjected to environmental, life, and experimental system tests. These tests have enabled IBM to accumulate over 20,000,000 test hours on circuit modules and over 1,000,000 on multilayer interconnection boards.

The first simplex engineering model of the Saturn V computer was delivered on 28 April of this year; it was used to confirm and verify the basic design of the computer. The first qualification model is scheduled for delivery on 12 February 1965 and the first production computer on 26 March.

The basic building block of the Saturn V system is the Unit Logic Device (ULD), a microminiature circuit package. Each of the 54 different types of ULD's contains up to 14 components including transistors, diodes, and resistors. Ninety percent of the Saturn V equipment is based on ULD technology, and 8918 ULD's are used in each Saturn V computer and data adapter.

The manufacture of a ULD starts with a  $0.3 \times 0.3$  inch alumina substrate upon which conductive land patterns are silk-screened on the top surface and resistors on the bottom (Figure 14). Conductive patterns and resistors are fired at high temperature to drive off the carrier and ensure adhesion to the substrate. Semiconductors, in the form of  $0.025 \times 0.025$  inch chips of silicon (each comprising either one transistor or two diodes) are reflow soldered to the ULD. Copper balls are used for contacts between the chips and the conductive patterns (Figure 15).

Screened edge connections provide conductive lines between the top and bottom surfaces of the ULD. Metal "S"-clips are soldered around

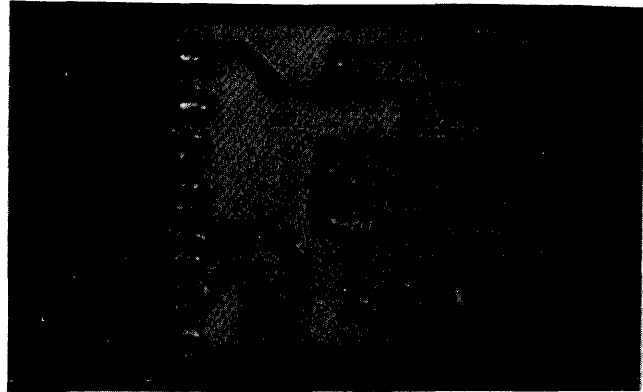


Figure 14. Unit Logic Device (ULD).

the edges of the ULD for greater reliability and to provide a means of connection to the printed circuit board that interconnects the ULD's.

Each ULD has been designed so the parameters of its components may be measured during production operations. This procedure ensures that component tolerances assumed for initial circuit design are met and that end-of-life tolerances will not be exceeded. Thus, when the values of each parameter are known, circuit design ground rules can be adjusted to permit greater fan-in and fan-out than when the circuit is tested as a whole and individual component values are not known.

The ULD's are interconnected by a MIB, or multilayer interconnection board (Figure 16). A 12-layer, 2.5 by 3-inch board can accommodate 35 ULD circuits. MIB's are laid out to provide a  $5 \times 7$  matrix of ULD locations. Signal, power, and ground layers are contained within the MIB and interconnected with plated-through holes. Tabs printed on the top surface

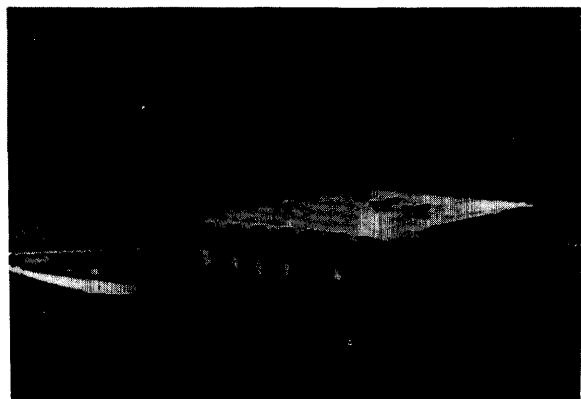


Figure 15. ULD Semiconductor Chips.

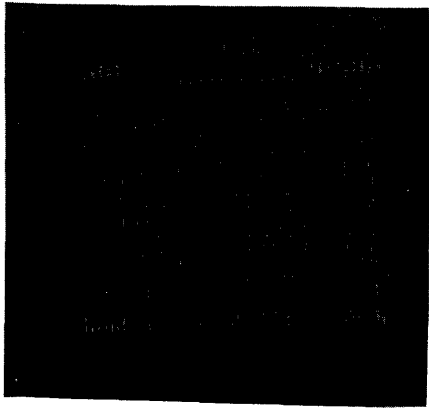


Figure 16. Multilayer Interconnection Board (MIB).

of the MIB permit reflow soldering of the ULD's to the MIB. Two beams of infrared are focused on the pre-tinned ULD connection clips to solder the ULD's to the MIB, affording both an electrical path and a mechanical bond.

Eighteen test points are available on each MIB; connections between MIB's bonded to each side of a page are provided with through-pins. A conductive pattern on the end of each MIB is soldered to a 98-pin connector. The page MIB's containing the ULD's are, in turn, bonded to a metal frame that draws heat away from the components and protects the circuits from vibration and mechanical shock (Figure 17).

The resultant page assembly contains up to 35 ULD's on each side. The page can also house various other types of circuits, including glass delay lines, circuit modules for precision analog circuits, or decoupling capacitors (Figures 18, 19, and 20). There are 150 ULD pages in the Saturn V computer and data

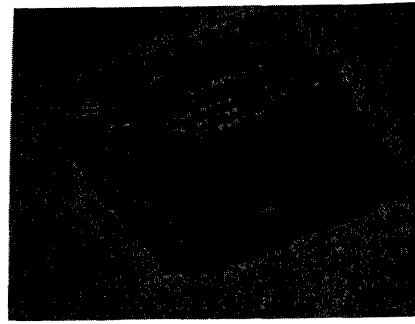


Figure 18. Delay Line Page.

adapter, each with an average of 500 components. Each page is tested over a 10°C to 100°C temperature range during production.

Maintenance performed at the Marshall Space Flight Center or Cape Kennedy includes removal and replacement of individual pages. ULD's can be replaced at the factory or depot. A 98-pin miniature connector plugs into a back panel for page-to-page interconnections. Heat is conducted out of the page through magnesium-lithium ears fastened to the machine structure with screws.

In the Saturn V equipment, pages are interconnected through multilayer printed circuit back panels bonded to a metal support plate (Figure 21). Layers are interconnected by 2000 plated-through holes. Each back panel, containing 12 layers of circuits, is subjected to 100 percent inspection of each of the 2000 connections. Each of the back panel subassemblies is called a channel, and there are five channels in each Saturn V computer (three identical



Figure 17. ULD Page.



Figure 19. Circuit Module Page.

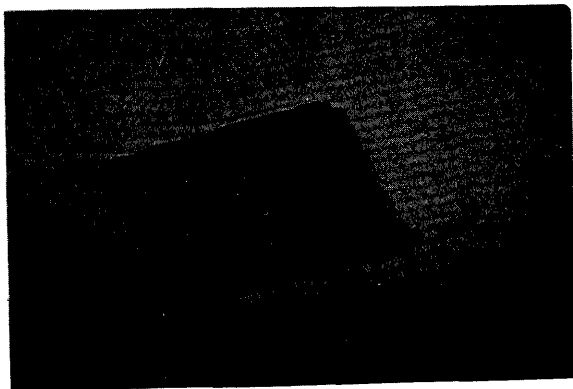


Figure 20. Capacitor Page.

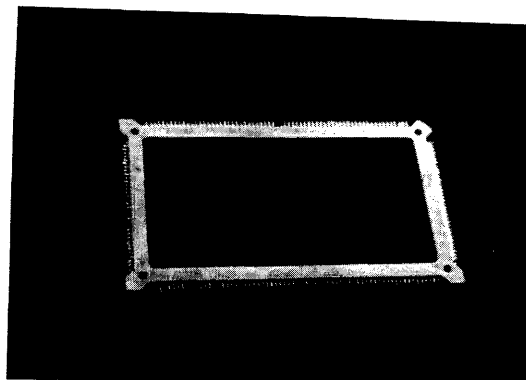


Figure 22. Memory Core Plane.

TMR logic channels and two channels containing voter and memory input-output registers).

The memory plane is the basic element of the Saturn V memory (Figure 22). Each plane accommodates some 8,200 toroid cores approximately 0.030 inch in diameter. Memory planes are individually tested using automatic plane test equipment before further assembly. The memory array is comprised of 14 horizontally stacked planes (Figure 23). It has a capacity of 4,096 twenty-eight-bit words or 115,000 bits of memory. Memory planes are unencapsulated and are separated by molded silicone rubber foam pads that protect the cores from vibration stresses.

A complete memory module combines the array and the electronics. From one to eight of these modules can be employed in the Saturn V system; present memory requirements call for four of these modules, each weighing 4.7 pounds and using 7.5 watts of power.

Electronic memory panels contain address, inhibit, and sense circuits to permit reading and writing as well as addressing. The five panels in the system are fabricated separately from the memory for ease in packaging memory-related electronics.

#### STRUCTURAL DESIGN

Magnesium-lithium alloy LA 141 was selected for use in the computer and data adapter structures because it is the most efficient material from a stiffness-to-weight standpoint (short of beryllium) and because of its good vibration damping characteristics.<sup>(6)</sup> Beryllium was not selected because of anticipated problems in fabrication. LA 141 provides minimum weight, and it minimizes the transmission of mechanical vibration between the unit structural mounting pads and the electronic subassemblies within the structure. Unit stress and physical properties of LA 141 are compared to those of other magnesium and aluminum alloys in Table II,

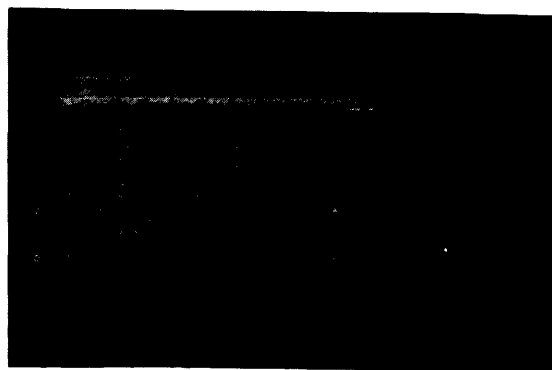


Figure 21. Multilayer Back Panel.

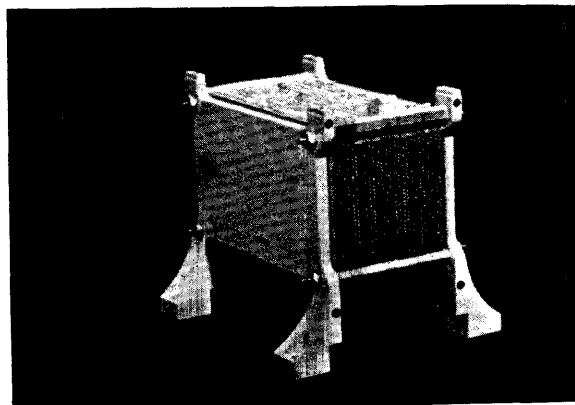


Figure 23. Memory Array.

TABLE II. COMPARISON OF VARIOUS STRUCTURAL MATERIALS

Properties	Material			
	Aluminum Al 1100-0	Aluminum Al 2024-T3	Mag.-Lith. LA 141	Magnesium AZ31B-0
Tensile Strength (Ksi)	11.0	68.0	21.0	36.0
Yield Strength (Ksi)	3.5	51.0	17.0	19.0
Compressive Strength (Ksi)	4.0	42.0	17.0	13.0
Elongation % in 2"	30.0	12-15	25-30	18
Modulus of Elasticity (10 <sup>6</sup> psi)	10.0	10.5	6.5	6.5
Density (lbs./in <sup>3</sup> )	0.098	0.100	0.0487	0.064
Specific Heat at 212°F (BTU/lb/°F)	0.214	0.23	0.35	0.25
Thermal Exp. Coef. (68-212°F, 10 <sup>-6</sup> in/in/°F)	13.1	12.6	21	15
Thermal Conductivity (BTU/ft <sup>2</sup> /hr/°F/ft)	128	71	25	44

below. Magnesium-lithium is only half the weight of aluminum and three quarters of the weight of magnesium. Tensile strength more than meets the requirements for this application.

The choice of magnesium-lithium necessitated a thorough investigation of the effects of humidity and salt atmosphere environments on the material, as well as the effects of liquid coolant. A fluoride-anodize surface treatment, coated with a sprayed coat of Laminar X500 (a polyurethane), was specified to protect all exterior surfaces.

Evaluation of the effects of the liquid coolant (methanol and distilled water) on the magnesium-lithium indicated that the corrosion rate of untreated material did not exceed 0.006 inch/surface/year as long as there are no other materials in contact with the magnesium-lithium that would form a galvanic couple (i.e., material less positive on the galvanic scale).

To effect an RFI seal between the unit structures and their covers, EZ33A welding rod is used on the flanged areas of the structure. The iridite finish used provides adequate protection to inhibit LA-141 oxidation in the presence of nitrogen and moisture. After oxidation, the LA 141 becomes non-conductive.

Cold plate mounting and integral liquid cooling were compared in terms of their effects

on installed weight and reliability. It was found that the installed weight of the system could be reduced by approximately 20 pounds, and the unit component temperatures reduced an average of approximately 18°F (with the accompanying increase in reliability) by employing integral liquid cooling. Thus, integral liquid cooling passages were incorporated in the structure designs (Figure 24).

The incorporation of integral liquid coolant passages in the structure and the need for sufficient material on each side of the coolant passages to assure meeting leak rate requirements at a burst pressure of 100 psia dictated the use of relatively heavy structural sections in the page attachment and outside walls of the structure. Taking advantage of the page

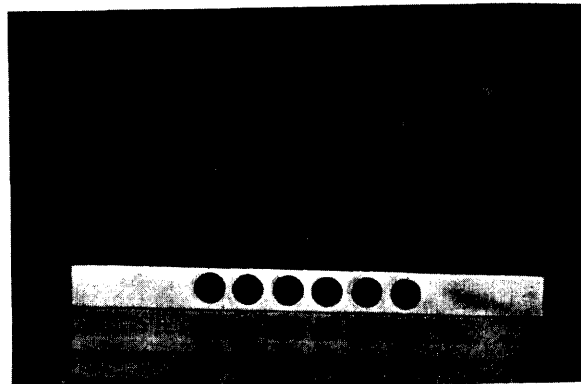


Figure 24. Gun-Drilled Coolant Passages.

attachment coolant wall sections, slots are incorporated at each page location in the structure design to guide the pages into their electrical connector receptacles on the interconnection back panels. This feature eliminates the need for additional brackets to guide the pages.

The above design considerations lead IBM to select a machined billet to fabricate the desired structure configuration. A structural weldment was avoided because of anticipated problems and fixture complexity inherent in controlling dimensions throughout the welding, stress relieving, and machining processes.

The computer structure, as shown in Figure 25, is rough machined from a cast billet, the cooling passages in the page mounting walls are drilled using a technique developed for the drilling of rifle barrels, and final machining is accomplished on a tape-controlled milling machine. Mockups of the computer and data adapter structures are shown in Figures 26 and 27.

#### THERMAL DESIGN

The inlet liquid coolant (60%-40% methanol-water by weight) temperature is between 55°F and 65°F with a total flow rate of 5.5 pounds/minute; 2.2 pounds/minute to the computer and 3.3 pounds/minute to the data adapter. The electrical heat dissipation of the computer is 131 watts and the maximum dissipation of the data adapter is 343 watts with discretes "on."

Tests were conducted on several cooling passage configurations, including finned and drilled passages. The configuration selected has five drilled 3/16-inch diameter holes in each

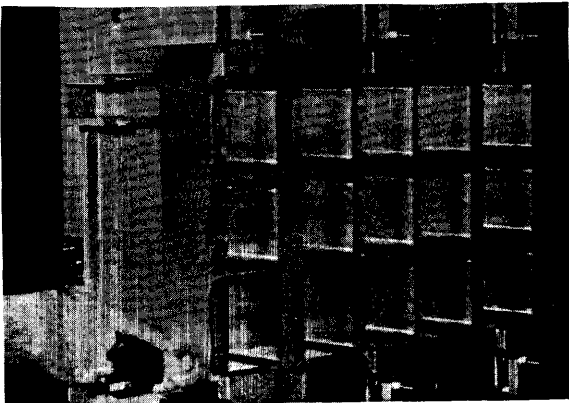


Figure 25. Computer Structure.

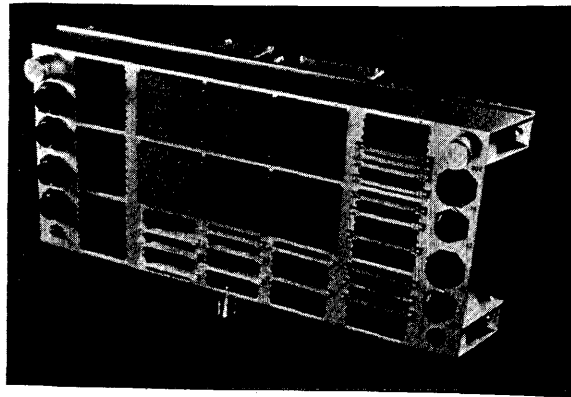


Figure 26. Computer Mock-Up.

computer channel and one 5/16-inch diameter hole in each data adapter channel.

Heat transfer analyses within the units were analyzed using an IBM 7090 three-dimensional heat-transfer program.<sup>(7)</sup> This program uses a numerical approach with rectangular nodes to mathematically represent all solid portions of the unit. The nodal description includes both physical and thermal properties and a definition of the boundary condition on each face. The node faces or boundaries can be defined to have any combination of common heat-transfer processes such as conduction, radiation, and convection.

The thermal design ground rules for the logic electronics were as follows:

Max. transistor junction temperature—	100°C
Max. junction to junction $\Delta T$ on a page—	20°C
Max. page-to-page $\Delta T$ (in a simplex channel)—	50°C

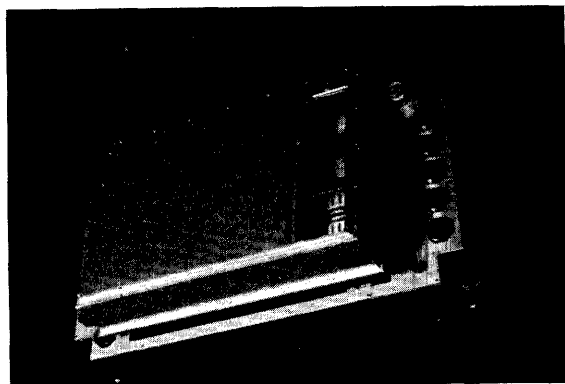


Figure 27. Data Adapter Mock-Up.

The following additional ground rules were imposed on the memory array and power supply electronics:

Max. allowable memory array temperature—	70°C
Max. allowable memory array $\Delta T$ —	5°C
Max. allowable power supply power transistor surface temperature—	85°C
Max. allowable power supply diode surface temperature—	120°C
Max. allowable power supply transformer temperature—	110°C

Initial IBM 7090 computer analysis of the structure indicated that the following maximum temperatures and/or temperature gradients could be expected:

	Computer	Data Adapter
Max. logic transistor junction temperature—	70°C	45°C
Max. junction-to-junction $\Delta T$ on a page—	7°C	7°C
Max. page-to-page $\Delta T$ (in a simplex channel)—	11°C	10°C
Max. memory array temperature—	39°C	—
Max. memory array $\Delta T$	1°C	—
Max. power transistor temperature—	—	100°C
Max. power diode temperature—	—	56°C
Max. transformer temperature—	—	66°C

To verify temperature gradients used in the thermal analysis across thermal interfaces

within the units, and to establish mounting-bolt torques and surface finishes to maintain predictable temperature gradients at these interfaces, thermal-vacuum tests were conducted on the major unit subassembly mockups.

#### VIBRATION TESTING

A computer structure was fabricated from final machining drawings and assembled with dummy subassemblies to simulate final unit weight and center of gravity. This mock-up computer was instrumented with accelerometers to monitor vibration response at the various subassembly mounting points on the structure. The unit was subjected to a 5g RMS sinusoidal vibration environment (2g RMS above acceptance vibration level) and a 0.057  $g^2$ /cps random vibration environment (qualification level environment) with both inputs applied along all three orthogonal axes over a frequency range of 20 to 2000 cps.

The maximum vibration transmissibility measured at three points on the four-memory structure with a 5g RMS input is shown in Table III below.

#### LABORATORY TEST EQUIPMENT

IBM is developing special-purpose laboratory test equipment to perform system-level test functions. This equipment, termed "ASTECS", (Figure 28) will contain 125,000 component parts. Equipment will be located at the IBM Space Guidance Center, Marshall Space Flight Center, and Cape Kennedy. The ASTEC is used to test the computer and data adapter and for computer operational program check-out. It simulates the portion of the Instrument Unit that interfaces with the data adapter.

TABLE III MAXIMUM MEASURED VIBRATION TRANSMISSIBILITIES

Direction	Input (g RMS)	Natural Frequency (cps)	Memory Mounting Plate at Center of Four Memories	Middle Center of Structure	Middle End of Structure
Missile Flight	5	235	2.25	5.6	2.35
Perpendicular To Missile Circumference	5	192	8.06	6.2	2.27
Parallel to Missile Circumference	5	500	1.0	1.83	1.43

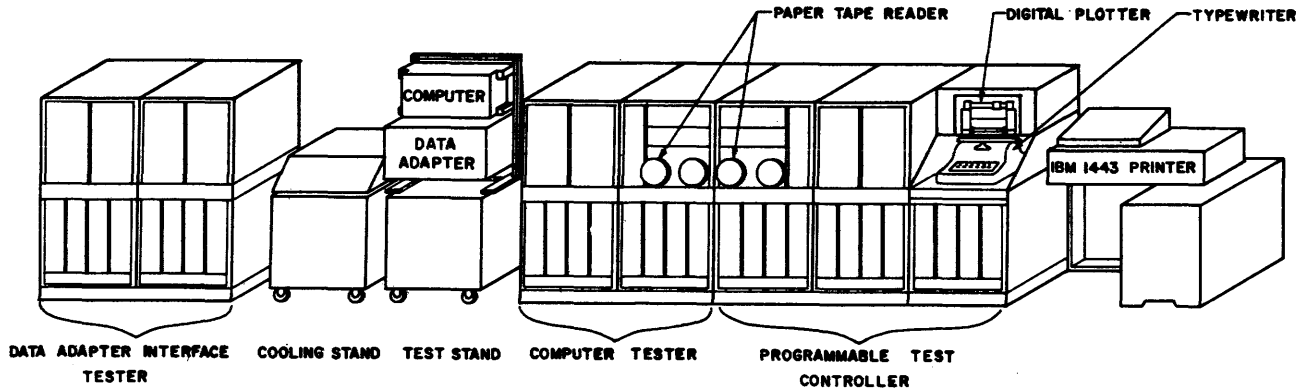


Figure 28. Computer/Data Adapter Laboratory Test Equipment (ASTEC).

ASTEC contains a programmable test controller (PTC), with an 8000-word magnetic core memory. To record test results and minimize human error, a 600-line-per-minute printer is used. All of the equipment employs the same Standard Module System (SMS) electronic packaging techniques used in many types of IBM commercial equipment.

The ASTEC can test the computer and data adapter separately or as a system. It displays significant register results and has a "history file" in delay line storage for analysis of intermittent malfunctions. Disagreement detector outputs are also displayed. The front panels contain controls for power, channel and module switching, single-step program and data adapter control, interrupt features, tape reader for memory loading, and a digital plotter.

The ASTEC may be used for program check-out, static testing, or dynamic testing of the prime equipment under environmental test conditions.

Test stands, for mounting the computer and data adapter undergoing test contain a refrigerator and a heating unit for maintaining proper coolant temperature.

The PTC programs will control the testing of the flight equipment while it is undergoing vibration and other environmental tests. Analog and digital test signals are applied to equipment interface lines under program control, and responses are evaluated by the ASTEC by comparing flight equipment outputs with computed or pre-stored values. Significant com-

puter and data adapter registers are brought out as test points to facilitate testing.

When the data adapter is tested alone the ASTEC will excite inputs and interrogate outputs to test all logic within the unit. Addressing of registers as well as data paths are tested. In testing the computer, diagnostic and test programs are run on the computer in conjunction with PTC programs. Computer self-test programs are designed to exercise the great majority of diodes in the logic to locate intermittent and catastrophic failures under actual environmental conditions.

The ASTEC contains provisions for operating the computer and data adapter in single-step mode, permitting the execution of one program step at a time. Flight operational program check-out is thus facilitated.

The ASTEC is so designed that it may be divided into two parts for use in debugging and testing the computer and data adapter separately in factory operations. Program economy has been achieved by providing common factory and field test equipment.

The computer and data adapter will be used in simulation facilities at the Marshall Space Flight Center to simulate operation of various parts of the instrument unit. The ASTEC, with its inherent programmable capability, may be connected to the data adapter to simulate equipment not actually a part of the simulation experiment. This will provide a very flexible facility for verification of system design prior to installation into the Saturn V launch vehicle.

## REFERENCES

1. DICKINSON, W. E. and WALKER, R. M., "Reliability improvement by the use of multiple-element switching circuits," IBM Journal of Research and Development, 2, No. 2, pp 142-147, (April 1958)
2. FLEHINGER, B. J., "Reliability improvement through redundancy at various system levels," IBM Journal of Research and Development, 2, No. 2, pp 148-158, (April 1958)
3. LYONS, R. E. and VANDERKULK, W., "The use of triple modular redundancy to improve computer reliability," IBM Journal of Research and Development," 6, No. 2, pp 200-209, (April 1962)
4. ERGOTT, H. L. and ROZENBERG, D. P., "On the analysis of reliability improvement through redundancy," Proc. of the Spaceborne Computer Engineering Conference, (October 30-31, 1962)
5. BONIS, S. A., "An improved microminiature circuit packaging technique," Proc. of the Third Annual Microelectronics Symposium of the St. Louis Section of the IEEE, (April 1964)
6. BONIS, S. A., JACKSON, R. N., and PAGNANI, B. R., "Mechanical and electronic packaging for a launch vehicle guidance computer," Proc. of the Fifth International Electronic Circuit Packaging Symposium, (August 21-24, 1964)
7. LALLIER, K. W. and PAGNANI, B. R., "A three-dimensional heat transfer computer program for aerospace applications," IBM Space Guidance Center, Owego, New York, IBM No.: 64-825-862A, (March 1964)



# THE 4102-S SPACE TRACK PROGRAM

*E. T. Garner and J. Oseas  
Radio Corporation of America  
Moorestown, New Jersey*

## SPACE TRACK MISSION

There are five functions which the *SPACE* Track *SE*nsor *C*omputer (SPASEC) real time program performs as part of an effective satellite surveillance system. It locates new satellites soon after launching; keeps up to date records of known satellite orbits; provides positional data of high accuracy for use by other systems; provides information on object size, shape and stability; indicates orbital change. The radar associated with this system can operate in both a surveillance mode to provide data on orbiting objects passing through the volume scanned or in a tracking mode to provide more accurate orbital data on observed objects. The computer program senses, identifies and discriminates among the objects penetrating the surveillance volume and then gathers the required data. This information then undergoes evaluation and the results are forwarded to the Spacetrack Center for additional analysis and correlation.

The mission of the satellite surveillance system is to identify and acquire track or scan data on the satellites. Care is exercised in the gathering of data so that the minimum amount of information is collected that will satisfy the requirements of the particular satellite observation. This insures that the radar will be available to gather information on other objects and that redundant information will not be transmitted. The information leaves the computer as smoothed radar observations in the form of time, range, azimuth, elevation, range rate and identification of the satellite, if

known. This information is transmitted over Teletype lines to the Spacetrack Center.

In addition to the real time aspects of the mission the site must produce its own schedule of what it is able to see (penetration listings). The site is furnished with orbital elements by the Spacetrack Center on all known satellites. This list of known satellites is constantly being updated by means of Teletype messages. A list of desired observations (priority list) is also furnished by the Center.

Figure 1 shows the Operational System Organization. The Operations Director has the responsibility of coordinating the data requirements with the detailed problems of site operation. Missions are planned based on new information requests, object priority, number of expected sightings, and probability of detection. Changes in an object's status are entered into the computer using simple mnemonic codes and decimal numbers. Changes or additions to the satellite file received over the Teletype lines are automatically entered into the computer ephemeris file and written on the master tape. The expected satellite penetrations are computed periodically and a hard copy is produced. This allows an operational check against the master plan based on the most current information at all times.

The plan is executed by the Tracking Radar Control Console (TRCC) operator who exercises gross manual control over the radar system directing the radar to scan the selected sector. Under these manual constraints the sys-

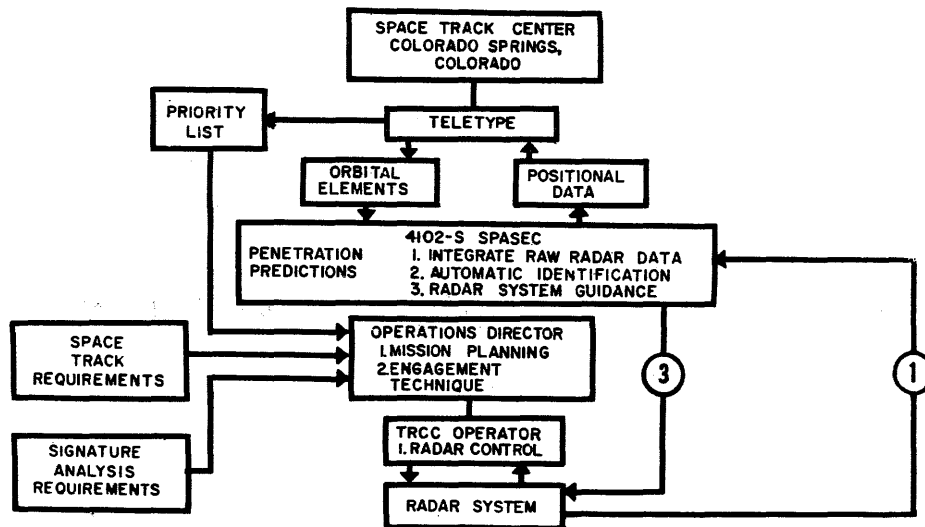


Figure 1. Operational System Organization.

tem is automatic and is controlled by the 4102-S computer.

The raw data collected by the radar system is processed and filtered by the computer. Data identification, data quality and data consistency checks are performed by the computer. The program directs the radar in the gathering of additional information and returns it to the surveillance mode of operation when it is satisfied that no more useful data can be collected. The TRCC operator then executes the next phase of the plan.

The data flow in the operational program is as follows (see Figure 2). As the radar scans its assigned region of space, parameters of objects detected by the radar are stored in the computer. These reports are compared on a scan to scan to scan basis and are said to associate if they might reasonably be expected to have come from a single object. The associated reports are combined and smoothed into an approximation of target position and velocity called a Q-point. When the program determines that no additional useful scan data is available an attempt is made to identify the object. Recognition is made by comparing the Q-point with predicted radar fan penetrations of known satellites. The orbital elements of known satellites are stored in the computer memory. Radar fan penetration or S-points are computed in advance based on the radar scan

coverage and are stored in time ordered sequence. When a Q-point is identified its parameters are transmitted to the Central Data and Cataloging Center via the Teletype link. If no positive identification can be made the Q-point is then subjected to discrimination tests to determine if this object is a satellite. Q-points failing discrimination are discarded. Those which pass are tentatively identified as uncorrelated objects and are tracked.

Identified objects are tracked based upon either priority assignment or upon data validity tests. All objects to be tracked are compared and the one with the highest track priority is designated. The program directs the radar to the expected target position and continually up-dates this position until the radar "locks on" or can no longer be expected to acquire the object. If there are no more track requests, the system returns to scan. After "lock-on," an automatic radar hardware function, the radar sends track reports to the computer which are smoothed to form Q-points. The first of these is subjected to the same identification tests as the scan Q-point in order to insure proper identification. This orbital information is transmitted back to the Space Track Center. The number of data points transmitted and the length of track are dependent upon the priority of the track mission and the limits of the system.

4102-S SYSTEM

4102-S Computer

The 4102-S is the most recent addition to the 4100 series built by RCA, Van Nuys, California. The RCA 4102-S is a 16-level, 30-bit word length, parallel, binary, 2's complement, fixed point, fractional, interrupt I/O, stored program computer. The computer has 16 program counters and 96 full length index registers. An add instruction requires 19.2 microseconds and a multiply 75.5 microseconds.

The prime requirements for the SPASEC system demanded that the computer be reliable, binary, economical, and capable of communicating with the radar system in real-time. The RCA 4102-S met the real-time requirements with interrupt I/O as opposed to interleave I/O.

With the interrupt processing method used, a section of coincident current memory (CCM) is set aside and designated as "executive storage." This area is used by hardware to store the program counters and the index registers. There is an internal 15-bit register known as the "flag register" which is examined each time an instruction is executed. If a higher priority level flag is set the program counter for the higher level is accessed and a new instruction sequence is begun. When a level is through operating it may erase its own flag (suicide).

All levels may set any bit in the "flag register" but the bit may be cleared only by the level associated with the flag. With these set and reset commands programs and external devices can intercommunicate.

Interrupt processing uses the hardware level control to accomplish input/output while tying up main frame only a small portion of the time. Each command and I/O instruction suicides and the particular I/O device will set the priority flag when it is ready to accept or transmit more data. This allows the program to proceed normally, processing data and being interrupted when I/O devices need servicing.

Economies in computer hardware were achieved by the interrupt processing method, by fixing point logic and medium speed circuitry. Effective computer utilization was realized through the use of a simple assembly language and fixed point coding.

4102-S Configuration

In the 4102-S both program units and hardware devices have been assigned priorities. The level assignment is shown in Figure 3. The hardware devices are fixed priorities and are arbitrarily assigned according to the maximum allowable time lag between request and service of the device. Hence the faster, more impatient devices have higher priorities. Input/Output devices are associated with the first ten priori-

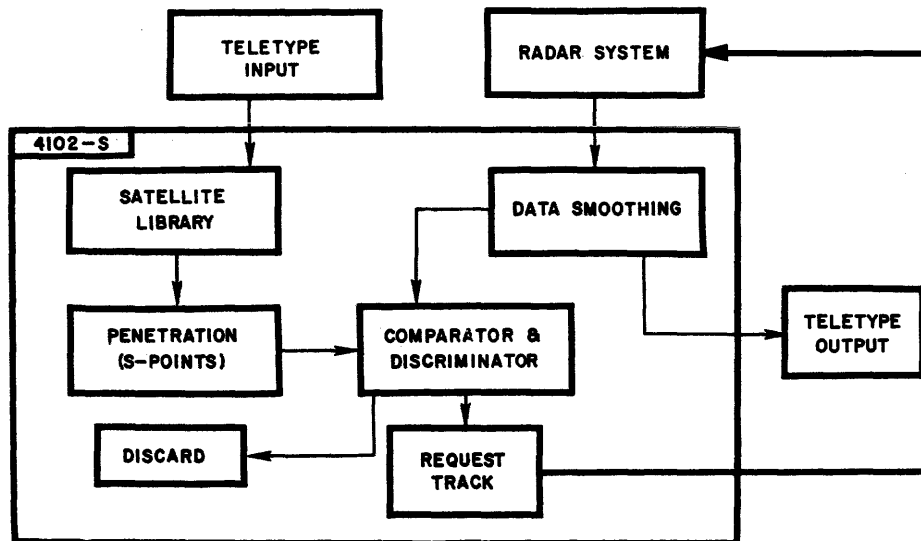


Figure 2. SPASEC Data Flow.

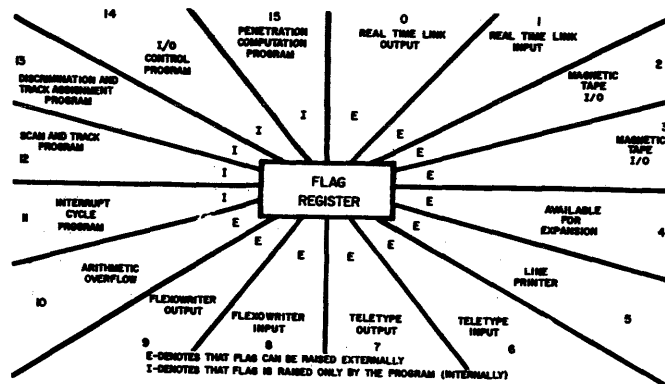


Figure 3. SPASEC Level Assignment.

ties; arithmetic overflow occupies the eleventh priority and program units occupy the next five priorities. The configuration is shown in Figure 4.

The real time interface is the computer's connection with the real world. Through this interface the computer receives information on radar returns in the form of digital signals representing range, azimuth, elevation, doppler and time. Radar system information messages are also given to the computer through this link. The computer can direct the radar to release or to acquire a target. The communications are more complex than this but the end results are as stated.

There are two magnetic tape drives, 7330's, which are used in low density only. The data rate of these units is 7.2 KC character rate. In order to match the IBM tape format a blank character is appended to or deleted from each word transmitted between the 4102-S and the tape units. This results in a data transfer rate of 833 microseconds per word. The tape units are used for fast program load and historical recording.

The line printer is an Analex 41000, capable of printing 1,000 lines per minute. The printer has a 64-character selection and a 120-character line. There is a punched vertical format control tape to allow program control of the output. The printer is used to provide a hard copy of the penetrations of the surveillance sphere.

A model 28 Teletype is connected to the computer through a special interface which can recognize sequences of Teletype characters as a request to start or end a message. The interface may operate in three input modes meaningful to the program: Automatic, in which case the information is made available to the computer directly from the tape punch heads; Program control—the program attempts to read the accumulated store of paper tape; reading of each Teletype message is initiated by the computer; Manual, the reading of the accumulated store of paper tape is initiated by enabling manual mode which must be reenabled for each succeeding Teletype message. The interface has two alternatives for Teletype output: Automatic—data generated by the computer is transmitted from the punch heads automatically to the Spacetrack Center and a back up paper tape is produced. Manual—the computer produces a punched paper tape which is transmitted later manually. All Teletype output messages are printed on the Teletype line printer.

The Flexowriter is a standard 2 case, 5-bit Baudot code machine with read, write and punch capabilities. The 4102-S hardware generated load program selects the Flexowriter, so initially the Flexowriter paper tape reader is used to load a bootstrap program which will load the operational program from a magnetic tape unit. After initial loading the Flexowriter reader is used to enter control messages to the program. The Flexowriter typewriter is used to print direction messages to the opera-

tor, equipment status messages and smoothed radar data.

PHILOSOPHY OF PROGRAM DESIGN

SPASEC was conceived to be a single purpose computer system as opposed to a general purpose system. Its task was to fulfill the Space Track mission twenty-four hours a day, seven days a week.

Throughout the design, coding and development phases of the program, particular attention was given to relieving operating personnel of responsibility for determining detailed system action and response. This design concept which stresses completely automatic operation is coupled with additional features to allow manual direction and/or intervention. The blending of operational flexibility and automatic control has made this system adept at gathering data for general and special purpose missions. This has allowed an operation which has significantly reduced the number of technical specialists necessary to conduct the most complicated assignments.

The SPASEC program is segmented and planned in such a way that the operation can continue with reduced capability. If the Flexewriter is inoperable the Teletype will be directed automatically to perform the Flexewriter output function in addition to its own. If as much as half of core memory is inoperable

the program continues relocated into the good half. In this case the satellite identification feature is lost. If the program is destroyed by a transient signal or intermittent error the program is quickly reloaded from magnetic tape.

The logic of the program and the system assumes that any changes in operating mode due to manual intervention are correct unless such changes are of such a nature as to physically damage the radar. If any procedure is unexpected the program will notify the operating personnel of such change by specific English reference to what has occurred (e.g., TRACK DENIED, RADAR INOP(erable), TRACK MODE). System malfunction is determined quickly and operator intervention is confirmed.

It was realized that with time, as additional operational requirements become known, program changes would be required. The actual requirements for changes which are being implemented attest to the validity of this concept. Based on this, features were incorporated into the program to facilitate modification, debugging and historical recall.

Historical recording is closely allied to the debug features within the SPASEC program. These features are used for program maintenance, system and hardware debugging as well as special studies. Access is allowed to the pro-

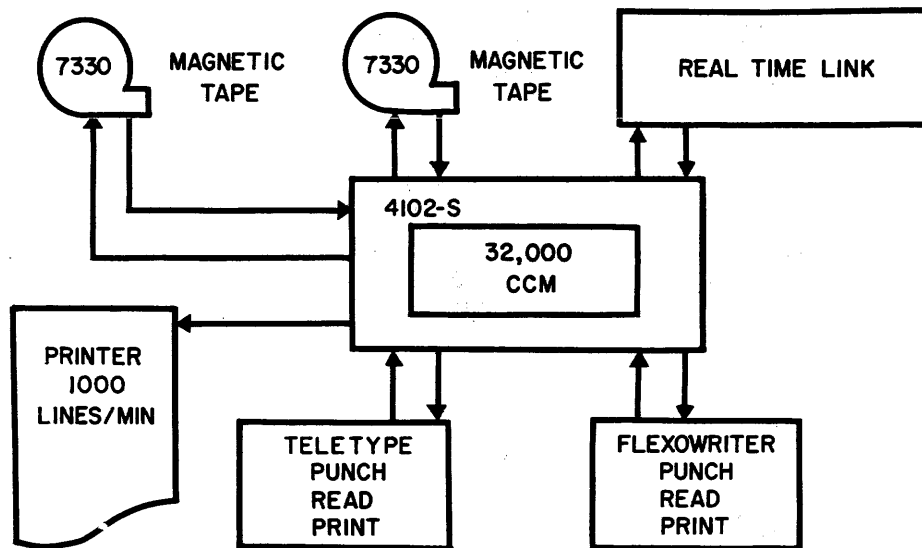


Figure 4. 4102-S System Configuration.

gram and its data in real time or off-line modes of operation.

## PROGRAM ORGANIZATION

### *Initialization*

The 4102-S is paper tape oriented, which for small programs is not unwieldy. For the SPASEC program a bootstrap procedure is used. The program is assembled on another computer with card I/O and brought to the 4102-S on a low density binary tape. This magnetic tape is then loaded under control of a small paper tape program loaded through the Flexowriter.

An integral part of the SPASEC program is the ephemeris file, a file which is constantly being changed. In order to reflect this change a new program tape is written during initialization and is positioned to receive changes to the ephemeris file. Each day a new tape is generated which is used for input on the next day. To enhance the reliability needed for an operational system, the program can be quickly reloaded from the next day's master without destroying any of the above capability.

### *Real Time Data Handling*

The interrupt cycle occupies the highest program-priority (see Figure 5), that is, program-priority as opposed to the priorities assigned

to the hardware devices. The duty of the interrupt cycle program is to act as an executive routine and clearing house for other sections of the program that communicate with the real time interface or are concerned with real time. The rationale for this is that the communication with the real time interface must be done at certain times within the radar pulse interval and to avoid confusion it was advisable to have one section of program control the data flow. This section must be executed periodically, hence the high priority level. The interrupt cycle is activated as a program once per radar pulse interval, therefore, it is well suited to update timers and in general furnish the program with a pulse. Periodic entry to lower priority levels is effected through this program.

Since the radar system can be in track or scan but not both, data processing is performed on the same level by two mutually exclusive programs. The scan portion of the program looks at returns from two complete scans and attempts to associate the various returns. The data that the scan program has access to consists of "end of scans" (EOS) and scan reports. When reports from two consecutive scans are believed to be from the same object they "associate." Association of reports is not attempted until a configuration of the type described in Figure 6 is attained. The only items of interest are the reports between EOS#K

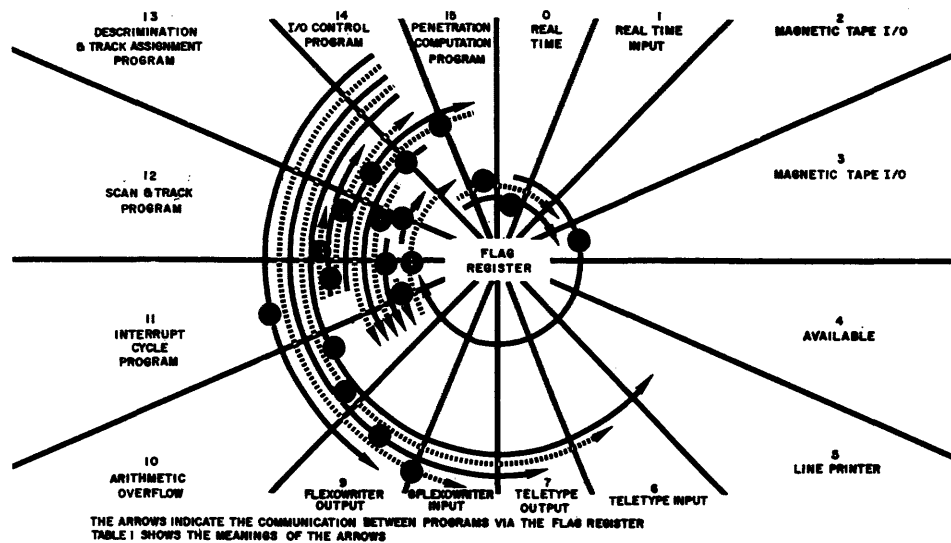


Figure 5. SPASEC Program Organization.

and EOS#(K+2). The reports between EOS#K and EOS#(K+1) are reports that represent data on an object for one scan and reports that represent smoothed data of more than one scan. The data between EOS#(K+1) and EOS#(K+2) represent current reports which may or may not associate with data between EOS#K and EOS#(K+1). When it is determined that there is no data to update a previously smoothed data report, the cumula-

tive information on this object is given to a track decision routine for further action (see Figure 7).

Smoothed scan reports (scan Q-points) are developed from weighted radar parameters (range, doppler, azimuth, elevation, time and credences). Additional parameters are developed from this base data such that a final scan Q-point contains range, range rate, range acceleration, azimuth, azimuth rate, elevation, elevation rate, time and an accuracy index. Q-points generated from scan data and initial track Q-points are compared with the ephemeris file (S-points) by the comparator program. The results of the comparison test yield one of the following results: fine, passed comparison within close time and position tolerances; coarse, passed comparison within broad time and position tolerances; coarse, tentatively identified with more than one object; uncorrelated, object unidentified.

Uncorrelated object Q-points are processed through discrimination tests which are designed to determine if the object is a satellite. These tests are based upon energy considerations and are designed to eliminate meteors, the moon and noise. Q-points which pass these tests are placed on a tracker waiting line and are tagged uncorrelated. Those which fail are discarded as being of no interest.

Each object on the tracker waiting line has associated with it a track priority. This priority is a function of the object identification, data age, and probability of detection. The program selects the object with the highest tracking priority and attempts to track for a period of time based upon the track criteria.

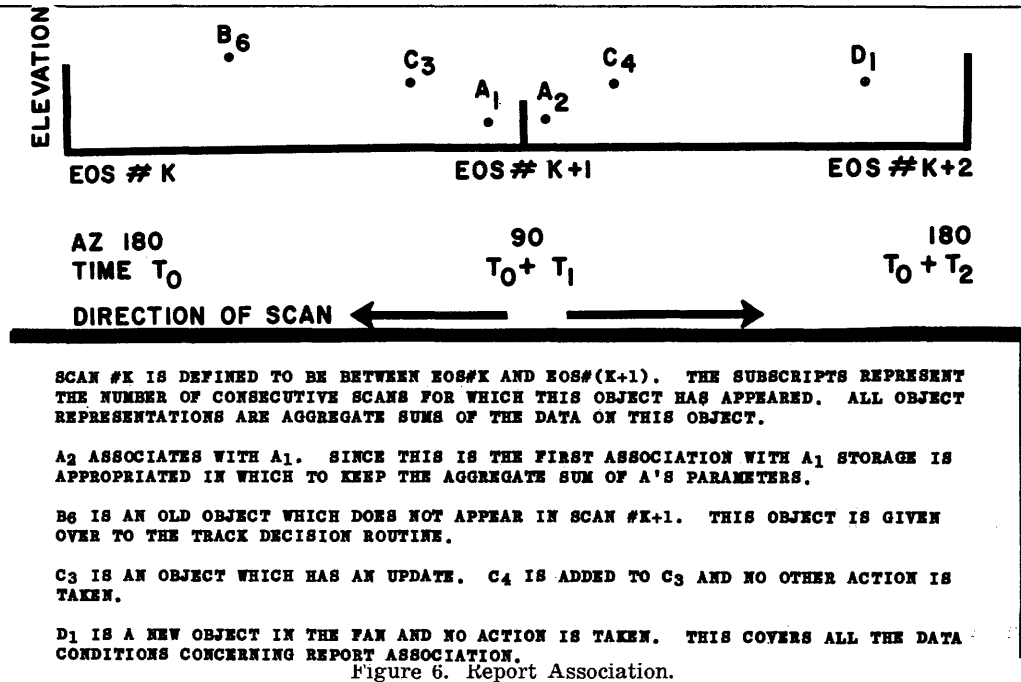
To automatically track an object the program directs the radar to a point along the object's path slightly ahead of the object. This designation message is transmitted to the radar system through the real time interface causing the radar antenna to be directed to a point in space. For a short period of time the object's path can be described as a second order curve in each of the radar's parameters. The six Q-point parameters are used to find values for the second derivatives of range, azimuth and elevation.

Table I.

Program Communication

Flag

- (1) Magnetic Tape I/O.
- (2) Magnetic Tape I/O.
- (3) Real Time Interrupt.
- (4) Start Message Output!
- (5) Compare Data!
- (6) File Error Message!
- (7) File Error Message!
- (8) File Error Message!
- (9) File Error Message!
- (10) File Error Message!
- (11) Start Penetration Calculations!
- (12) Cycle Through I/O Devices!
- (13) Check For Data To Compare!
- (14) Smooth Data!
- (15) Printer Control
- (16) Teletype Input Control
- (17) Teletype Output Control
- (18) Flexowriter Input Control
- (19) Flexowriter Output Control



The designation point is current time plus a half second and is valid for one second. If the object is not detected, a new designation point is computed. This procedure may continue for thirty seconds. If the object is detected, the radar automatically "locks-on" to the object and the track is begun.

Track data is smoothed using unweighted arithmetic means with least squares fits to develop the rates. The final track Q-point contains range, range rate, azimuth, azimuth rate, elevation, elevation rate, average angular credences and orbital elements consisting of inclination, period, semi-major axis, eccentricity and right ascension. The track Q-point presently is developed from 10 seconds of radar data.

A "cast out" routine was developed to limit the transmitted data to representative portions of the track. At termination of track this routine produces 3 Q-points if a short track and 10 if a long track. In the case of a long track the Q-points will be as equally spaced as possible throughout the entire time interval. The track program examines the reports and will terminate the track if the radar antenna should attempt to enter unavailable regions in the sphere of surveillance. For a short track the program terminates when three track Q-points have been developed.

#### *Penetration Computations*

The S-point file is generated from the set of current orbital elements contained in computer memory. Satellite elements, radar site coordinates, radar sector, and time period of interest are used to compute predictions. Non-penetrating satellites are quickly rejected from consideration based on tests of the satellite's inclination and time of horizon passage.

There are five types of penetration schedules available to the operations director to preplan missions. The five penetration requests are: 1, penetrations for all objects ordered by time; 2, penetrations for all objects ordered by satellite number in increasing order; 3, penetrations for a priority class of objects ordered by time; 4, penetrations for a priority class of objects ordered by satellite number; 5, look angles, a series of penetrations of a single object separated in time between two elevations.

When operating in real time, penetrations (S-points) are automatically generated for all azimuths at a specified elevation and filed for use by the comparator program. The file always contains one penetration interval, normally 40 minutes ahead of the current time. When S-points become more than six minutes old, they are automatically deleted. This allows continuous operation with no loss of S-point



storage. The S-points are printed after the computations for an entire interval are completed. When a print request is received in real time, the real time S-points are deleted and the print request is honored. Upon completion of that printing, the real time S-points are automatically regenerated and printed.

*I/O Control*

The prime considerations in the design of the I/O control program were ease of use and movement of data at the acceptance rate of each device. The input/output control routine communicates with each of the following I/O devices, magnetic tape units, Flexowriter, Teletype and line printer. This master control routine is responsible for scheduling each of these devices based on availability, speed of operation and message priority. For output the user program notifies the I/O routine through a calling sequence which specifies the I/O device, the address of the data and the type of message. From the point of view of the user program the data is considered transmitted to the output device after the calling sequence has been executed. In reality the request is stacked

and will be processed when the device is available.

For input the entire transfer of information is automatic and no processing is done on any message until the transfer is complete. The message is identified, converted and delivered to the responsible routine. This is accomplished by flagging the proper sub-program and giving the location of the data. Movement of large blocks of data is eliminated through use of simple list techniques. Only references to data are communicated between sub-programs.

Real time I/O is treated similarly but independently of the main I/O control program to assure minimum response time.

*Data Manipulation Techniques*

Data handling in this program makes use of simple list structures, e.g., chained lists, and key buffers referencing lists. Nearly all sections of the program work with chained lists.

In our application list structures have served several purposes. Lists are used to give items an effective consecutive order, regardless of

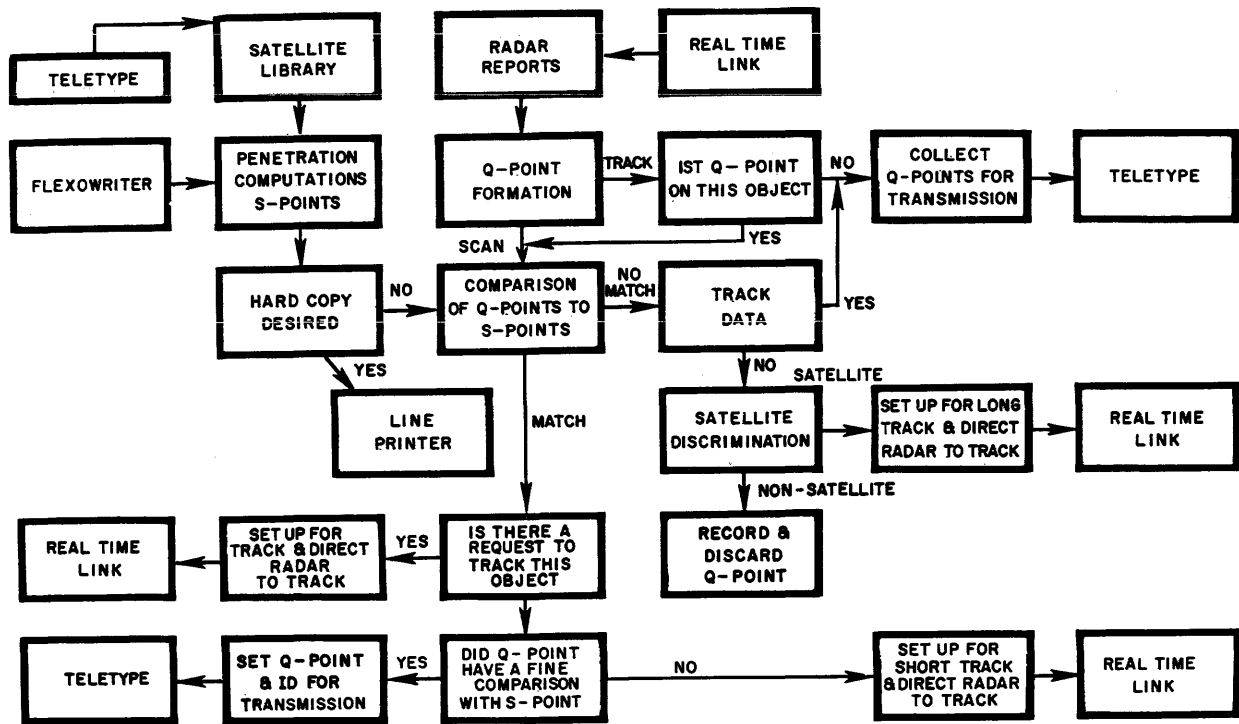


Figure 7. Detailed Data Flow in the SPASEC Program.

core location. They allow access to a large amount of data through a single reference. Lists are a convenient way to have several programs use common storage. They are time saving when large blocks of data are to be transferred among programs. Lists save storage when several routines of high peak storage requirement, at staggered times, can use a common storage. Instead of having to assign storage for the worst case for each of the programs, memory is assigned on the basis of the overall requirements.

It should be noted that it is not necessary to achieve sophistication in list structures to make use of them. List structures can be programmed in varying degrees of complexity tailor-made for the particular program. In programming for the Space Track mission, list processing was found to be efficient in terms of space and time and was a convenient way to visualize data transfers.

#### SYSTEM GROWTH

As with any experimental system certain growth can be expected in the hardware configuration and in the sophistication and diversification of the programming effort. This expansion may be divided into two categories: that which is presently being implemented and future planning. Present implementation includes transmission of pointing data in real time to a narrow beam, short range radar in Baltimore, Maryland. A high-speed (2400-bits/sec.) data link will provide communications. This service may be expanded in the future to provide similar data to other sites. Push button input for control of program options is under development. This will allow operations faster response to requests. Program changes are being installed to allow automatic recognition of radar coverage changes and to recompute S-points based on this new sector.

Plans for future improvement and growth include: additional storage for system enhancement; card reader/punch; displays. This additional storage would allow an expanded ephemeris file when the known satellite population becomes much larger than the present 400 objects, and would be used to store different production programs which would time share with the operational program. A card reader and punch would allow a desirable re-orientation of our present paper tape system. A displays facility with computer control of a slide projector would allow a concise picture and quick human identification in most cases.

#### CONCLUSION

The 4102-S SPASEC program has proved itself to be a flexible and reliable satellite surveillance program. Working as part of the Moorestown Space Track facility it has demonstrated its ability to identify satellites and direct the radar system in the collection, reduction and forwarding of data. The use of this program has enhanced the operations of the Space Track mission and made this site one of the network's more important members.

#### ACKNOWLEDGMENT

J. F. Russell assisted with the preparation of the material presented herein.

#### BIBLIOGRAPHY

1. E. T. GARNER and J. F. RUSSELL, "Description of the 4102-S Space Track Program," SPA-1100-2, July 19, 1963.
2. E. T. GARNER, "List Processing Techniques Used in the BMEWS Engineering Model Building Space Surveillance Program," presented at Information Processing Symposium, RCA, Princeton, New Jersey, October 1963.
3. J. A. CORNEL, "The RCA 4100 Series Military Digital Computers," RCA ENGINEER, Vol. 8, No. 6, April-May 1963.

# A HYBRID COMPUTER FOR ADAPTIVE NONLINEAR PROCESS IDENTIFICATION

*Rob J. Roy*  
*Rensselaer Polytechnic Institute*  
*Troy, New York*

*and*

*Bruce W. Nutting*  
*Burroughs Corporation*  
*Paoli, Pennsylvania*

## INTRODUCTION

The physical processes which exist are, for the most part, controllable. In order for man to exert control over these processes, however, it is necessary for him to design a system to suitably perform this task. Such a design requires a characterization of the process to be controlled.

Often the characterization of a process can be determined from the physical principles involved in the process behavior. This has been true for most of the control systems designed in the past. As man seeks to control more complicated processes, however, it is found that either many processes are extremely difficult to physically characterize, or that the characterization is too complicated to use in the determination of a suitable control system. For a complex process of high order with several nonlinearities, there is no procedure which can accurately obtain all the process parameters. Fortunately, however, there is no reason for obtaining all the process parameters and all the nonlinear functionalities.

The basic question of process identification is cause and effect. For a given cause, what is the effect? For example, in order to control

human response, it is not required that the complex nerve and brain structure be known. The only practical method would be to observe a series of causes and effects, and from this data infer the proper cause to produce a desired effect. Not only is this procedure practical, but it is adaptive as well. If the cause-effect relationship changes, the observer is made aware of this fact. Parameter tracking for a structure as complex as a human being would be absurd. It is equally as absurd to track the parameters of a tenth order nonlinear process.

For these reasons, a relatively simple and straightforward method of relating process inputs and outputs is necessary. This method should be suitable for nonlinear, as well as linear, processes. Furthermore, the process characterization should adaptively follow process changes due to such things as changes in environment. Since special test signals result in a disturbance of the process outputs, they should not be required to characterize the process.

It should be noted that for control purposes one is interested in determining an input to produce a desired output. It is important to

realize this does not mean it is necessary to determine the process transfer function or describing function, or differential equations, etc. In fact, some of the above mentioned types of characterizations are not applicable to many processes of interest.

This paper is concerned with those processes which have a two-level switched input and finite settling time. The importance of the switched two-level process has its foundation in the utilization of predictive control as a practical solution to least time optimization problems.<sup>1-9</sup> Chestnut et al. indicate significant applications for predictive control in space navigation and rendezvous missions, aircraft landing problems, and the control of chemical plants.<sup>4</sup> The process identification concepts presented in this paper are ideally suited for predictive control applications.

### THEORY OF THE MODEL

The hybrid computer model was designed for predictive-adaptive control systems. The model has a switched two-level input signal, which for convenience will be called the zero and one levels. The only *a priori* knowledge of the process which is required is an estimate of the settling time of the process, and the fact that this settling time be finite. The requirement of an estimate of the settling time of the process is needed to provide rapid identification of the process. This requirement can be removed, and the model can be made to adapt to the settling time of the process. However, even though the process may be nonlinear and only the input-output operating record is available, it is reasonable to assume that the range of the settling time of the process is known.

The theoretical basis for the model is the work of Wiener and Bose. The classic theorem of Wiener<sup>10</sup> states that any nonlinear system which has finite settling time can be characterized by a multiple output linear system which characterizes the input past, followed by a zero memory nonlinear system.

The multiple output linear system used by this model is an  $n-1$  stage shift register. If the binary input is fed into this shift register, the  $n$  outputs (including the present input) will approximate the input history for the past

$(n-1)T$  time units, where  $T$  represents the time interval between shift pulses.

The zero memory nonlinear system performs a nonlinear transformation of the  $n$  outputs of the shift register into the output of the model. Thus, the model can be viewed as a sequential machine with  $2^n$  states, where the states are defined by the  $n$  outputs of the shift register. The output of the model is a function of the present state of the model. An example of this is shown in Figure 1. This sequential machine has four states, the states of the machine being defined by the past two inputs to the device. The output of this device is a function of the present state  $S_i$ . If the state  $S_i$  is defined as a switching function, being one when the device is in state  $i$  and zero otherwise, then the output of this sequential machine can be written as

$$Z(t) = \sum_{i=0}^3 K_i S_i(t) \quad (1)$$

$Z(t)$  = model output

Extrapolating this simple model to a sequential machine with  $2^n$  states, then

$$Z(t) = \sum_{i=0}^{2^n-1} K_i S_i(t) \quad (2)$$

The model output  $Z(t)$  is equal to  $K_i$  when the model is in state  $S_i$ . Since the machine can be in only one state at any given instant in time, only one term of the summation of Eq. 2 is non-zero at any given time.

If the true output of the process is  $y(t)$ , then it is desirable that the  $K_i$  be adjusted such that the mean square error between the true output  $y(t)$  and the model output  $Z(t)$

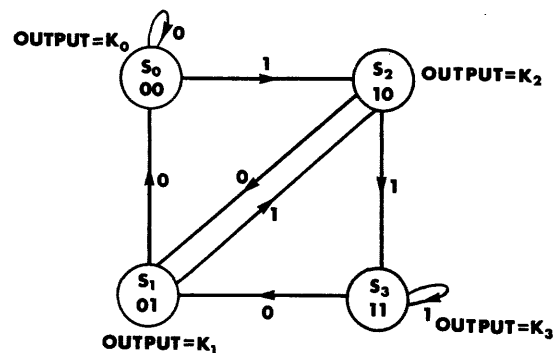


Figure 1. Sequential Model.

be minimized. The following function must then be minimized, the bar indicating an infinite time average.

$$E = \overline{(y(t) - Z(t))^2} = \overline{(y(t) - \sum_i K_i S_i(t))^2} \tag{3}$$

Taking the derivative with respect to a dummy term  $K_\beta$

$$\frac{\partial E}{\partial K_\beta} = \frac{\partial}{\partial K_\beta} \overline{2(y(t) - \sum_i K_i S_i(t)) (-S_\beta(t))} \tag{4}$$

Since the states are disjoint in time, the product of the switching functions  $S_i(t) S_\beta(t)$  is zero except when  $i = \beta$ . Since the switching function  $S_\beta(t)$  is either zero or one, the product  $S_\beta(t) S_\beta(t)$  is equal to  $S_\beta(t)$ . Therefore,

$$\frac{\partial E}{\partial K_\beta} = \overline{(-2) (y(t) S_\beta(t) - K_\beta S_\beta(t))} \tag{5}$$

If Eq. 5 is set equal to zero, then the coefficient  $K_\beta$  is found to be

$$K_\beta = \frac{\overline{y(t) S_\beta(t)}}{\overline{S_\beta(t)}} \tag{6}$$

The coefficient  $K_\beta$  can be interpreted as representing the average value of the true output  $y(t)$  when the sequential model is in state  $S_\beta$ . If the process is slowly time-varying, then  $K_\beta$  should be a function of time, such that the model follows the time variations of the process with minimum mean square error.

For control purposes it is desirable that the model have the capability of predicting future values of the process output from the knowledge of the present state of the model. This capability can be expressed as follows,

$$Z(t) = \sum_\beta K_\beta^{t+kT} S_\beta(t) \approx y(t+kT) \tag{7}$$

where  $kT$  represents the prediction time. The superscript  $t+kT$  indicates that  $K_\beta^{t+kT}$  is the predicted value of the process output  $kT$  units in the future. Following the same procedure as used to find  $K_\beta$ , the coefficient  $K_\beta^{t+kT}$  is found to be

$$K_\beta^{t+kT} = \frac{\overline{y(t+kT) S_\beta(t)}}{\overline{S_\beta(t)}} \tag{8}$$

Since it is impossible to know the future output of the process unless the future input is known, the best that can be done is to assume stationarity in both the process parameters and the input statistics.

Under these assumptions, the coefficient  $K_\beta^{t+kT}$  can be expressed as

$$K_\beta^{t+kT} = \frac{\overline{y(t) S_\beta(t-kT)}}{\overline{S_\beta(t-kT)}} \tag{9}$$

This expression for the prediction coefficient can be implemented with a shift register of  $n+k-1$  stages. The prediction coefficient is obtained by associating the present average process output with the state corresponding to the *last*  $n$  outputs of the shift register (taps  $k$  through  $k+n-1$ ). When that state next appears at the *first*  $n$  outputs of the shift register (taps  $0$  through  $n-1$ ) the model remembers what the average value of the subsequent output was when that sequence appeared previously, and predicts this value for the future model output. Essentially, this prediction is the conditional mean of  $y(t+kT)$  given the particular state  $S_\beta(t)$ . The implementation is shown in Figure 2. The average value of the present output can be used as the identification coefficient  $K_\alpha$  associated with state  $S_\alpha$ , or as the prediction coefficient  $K_\beta^{t+5T}$  associated with state  $S_\beta$ .

The preceding discussion describes the basic theory of the identification procedure. Since the purpose of this paper is to describe the hybrid computer which was designed from this basic viewpoint, the theoretical details of model error, identification time, etc., are omitted. These points are adequately discussed in Reference 11.

For the purposes of this paper, it is sufficient to state that if the minimum mean square error of identification is given by  $E_{min}$ , then the average value of the additional mean square error introduced by having only a finite input-output operating record is given by  $1/M E_{min}$ , where  $M$  is the number of times each state has occurred. Thus, the additional MSE due to a finite operating record is no more than  $1/M E_{min}$ , if each sequence occurs at least  $M$  times in the record. Consequently, the mean square error caused by having each state appear only

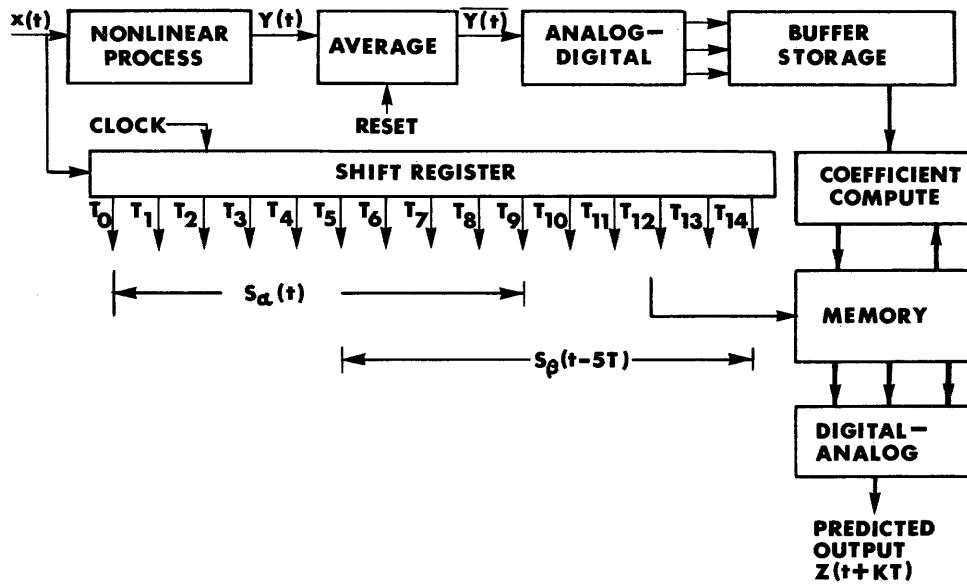


Figure 2. General System Diagram.

once is at most twice the minimum mean square error.

Since the minimum mean square error can be made extremely small by increasing the length of the shift register (increasing  $n$ ), the identification using this technique should be quite good.

Before passing to the discussion of the design of the process identifier, there are two points of practical interest. The first point is that although there are  $2^n$  possible states which may occur in the theoretical model, practical considerations will limit the number of states that the actual model will experience. For a practical control system, the input will probably not switch  $n$  times during the process settling time. Some studies<sup>11-12</sup> on a ten tap system have shown that it is reasonable to expect that no more than 200 different states will occur, although the specific states which do occur are not known. This important observation reduces the required memory capacity considerably. The second point is that knowledge of the process input in the immediate past is generally of more importance in determining the present process output than inputs which occurred some time in the past. Therefore, it is advisable to have the capability of tapering the outputs of the shift register. That is, spacing the first few shift register outputs closer timewise than

the last few shift register outputs. This is accomplished by increasing both the length of the shift register and the shifting frequency by a multiple of two. A typical example of tapering is shown in Figure 3.

#### COMPUTER DESIGN

The basic components of the hybrid computer are the input shift register, the main memory, analog-to-digital and digital-to-analog converters, and an analog weighting and averaging section. See Figure 4. The input shift register is used to store the present input sequence to the process. The main memory is a converted IBM 650 magnetic drum, and is used to store the past input sequences  $S_\alpha$ , the associated coefficients  $K_\alpha$ , and a weighting number which indicates the number of times that a particular sequence has occurred. The analog section is used to find the average process output over the time that a particular sequence appears. This average value is combined with the appropriate coefficient  $K_\beta$  and weighting number to form the new coefficient  $K_\beta$ . The analog-to-digital and digital-to-analog converters provide the required interface between the analog and digital portions of the computer.

The input shift register, used to form a temporary storage for the binary process input, consists of 20 flip-flops. The shifting frequency

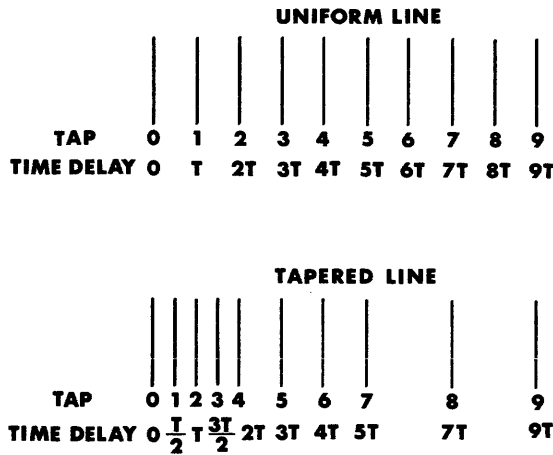


Figure 3. Tap Arrangement.

(input clock) is determined by the settling time of the process being observed. The input and input past are taken from the shift register at 10 points (via a patch board), and these 10 points are used to form an address ("predict" address). The drum memory is searched for a coincident address, and if one is found the information and weighting number associated with that address are placed in registers. A digital-to-analog conversion is performed on the information, and this analog signal is combined with the process output in a ratio determined by the weighting number. An analog-to-digital converter produces the binary equivalent of the analog signal formed by the weighted combination mentioned above. If the weighting number has not reached a preset limit, it is increased by one, and both the new weighting number and the information from the analog-to-digital converter are placed in memory at the coincident address. Each memory location contains 10 address bits (the sequence  $S_a$ ), 6 information bits (the coefficient  $K_a$ ), 4 weighting number bits (the number of times  $S_a$  has appeared), and a check bit which indicates unused memory locations.

If the outputs of the 10 taps form an address which has not been observed before, the machine analog output and the process output are combined in a 0:1 ratio and the weighting number is set equal to 1. The information from the analog-to-digital converter, the weighting number, and the new address are all placed in memory at a previously unused position.

A second group of 10 points is taken from the shift register (via the patch board) to form the model output address ("identify" address). A second coincidence circuit searches the drum for this address, and if coincidence occurs the information associated with the model output address is placed in a second register. The digital-to-analog conversion of the information in this register is used as the model output. The weighting number associated with this information is not used at this time.

If the "predict" and "identify" addresses are taken from the same 10 taps of the shift register, the model output will identify the process observed. If the "identify" address is taken from a group of ten taps which are of the same configuration as the 10 taps of the "predict" address, but located closer to the start of the shift register, the model output will predict the process observed.

The analog section of the model consists of four operational amplifiers. Two of these operational amplifiers are used for the output of the digital-to-analog converters. One amplifier is used to perform the averaging of the process output over the time interval  $T$ , and one amplifier is used to perform both the weighting of the present process output with the stored coefficients and the sample and hold function required for the analog-to-digital converter.

The process averager is an integrator which is sampled and reset to zero once each cycle. If the time between input clock pulses is  $T$ , then the average of the input voltage over one cycle

$$\text{is: } e_{\text{average}} = \frac{1}{T} \int_0^T e_{\text{input}} dt. \text{ The equation of the}$$

integrator (assuming zero initial conditions)

$$\text{is: } e_{\text{output}} = -\frac{1}{RC} \int_0^T e_{\text{input}} dt. \text{ Therefore, RC is}$$

set equal to  $T$ . The reset time of the integrator (1 millisecond) is negligible compared with the values of  $T$  to be used (of the order of 0.4 second).

The two digital-to-analog converter blocks marked DAC in Figure 4 serve to connect an

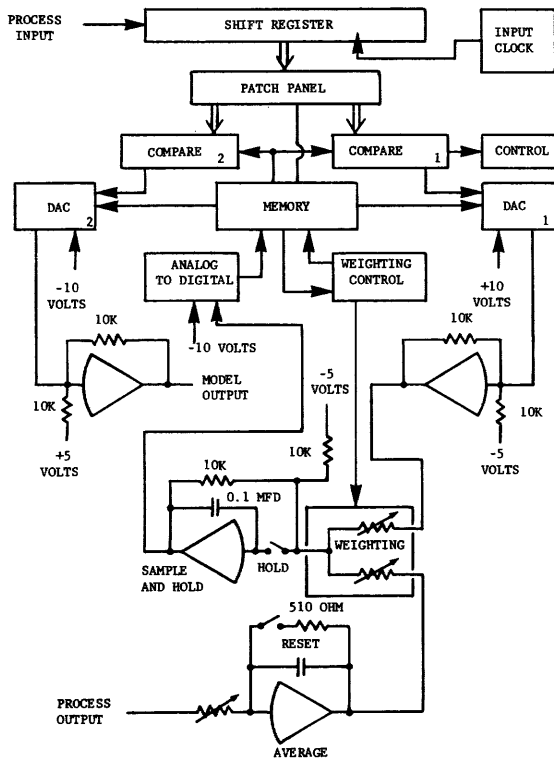


Figure 4. Detailed System Block Diagram.

input reference voltage to the input of an operational amplifier through a variety of resistances. These networks are arranged such that when they are connected to the input of an operational amplifier with a 10,000 ohm feedback resistor, the amplifier output will vary from 0 to 10 volts in 64 steps corresponding to the 64 possible binary numbers in the digital-to-analog converter register. Note that a 5 volt bias signal is added to each DAC amplifier input to change the 0 to 10 volt output to a  $\pm 5$  volt output. This will cause the binary number 32 in the DAC register to be equivalent to an analog output of zero volts.

The two digital-to-analog converters have opposite polarities. The DAC No. 2 unit which produces the model output is connected for a positive or "correct-polarity" output. The DAC No. 1 unit produces an "inverted-polarity" output. Since the process output has also been inverted (by the process averager), these two signals are weighted and added directly. The sample-and-hold amplifier will invert the weighted sum, causing a "correct-polarity" signal to enter the analog-to-digital converter.

The weighting unit is a resistor and reed relay network controlled by digital logic. This network is designed such that when it is connected to an amplifier with 10,000 ohms feedback resistance, the output of the amplifier will be  $\frac{e + Xe'}{1 + X}$  where  $e$  is the signal from the process averager,  $e'$  is the signal from the DAC No. 1 amplifier, and  $X$  has integer values from 0 to 15 depending on the binary number in the weighting register.

The amplifier which is used to perform the weighted addition also performs a sample-and-hold function. The 0.1 microfarad feedback capacitor will cause the circuit to remain at a given voltage when the "hold" relay is opened. This is necessary in order to present a constant voltage to the analog-to-digital converter circuit during conversion. Note that a 5 volt bias is added to the input of this amplifier to convert the  $\pm 5$  volt inputs to the 0 to 10 volt output which is presented to the analog-to-digital converter. The analog-to-digital converter uses the 0 to 10 volt analog input to produce a 6 bit binary output. Again it should be noted that because of the bias employed, the binary output number 32 actually represents a zero volt signal.

The memory element of this computer is a modified IBM 650 magnetic drum memory. The modification has been such that instead of the one head per track formerly used, there are 42 tracks available with three heads per track. The three heads on the modified tracks are used for read, bias, and write.

The method of recording used is non-return-to-zero. The bias head continually records a "zero" on a given track, and the write head then records a "one" over the "zero" if it is on, or allows the zero to remain if it is off. The read head detects changes between the "one" and "zero" states as positive and negative pulses, which are used to set a flip-flop to either the "one" or "zero" state, corresponding to the information recorded on the drum. The memory is of the circulating type in that the information must be read and rewritten each time the drum makes one revolution.

Only 22 of the tracks on the drum are used in the machine (the drum originally contained over 200 tracks; the modification reduced this



number to around 140, with 42 of these being accessible from three different heads). Twenty-one of the 42 three-head tracks are used for information storage, and a timing track is used to generate clock pulses at a 125 Kilocycle rate (this pulse rate is divided in the timing logic, so the machine clock frequency is 62.5 Kilocycles).

It takes approximately 3.85 milliseconds for information to pass from a position under the write head to a position under the read head. Both the clock frequency and the time necessary for the drum to complete one cycle (from write head to read head) depend on the drive motor speed (nominally 12,500 rpm) and will vary somewhat, but one cycle will always contain 240 clock pulses—one clock pulse for each storage location.

Each word in the memory is stored in three parts, which are available at consecutive machine clock times. First there is a check bit (bit 21) which indicates the presence of a word on the drum as opposed to an unused memory position. Following the check bit by one bit-time is the address (bits 1 through 10) which represents a particular shift register pattern (as observed via the patch board). One bit-time later the information (bits 11 through 16) and

the weighting (bits 17 through 20) are available.

The front panel of the completed machine is shown in Figure 5. Side views of the machine (just prior to completion) are shown in Figures 6 and 7.

## TESTS AND RESULTS

In order to check the capabilities of the model, several test systems were programmed on a PACE TR-10 analog computer. The systems used are shown in Figure 8. The input to all systems was a pseudo-random binary function (limits were set on the rate of input switching).

A four channel Sanborn recorder was used to observe the input, the systems, and the model. All binary inputs were between the levels of +10 and -10 volts throughout the test. Attenuators were used to reduce the system outputs to  $\pm 5$  volts.

Figure 9 shows the results of modeling a simple  $\frac{-1}{s+1}$  system (system No. 1). An input clock rate of 2.5 pulses per second was used, with a minimum of 4 input clock pulses per binary input change. The weighting limit was set to 15. It can be seen that smoothing the

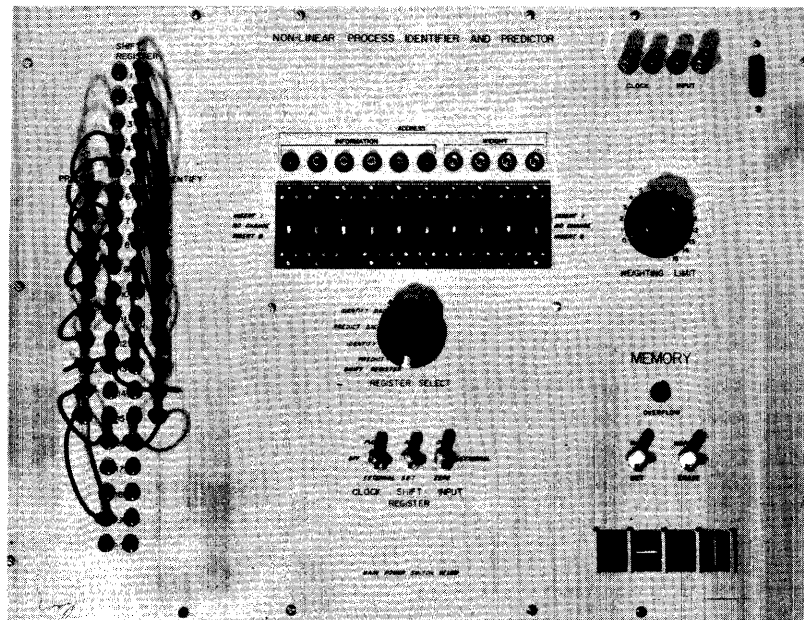


Figure 5. Front Panel of Computer.

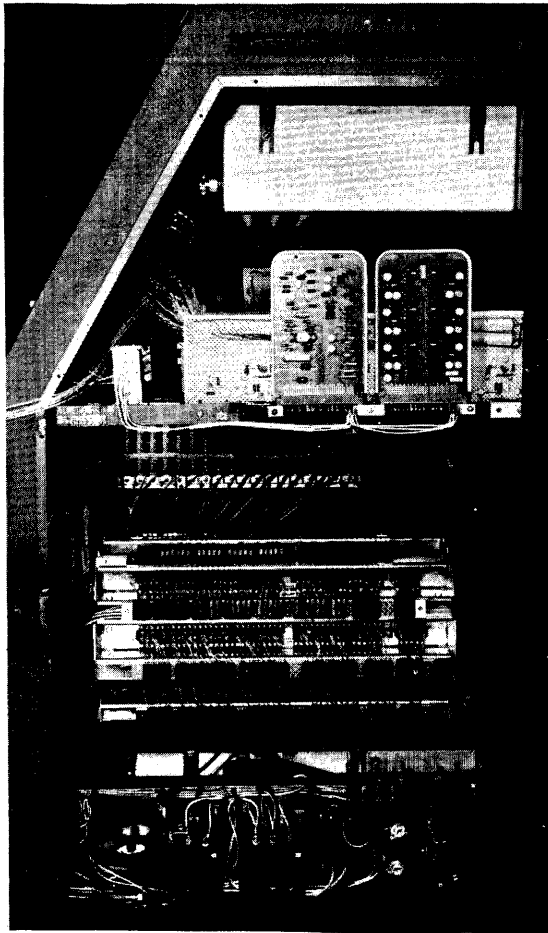


Figure 6. Right Side View of Computer.

model output (through a  $\frac{10}{s+10}$  transfer function) helps create a better looking output, but induces a slight delay in the model output.

Figure 10 shows the result of modeling the more complex system formed by following the  $\frac{-1}{s+1}$  function with a nonlinear element (system No. 2, Figure 8). The same input conditions and machine limits as above were used.

Figure 11 and Figure 12 show the use of prediction to compensate for the delay caused in the model output smoothing circuit. The system observed is composed of a  $\frac{-1}{s+1}$  function and a nonlinear element followed by a second  $\frac{-1}{s+1}$  function (system No. 3). A 2.5 pulse per second input clock was used with a

minimum of 8 clock pulses per binary input change. A weighting limit of 15 was employed. In Figure 11 the model is identifying the system quite well, but a rather large error is present due to the delay in the model smoothing circuits. In Figure 12 a 0.8 second prediction is used to compensate for this delay, and the error in this case is much smaller than before, being caused primarily by fast variations in the system which are represented by an average value at the model output.

Figure 13 illustrates the action of the model in identifying the  $\frac{-1}{s+1}$  transfer function (system No. 1) when the input binary function is allowed to change every 2 input clock pulses. An untapered delay line, weighting limit of 15, and 2.5 pulse per second input clock were used in this test. A 0.4 second prediction is used to

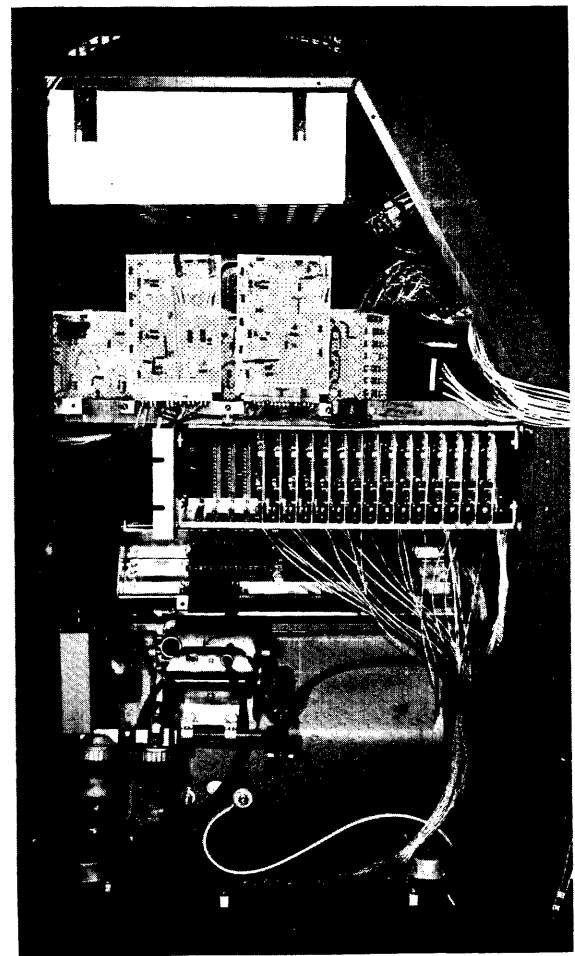


Figure 7. Left Side View of Computer.

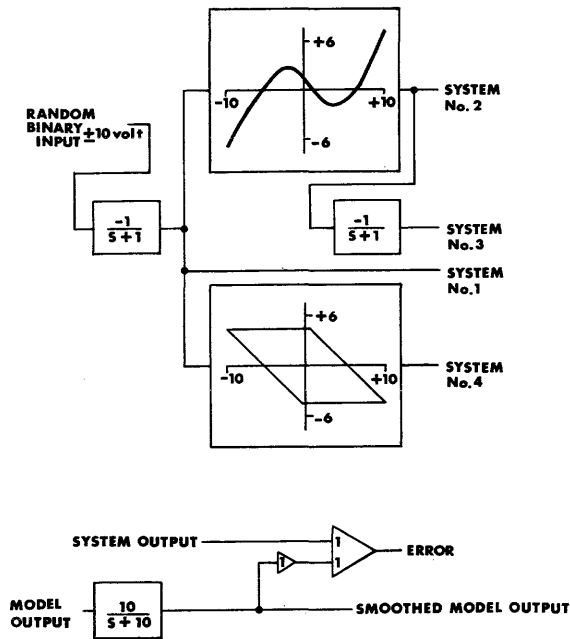


Figure 8. Test Systems.

offset the delay caused by model output smoothing.

Figure 14 gives the results of modeling a system with a large amount of hysteresis (system No. 4, Figure 8). A 2.5 pulse per second input clock is used with a minimum of 4 bits per binary input change. A prediction of 0.4 second is used to compensate for smoothing delay. The weighting limit was set to 15. All of the waveforms shown in Figures 9 through 14 were taken after a 60 to 120 second learning period.

Figure 15 illustrates the use of the machine as an adaptive model. The system observed in this case was abruptly changed from system No. 1 to system No. 3. These systems differ not only in general output waveforms, but are

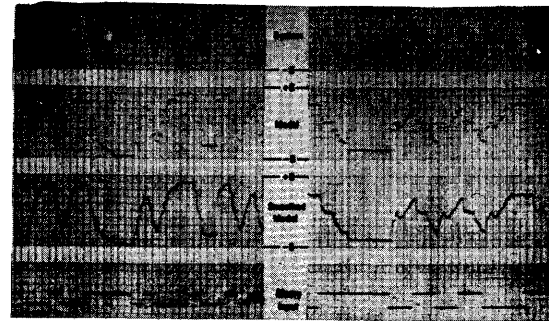


Figure 9. Identification—System No. 1  
Figure 10. Identification—System No. 2.

also of opposite polarity for a given binary input. An input clock frequency of 2.5 pulses per second was used, with a minimum of 8 bits per binary input change. A 0.4 second prediction was used to compensate for smoothing delay. A weighting limit of 2 was used so that the model could adapt quickly to the changing system. Although the error does not reach a minimum in the short record shown, it is obvious that the model is adapting very quickly indeed to the abruptly changing system.

SUMMARY AND CONCLUSIONS

The concept of using a synchronous sequential machine to model a continuous nonlinear process appears to be quite useful. The model which was constructed for processes which have a switched two-level input performed quite well both in the identify and the predict mode. This model is presently being used in the study of adaptive predictive nonlinear control systems.

Clearly, the model which is used for processes with two-level switched inputs cannot be used for processes with more than a two-level input. This is due to the large number of coefficients which must be stored. However, the basic philosophy of matching an input pattern

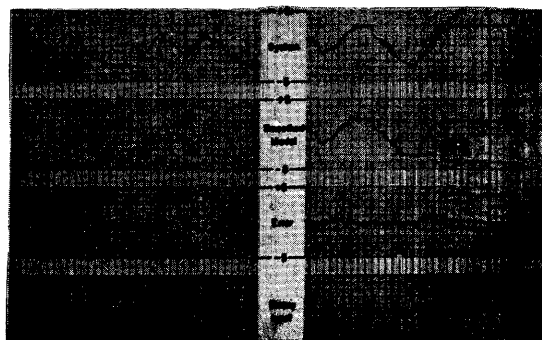


Figure 11. No Prediction—System No. 3.

Figure 12. 2 Bit Prediction—System No. 3.

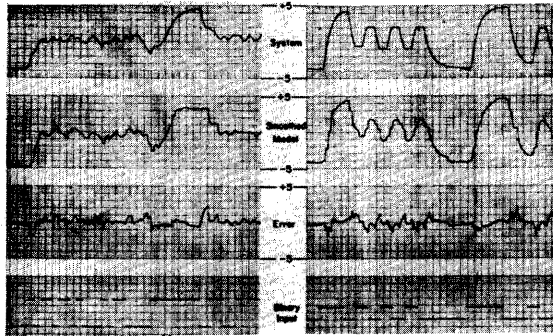


Figure 13. Modeling—System No. 1.

Figure 14. Modeling—System No. 4.

with a particular output is still valid. This philosophy has been successfully employed in a study of nonlinear process identification using feature detection of the input pattern.<sup>13</sup>

#### ACKNOWLEDGMENTS

The theory of this model was developed under grant NSG 244-62 from the National Aeronautics and Space Administration. Funds for the construction of the model were provided by the National Science Foundation under Rensselaer Polytechnic Institute Research Grant 635.04. The magnetic drum was donated by the International Business Machines Corporation. A complete description of the model, including circuit schematics, construction details, maintenance procedure, and timing diagrams, is presented in Reference 14.

#### REFERENCES

1. CHESTNUT, H., and WETMORE, G., "Predictive Control Applied to a Simple Position Control," General Electric Engineering Laboratory, Report No. 59 GL104 (1959).
2. CHESTNUT, H., DERUSSO, P., and TROUTMAN, P., "An Analog Simulation of a Predictive Control System," General Electric Engineering Laboratory, Report No. 59 GL219 (1959).
3. CHESTNUT, H., SOLLECITO, W., TROUTMAN, P., and GACEK, A., "Automatic Landing of Aircraft Using Predictive Control," General Electric Engineering Laboratory, Report No. 60 GL12 (1960).
4. CHESTNUT, H., SOLLECITO, W., and TROUTMAN, P., "Predictive Control System Applications," General Electric Engineering Laboratory, Report No. 60 GL119 (1960).
5. ZIEBOLZ, H., and PAYNTER, H., "Possibilities of a Two-Time Scale Computing System for Control and Simulation of Dynamic Systems," Proc. Nat. Electronics Conf., Vol. 9, pp. 215-223 (1953).
6. COALES, J., and NOTON, A., "An On-Off Servomechanism with Predicted Change-Over," Proc. IEE, Vol. 103, pt. 10, pp. 449-462 (1955).
7. BAUER, R. F., "A Predictive and Adaptive Control System," Master's thesis, Rensselaer Polytechnic Institute, Department of Electrical Engineering (1962).
8. DROSDECK, J. S., "Variable Interval Repetitive Prediction of Stochastic Signals,"

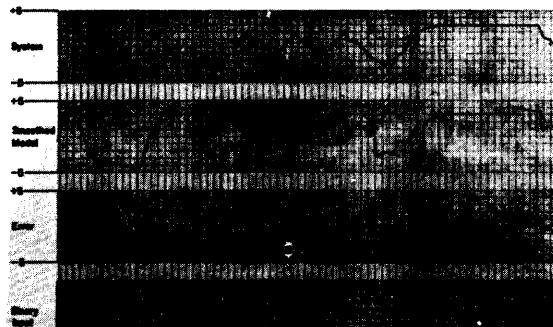


Figure 15. Adaptive Model.

- Master's thesis, Rensselaer Polytechnic Institute, Department of Electrical Engineering (1962).
9. KAUFMAN, H., "An Adaptive Predictive Control System for Random Signals," Master's thesis, Rensselaer Polytechnic Institute, Department of Electrical Engineering (1963).
  10. BOSE, A. G., "A Theory of Nonlinear Systems," Research Laboratory of Electronics, Mass. Inst. of Tech., Cambridge, Technical Report 309 (1956). (Wiener's original work remains in unpublished form.)
  11. ROY, R., MILLER, R. W., and DERUSSO, P. M., "An Adaptive Predictive Model for Nonlinear Processes with Two-Level Inputs," Fourth Joint Automatic Control Conference, University of Minnesota, Paper VIII-3, pp. 204-210 (1963). IEEE Transactions on Applications and Industry, Vol. 83, No. 72, pp. 173-178 (1964).
  12. ROY, R., and DERUSSO, P. M., "A Digital Orthogonal Model for Nonlinear Processes," IRE Transactions on Automatic Control, Vol. AC-7, No. 5, pp. 93-101 (1962).
  13. ROY, R., and MILLER, R. W., "Nonlinear Process Identification Using Statistical Pattern Matrices," Proceedings of the Fifth Joint Automatic Control Conference, Stanford University, California, pp. 349-354 (1964).
  14. DERUSSO, P. M., ROY, R., MILLER, R. W., and NUTTING, B. W., "Adaptive-Predictive Modeling of Nonlinear Processes," Final Report—National Aeronautics and Space Administration Grant NSG 244-62 (1963). (To be published as a NASA Report.)



# THE NEGATIVE GRADIENT METHOD EXTENDED TO THE COMPUTER PROGRAMMING OF SIMULTANEOUS SYSTEMS OF DIFFERENTIAL AND FINITE EQUATIONS

*Albert I. Talkin  
Harry Diamond Laboratories  
Washington, D. C.*

## I. INTRODUCTION

In programming a system of simultaneous nonlinear equations on an analog computer it is most convenient to use the equations in their implicit form. In addition since the equations are nonlinear certain partial derivatives may change sign resulting in computer instability unless the equations are programmed by the negative gradient method.<sup>1,2</sup> This paper considers programming nonlinear, explicitly time varying trajectory problems by coupling together a set of nonlinear positional equations and a set of nonlinear first order differential equations, each set being separately programmed by the negative gradient method.<sup>3</sup> The

<sup>1</sup> This paper draws no distinction between the terms "gradient method," "least squares," "steepest descent" or "transpose matrix method."

<sup>2</sup> For an interesting alternative see M. G. Bykhovski "Ultrastability in Electronic Computers when Realizing Nonlinear Equations in Implicit Form" IFAC Proceedings. Moscow 1960, Vol. 2, Butterworths 1961, pp. 1032-1038.

<sup>3</sup> The method described herein may be compared to that of Turner who considers both finite equations and differential equations in implicit form as "constraint" equations. His computer programming is done in a somewhat intuitive manner which does not necessarily guarantee stable programming. See R. M. Turner "On the Reduction of Error in Certain Analog Computer Calculations by the Use of Constraint Equations" Proceedings of the Western JCC, San Francisco, May 1960.

equation of the first approximation to the perturbed motion is used to examine the convergence of the computer program to the solution of the given mathematical system.

### 1.1 Definition of the Problem

The problem will be defined in n-dimensional space. Vector/matrix notation will be employed to reduce the labor of writing equations. The following definitions will be used:

- (1) position vector  $\mathbf{x} \equiv [x_1, x_2, \dots, x_n]$
- (2) i-th position function  $\phi_i \equiv \phi_i(\mathbf{x}, t)$ ;  
 $i = 1, \dots, m, m < n$
- (3) position function vector  $\phi \equiv [\phi_1, \phi_2, \dots, \phi_m]$
- (4) velocity vector  $\mathbf{u} \equiv [u_1, u_2, \dots, u_n]$
- (5) j-th velocity function  $f_j \equiv f_j(\mathbf{u}, \mathbf{x}, t)$ ;  
 $j = m + 1, m + 2, \dots, n$
- (6) velocity function vector  $\mathbf{f} \equiv [f_{m+1}, f_{m+2}, \dots, f_n]$
- (7) position error vector  $\mathbf{v} \equiv [v_1, v_2, \dots, v_n]$
- (8) velocity error vector  $\mathbf{a} \equiv [a_1, a_2, \dots, a_n]$
- (9) gradient vector  $\nabla_{\mathbf{x}} \equiv \left[ \frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \dots, \frac{\partial}{\partial x_n} \right]$

- (10) gradient vector  $\nabla_u \equiv \left[ \frac{\partial}{\partial u_1}, \frac{\partial}{\partial u_2}, \dots, \frac{\partial}{\partial u_n} \right]$
- (11) general matrix notation:  $A \equiv (a_{ij})$  and  $I \equiv$  unity matrix
- (12) scalar or inner product of two n-dimensional vectors  $(p, q)$ :  
 $(p, q) \equiv p_1q_1 + p_2q_2 + \dots + p_nq_n$ .
- (13)  $k, g$ , are arbitrary scalar quantities representing integrator gains
- (14)  $t$  is a parameter appearing in the mathematical problem statement i.e. "mathematical time"
- (15)  $\tau$  is computer time
- (16)  $\alpha$  is an arbitrary time scale factor  
 $0 < \alpha \leq 1$  ( $t = \alpha \tau$ )

Using the above definitions the problem can be stated as follows: Given the set of independent equations

$$\phi(x, t) = 0 \quad (1.1)$$

$$f(u, x, t) = 0 \quad (1.2) \quad (1)$$

$$u = x \equiv \frac{dx}{dt} \quad (1.3)$$

program an analog computer to generate the unknowns  $x(t)$ ,  $u(t)$ . Equation (1.1) represents  $m$  equations, (1.2) represents  $n-m$  equations and (1.3) represents  $n$  equations for a total of  $2n$  equations in  $2n$  unknowns  $x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_n$ . A full statement of the problem also requires that  $n-m$  of the position coordinates be specified as initial conditions. Before continuing with the analysis of such a system, it may be helpful to expand upon the above problem statement by considering some examples and applications in two and three dimensional space.

## 1.2 Examples and Applications

In this section vector notation is momentarily abandoned in favor of the more conventional  $x, y, z$  notation. Also  $u$  is eliminated from the equations by the substitution  $x = u$ .

Consider the three dimensional system (2):

$$\begin{aligned} \phi_1(x, y, z) &\equiv x^2 + y^2 - c^2z^2 = 0 \\ \phi_2(x, y, z) &\equiv \alpha x + \beta y + \gamma z + h = 0 \\ f_3(x, y, z, x, y, z) &\equiv x^2 + y^2 + z^2 - s^2 = 0 \end{aligned} \quad (2)$$

( $c, \alpha, \beta, \gamma, h, s$  are arbitrary constants)

The simultaneous solution of (2) is a conic section traced at constant speed  $s$ . In (2) the variable  $t$  does not appear explicitly in  $\phi$  or  $f$ .

Consider the two dimensional system (3):

$$\begin{aligned} \phi_1(x, y, t) &\equiv x^2 + y^2 - r^2(t) = 0 \\ f_2(x, y, x, y, t) &\equiv x^2 + y^2 - s^2(t) = 0 \end{aligned} \quad (3)$$

If  $r(t)$  is constant and  $s(t)$  is a linear ramp, (3) represents linear frequency modulation without any amplitude modulation. If both  $r(t)$  and  $s(t)$  are constant, (3) represents an oscillator with highly stable amplitude and frequency characteristics. If  $r(t)$  is constant and  $s(t) = \frac{d\theta}{dt}$ , (3) performs trigonometric resolution.<sup>4</sup> If  $s(t)$  is a constant, (3) represents a complex modulation scheme obeying the law (AM)  $\times$  (FM) = constant. The modulation intelligence  $r(t)$  can be received by either an AM or FM receiver and is redundant.

Consider the system (4):

$$\begin{aligned} \phi_1(x, y, t) &\equiv x^2 + y^2 - r^2(t) = 0 \\ f_2(x, y, x, y, t) &\equiv x^2 + y^2 - r^2(t)s^2(t) = 0 \end{aligned} \quad (4)$$

If in (4) both  $r(t)$  and  $s(t)$  vary independently, the system represents a simultaneous AM and FM waveform with independent messages  $r(t)$  and  $s(t)$ . As a final example consider the system (5) for the simulation of planetary motion in two dimensions. Equation 5.2 involving mixed position and velocity coordinates is a result of the requirement that equal areas be swept in equal times.

$$b^2(x-h)^2 + a^2y^2 - a^2b^2 = 0 \quad (5.1)$$

$$xy - yx - c = 0 \quad (5.2) \quad (5)$$

## II. ANALYSIS

### 2.1 Partitioning the Problem

Consider first the situation in which the analog computer is in the "hold" or "reset" mode, i.e. switches  $S_1$  and  $S_2$  of Fig. 1 are open. In "reset" correct stationary values of  $x$  and  $u$  must be generated. Since  $\phi$  is not a function

<sup>4</sup>R. M. Howe and E. G. Gilbert, "Trigonometric Resolution in Analog Computers by Means of Multiplier Elements," IRE Transactions, Vol. EC 6, June 1957.



of  $u$ , it is possible to generate  $x$  by programming the equations

$$\phi_1(x, t_0) = 0, \phi_2(x, t_0) = 0, \dots, \phi_m(x, t_0) = 0 \quad (6)$$

with  $n-m$  of the coordinates of  $x$  specified as initial conditions. This, of course, results in a system of  $m$  independent equations with  $m$  unknowns. Since these equations are in general nonlinear the gradient method of programming is used to guarantee convergence.

Turning now to the generation of  $u$ , the condition  $f = 0$  provides only  $n-m$  equations to be solved for  $n$  unknowns. To obtain the  $m$  additional equations an augmented velocity function vector ( $\psi$ ) must be defined. In vector notation,

$$\psi = \frac{d\phi}{dt} + f \quad (7)$$

For the rigorous interpretation of (7) the original definitions of  $f$  and  $\phi$  must be expanded to  $n$ -dimensions, with the first  $m$  components of  $f$  being identically zero, and the last  $n-m$  components of  $\phi$  being identically zero. Written out:

$$\psi \equiv \left[ \frac{d\phi_1}{dt}, \frac{d\phi_2}{dt}, \dots, \frac{d\phi_m}{dt}, f_{m+1}, f_{m+2}, \dots, f_n \right] \quad (8)$$

If in the first  $m$  coordinates of  $\psi$  the substitution of  $u$  for  $x$  is made, then

$$\psi = 0 \quad (9)$$

represents a system of  $n$  equations in  $n$  unknowns  $u_1, u_2, \dots, u_n$ . The system (9) can be programmed by the gradient method, and requires that  $x$  developed from (6) be inserted as

a parameter. The conditions obtained in "reset" are

$$\phi(x, t_0) = 0, \psi(u, x, t_0) = 0 \quad (10)$$

Equations (10) represent the partitioned system. Note that in "reset" ( $n-m$ ) of the  $x$  integrators have initial conditions imposed and the remaining  $m$  coordinates are determined uniquely. In "hold" the  $x$  coordinates are no longer determined uniquely and there will be some tendency for the stationary values to be sensitive to perturbations of the position vector.

2.2 Closing the Switches—(The "compute" mode)

Fig. 1 is a simplified schematic illustrating the connections for the  $i$ -th component of  $x$  and  $u$ . In "reset" or "hold" the switches  $S_1$  and  $S_2$  are open and the negative gradient programming guarantees that  $x$  and  $u$  will assume stationary values as quickly as possible. It is reasonable to suppose that if the switches were closed the resulting system would produce a very close approximation to the desired trajectory provided that the ensuing continuous iteration converges.

2.3 Perturbation Analysis

The following two assumptions are made:

- (1) The conditions  $\phi = 0$  and  $\psi = 0$  are approximately satisfied.
- (2) Arbitrary small instantaneous perturbations  $\delta x$  and  $\delta u$  are imposed on the system.

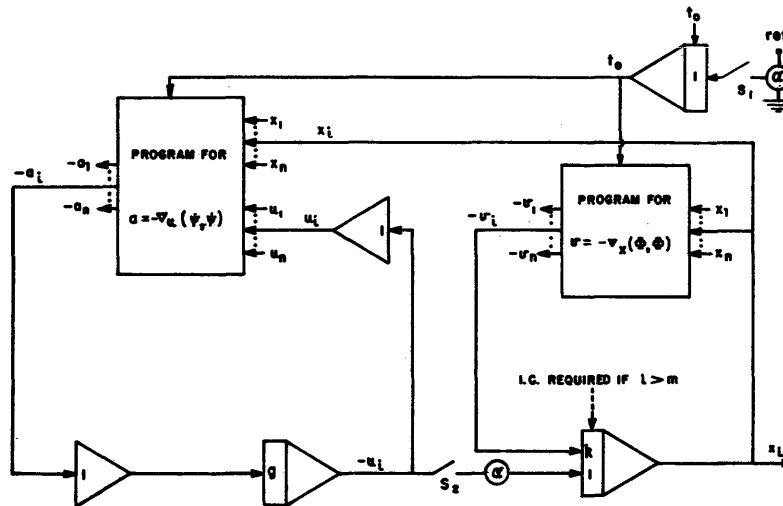


Figure 1. Simplified Computer Diagram Showing Connections for  $x_i, u_i$ .

In accordance with the gradient method, let

$$\mathbf{v} = -\nabla_{\mathbf{x}}(\phi, \phi) \quad (11)$$

$$\mathbf{a} = -\nabla_{\mathbf{u}}(\psi, \psi) \quad (12)$$

From fig. 1

$$\frac{d\mathbf{x}}{d\tau} = \alpha\mathbf{u} + k\mathbf{v} \quad (13)$$

$$\frac{d\mathbf{u}}{d\tau} = g\mathbf{a} \quad (14)$$

Differentiating (13) and substituting (14):

$$\frac{d^2\mathbf{x}}{d\tau^2} = \alpha g\mathbf{a} + k\frac{d\mathbf{v}}{d\tau} \quad (15)$$

Operating on equations (13) and (15) with the variational operator  $\delta$ :

$$\frac{d(\delta\mathbf{x})}{d\tau} = \alpha(\delta\mathbf{u}) + k(\delta\mathbf{v}) \quad (16)$$

$$\frac{d^2(\delta\mathbf{x})}{d\tau^2} = \alpha g(\delta\mathbf{a}) + k\frac{d(\delta\mathbf{v})}{d\tau} \quad (17)$$

It is now necessary to express  $\delta\mathbf{v}$  and  $\delta\mathbf{a}$  in terms of  $\delta\mathbf{x}$ . Taking variations of both sides of (11):

$$\begin{aligned} \delta v_i = & \sum_{j=1}^n \left( -2 \sum_{k=1}^m \frac{\partial \phi_k}{\partial x_i} \frac{\partial \phi_k}{\partial x_j} \right) \delta x_j \\ & + \sum_{j=1}^n \left( -2 \sum_{k=1}^m \phi_k \frac{\partial^2 \phi_k}{\partial x_i \partial x_j} \right) \delta x_j \quad (18) \end{aligned}$$

As variations are taken starting from an assumed equilibrium state  $\phi \approx 0$ , the second term on the right in (18) vanishes. The matrix

$$\mathbf{A} \equiv (a_{ij}) \equiv \left( 2 \sum_{k=1}^m \frac{\partial \phi_k}{\partial x_i} \frac{\partial \phi_k}{\partial x_j} \right)$$

is positive definite for it can be factored into the product of a matrix by its transpose. Equation (18) then becomes

$$\delta\mathbf{v} = -\mathbf{A} \delta\mathbf{x} \quad (19)$$

As  $\phi$  is not a function of  $\mathbf{u}$ , only the variation with respect to  $\mathbf{x}$  need be considered in (18). To find  $\delta\mathbf{a}$ , variations of both sides of (12) are taken, noting now that  $\delta\mathbf{u}$  and  $\delta\mathbf{x}$  contribute to  $\delta\mathbf{a}$ : thus,

$$\delta a_i = \sum_{j=1}^n \left( -2 \sum_{k=1}^n \frac{\partial \psi_k}{\partial u_i} \frac{\partial \psi_k}{\partial u_j} \right) \delta u_j$$

$$+ \sum_{j=1}^n \left( -2 \sum_{k=1}^n \frac{\partial \psi_k}{\partial u_i} \frac{\partial \psi_k}{\partial x_j} \right) \delta x_j \quad (20)$$

In (20) terms involving second partial derivatives are absent because of the assumption that  $\psi \approx 0$ .

Rewriting (20)

$$\delta\mathbf{a} = -\mathbf{B}\delta\mathbf{u} - \mathbf{H}\delta\mathbf{x} \quad (21)$$

where

$$\mathbf{B} \equiv (b_{ij}) \equiv \left( \sum_{k=1}^n 2 \frac{\partial \psi_k}{\partial u_i} \frac{\partial \psi_k}{\partial u_j} \right)$$

$$\mathbf{H} \equiv (h_{ij}) \equiv \left( \sum_{k=1}^n 2 \frac{\partial \psi_k}{\partial u_i} \frac{\partial \psi_k}{\partial x_j} \right)$$

The matrix  $\mathbf{B}$  is positive definite, but the matrix  $\mathbf{H}$  is not so restricted. Substituting from (19) and (21) in (17) gives:

$$\begin{aligned} \frac{d^2(\delta\mathbf{x})}{d\tau^2} + \alpha g\mathbf{B}\delta\mathbf{u} + \alpha g\mathbf{H}\delta\mathbf{x} + \\ k\frac{d(\mathbf{A}\delta\mathbf{x})}{d\tau} = 0 \quad (22) \end{aligned}$$

$$\begin{aligned} \frac{d^2(\delta\mathbf{x})}{d\tau^2} + \alpha g\mathbf{B}\delta\mathbf{u} + \alpha g\mathbf{H}\delta\mathbf{x} + \\ k\frac{\partial \mathbf{A}}{\partial \tau} \delta\mathbf{x} + k\mathbf{A}\frac{d(\delta\mathbf{x})}{d\tau} = 0 \quad (23) \end{aligned}$$

From (16) and (19),

$$\alpha\delta\mathbf{u} = k\mathbf{A}\delta\mathbf{x} + \frac{d(\delta\mathbf{x})}{d\tau} \quad (24)$$

Substituting (24) in (23) gives:

$$\begin{aligned} \mathbf{I} \frac{d^2(\delta\mathbf{x})}{d\tau^2} + (k\mathbf{A} + g\mathbf{B}) \frac{d(\delta\mathbf{x})}{d\tau} + \\ (gk\mathbf{B}\mathbf{A} + \alpha g\mathbf{H} + k\frac{\partial \mathbf{A}}{\partial \tau}) \delta\mathbf{x} = 0 \quad (25) \end{aligned}$$

Equation (25) is the equation of the first approximation of the perturbed motion. Note that  $(\mathbf{I})$ , the unity matrix, is positive definite and  $(k\mathbf{A} + g\mathbf{B})$  is positive definite, as it is the sum of two positive definite matrices  $k\mathbf{A}$  and  $g\mathbf{B}$ .

Now the third term on the left in (25) can be written as the matrix sum

<sup>5</sup> Bellman, R. "Introduction to Matrix Analysis," McGraw-Hill Book Company, Inc., New York, N.Y., 1960, p. 246.

$$gkBA + \alpha \left( gH + k \frac{\partial A}{\partial \tau} \right) \quad (26)$$

noting that  $\frac{\partial A}{\partial \tau} = \alpha \frac{\partial A}{\partial t}$

The matrix product  $gkBA$  is positive definite, and the term  $\alpha \left( gH + k \frac{\partial A}{\partial t} \right)$  may be considered a perturbation term which can be made small by reducing  $\alpha$ . Hence eq. (25) represents a stable system, all coefficient matrices being essentially positive definite. Bellman<sup>4</sup> indicates that this result also holds when the coefficient matrices are time varying as in this case.

#### CONCLUSION

It is now evident that the time scale factor  $\alpha$  can be reduced whenever the perturbation term grows too large. This may be done

selectively so that the harder to stabilize segments of the trajectory are traced at a slower rate. It may sometimes occur that the required  $\alpha$  is prohibitively small, *i.e.*, the solution time is exorbitant, or perhaps it is required not to alter  $\alpha$  during a trajectory. In this case a trade off between the scalar gains  $g$  and  $k$  must be made, *i.e.*, some gain modulation scheme may be used to minimize the sum

$gH + k \frac{\partial A}{\partial t}$  while holding the product  $gk \geq M$ .

#### ACKNOWLEDGMENT

The problem generalization and analysis in this report was inspired by the work of Arthur Hausner of these laboratories, who intuitively used this method to program a special problem proposed by the author.



# QUANTIZING AND SAMPLING ERRORS IN HYBRID COMPUTATION

*C. R. Walli*  
*Space and Information Systems Division*  
*North American Aviation, Inc.*  
*Downey, California*

## INTRODUCTION

Quantizing and sampling errors have been examined by a number of authors. For example, the errors have been extensively studied in the communication<sup>2, 6, 7</sup> and servomechanisms<sup>4, 5, 11</sup> fields of electrical engineering. Studies of these errors have also been made by numerical analysts,<sup>1, 3</sup> primarily from the standpoint of round-off error and error propagation. None of these efforts, however, has been specifically directed to the analysis of the total errors generated in a hybrid computation loop. In each field, different methods have been used and different portions of the problem studied.

In the communications field, sampling and quantizing have been studied in connection with PCM signals. The major effort has been associated with the transmission of intelligible audio and visual data. Three to seven (binary) bits have been used for the most part, because this range is adequate for communications work. For example, when four bits (16 levels) are used to code the intensity of a transmitted television picture, good picture quality is obtained, and even a fair picture is achieved with three-bit (8-level) transmission.

In these applications, the percentage error as a function of signal amplitude is improved by

companding (compressing) the signal amplitude prior to quantizing and expanding it upon conversion. The most frequently used techniques are sinusoidal frequency response and stochastic variable frequency response.

In the servo field, attention has been concentrated on the low-bit quantizing problem (normally one to two bits) in applications which do not use a digital computer for control.<sup>11</sup> Sampling is not usually used in the case of servos with only a few bits, and, when a digital computer is used for control, slight emphasis has been placed upon the quantization errors. The techniques used in these studies have been those of the phase plane, state variables, sampled data system theory, etc.

In the case of numerical analysis, the statistical approach has been used in conjunction with the Taylor series for the estimate of the round-off errors at each step. Primary consideration has been given to open-loop multiplication and addition errors. However, the presence of predictor-corrector-modifier formulae testifies to the existence of closed-loop error analyses. Frequency response techniques have not been used.

A statistical approach<sup>4, 5</sup> has been used to describe the quantization errors by means of the errors in probability distributions resulting

from quantization. This approach is more related to the numerical analysis approach than to the others, since the harmonic content of the signal is not treated.

The sampling and execution time errors in a hybrid loop have been studied and simulated on analog computers, but quantization has been neglected.<sup>13, 14</sup>

A typical hybrid computation loop consists of an analog computer, a digital computer, and conversion equipment. When this loop is viewed from the analog computer, the portion of the loop outside the analog computer can be considered to be a black box that operates on the signals sent to it, adds errors, and then sends the signals back to it, as indicated in Figure 1.

The system state vectors,  $X$ ,  $Y$ ,  $Z$ , and  $E$ , are described by

$$\begin{aligned} Z(t) &= Y(t) + E(t) \\ Y(t) &= Y(X(t), t) \end{aligned}$$

The vector function  $Y(X(t), t)$  specifies the operations of the processor, and the error vector  $E(t)$  is a function of numerical computational errors and of the conversion errors. The errors of major interest to this paper are those generated by the combined conversion processes. The other errors will not be discussed because their consideration can easily be added. For example, round-off errors should be negligible if computer word lengths are 20 to 24

bits, and a finite digital computer processing time,  $\tau$ , represents only a pure time delay of  $\tau$  seconds.

What kind of signals in the hybrid loop will most likely be affected by the quantization and sampling errors? Certainly sinusoidal signals should have the highest probability in this regard, since physical plants either have inherent natural frequencies, or gain them when a control loop is added. Even for nonlinear processes, the system variables can usually be decomposed into a Fourier series. Thus, the choice of characterizing the "Block Box" errors in terms of the errors generated in response to a sine wave is a logical one.

The required precision can be easily estimated. Individual analog components have errors on the order of 0.01 percent, and the maximum voltage is 100 volts. The maximum precision expected of the converters, therefore, would be 13 bits plus sign, since the error maximum is one-half the step size. If the maximum represents 128 volts, a 100-volt signal would use 12.6 bits and a 1-volt signal 6 bits (plus sign, in both cases). In practice, the signals of concern are those whose amplitude may be 1 bit—or 13 bits. Thus, the full range must be considered.

In what way can past research, carried on in the areas previously mentioned, be applied to the hybrid computation problem? In particular, for any specified error bound, how finely should the signal be quantized, how rapidly should it be sampled, and what trade-offs exist between sampling rate and the number of quantization levels? How does the presence of noise on the input signal affect the errors? In short, is there an optimum combination that can be used, or must the maximum possible number of levels be used with the maximum possible sampling rate?

Results from each of the previously mentioned areas should of course be utilized. The work accomplished in the communications field appears to be most applicable because of the major use of harmonic analysis. Some differences should be expected, however, since the signal band is less important than the phase-shift errors to the present problem.

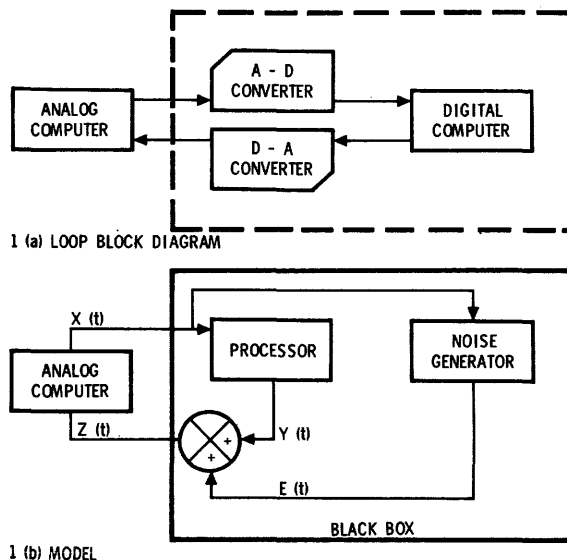


Figure 1. Hybrid Computation Loop.

QUANTIZING

It is readily apparent from Figure 2 that either the sampling or the quantization can be thought of as occurring first. Because interest here is in a number output from the converter, the number will be the same in either case, and this will be true for non-uniform quantizing and non-uniform sampling. The digital-analog conversion process can be represented simply by some holding circuit.

The quantizer characteristic of Figure 3 can readily be decomposed into the two functions shown in Figure 4. From Figure 4, the following equations can be written

$$Q(x) = x + e(x) \tag{1}$$

$$e(x) = \sum_{n=1}^{\infty} e_n \sin \frac{2\pi n}{q} x \tag{2}$$

$$e_n = \frac{2}{q} \int_{-q/2}^{q/2} (-x) \sin \frac{2\pi n}{q} x = \frac{q}{\pi} \frac{(-1)^n}{n} \tag{3}$$

therefore

$$Q(x) = x + \frac{q}{\pi} \sum_{n=1}^{\infty} \frac{(-1)^n}{n} \sin \frac{2\pi n}{q} x \tag{4}$$

Thus, an infinite range of "amplitude frequencies" exists. The response to any specific input can be obtained by direct substitution into Equation 4. The harmonics at  $\frac{2\pi n}{q}$  provide an insight into the reason for the repeated spectra at intervals of  $\frac{2\pi}{q}$  in the  $\alpha$  domain, which is given by the statistical approach.

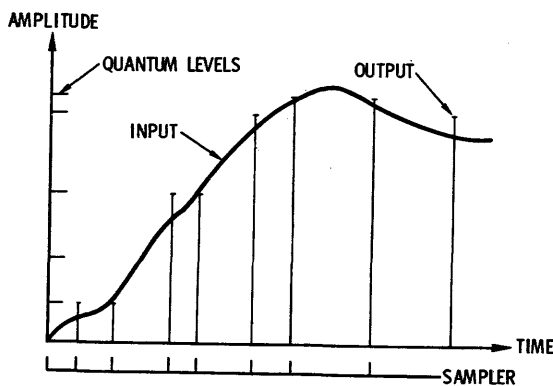


Figure 2. Sampling and Quantizing.

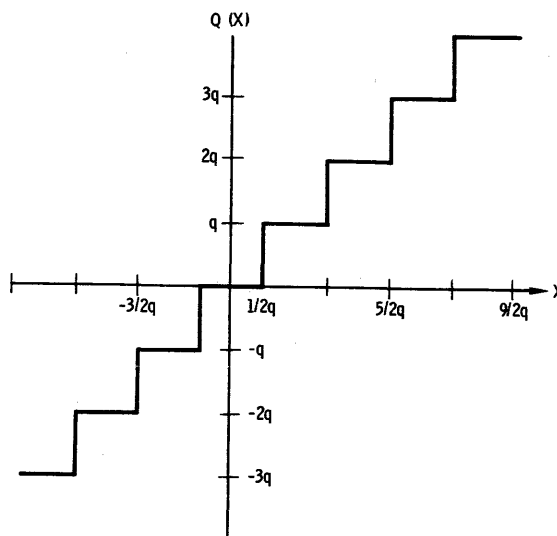


Figure 3. Quantizer.

THE STATISTICAL APPROACH TO QUANTIZATION ERRORS

The statistical approach to quantization errors is well described in the literature.<sup>4,5</sup> Therefore, only a few of the concepts involved in this method will be given here.

The quantizer operates on the signal amplitude of its input rather than on the signal frequency. It is logical to characterize the quantizer performance in terms of the input amplitude. In order to provide a general characterization similar to the standard frequency domain characterization of time signals, the

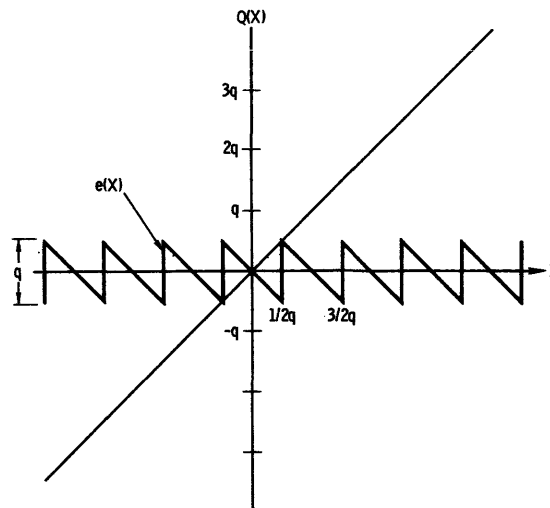


Figure 4. Decomposed Characteristics.

probability density distribution of the signal amplitude must be considered. The Fourier transform (characteristic function) of the probability density can then be defined in terms of a frequency-like variable, usually denoted by  $\alpha$ .

Using these concepts, the following results are obtained, where  $q$  is the quantization level.

Let  $p(f(t))$  denote the probability density of the input signal  $f(t)$  and  $P(\alpha)$  denote the Fourier transform of  $p(f)$ :

1. If  $P(\alpha) = 0$  for  $|\alpha| > \frac{\pi}{q}$ , then  $p(f)$  can be completely recovered from the quantized signal.
2. If  $P(\alpha) \neq 0$  for  $|\alpha| > \frac{\pi}{q}$ , then  $p(f)$  cannot be recovered, since erroneous signals will be introduced into the data.
3. The quantized signal in the  $\alpha$  domain has the basic Fourier transform  $P(\alpha)$  repeated at intervals of  $\frac{2\pi}{q}$  in similar fashion to the frequency spectrum of a time-sampled signal.
4. The quantizing error may be assumed to be uniformly distributed between  $+1/2 q$  and  $-1/2 q$ . In this case, the mean square quantizing error is given by  $1/12 q^2$ .

The expression for the quantizing error may readily be derived from a simple viewpoint. If the quantizing error (Figure 4) is considered, it is seen to be linear between  $-q/2$  to  $+q/2$  and to be repeated along the  $x$  axis. Letting  $\langle \rangle$  denote the expected value and assuming that the input amplitude probability distribution function is linear, then

$$\langle e^2 \rangle = \frac{2}{q} \int_{-q/2}^{q/2} (-x)^2 dx = \frac{1}{q} \frac{2}{3} (x^3) \Big|_{-q/2}^{+q/2} = \frac{1}{12} q^2$$

It should be noted that the results of this statistical approach to quantizing do not indicate relationships in terms of the input signal. Reference is made only to probability distribution.

Typical signal characteristics in the time, probability, and  $\alpha$  domains are given in Figure 5. The probability distribution is "area sam-

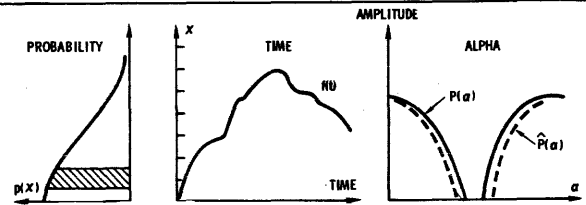


Figure 5. Quantized Signal Characteristics.

pled" by the quantizer, which is equivalent to a square pulse. The Fourier transform of this is a  $\frac{\sin x}{x}$  term which, when multiplied by  $P(\alpha)$ , gives  $P(\alpha)$ .

### SINGLE SINE-WAVE INPUT

There are two simple reasons for concentrating attention upon a single sine-wave input to the quantizer: (1) the response and implications of the spectral distortion resulting from the quantizing action can readily be visualized, and (2) in many of the applications for hybrid computers there will exist simple natural frequencies in the hybrid loop imposed by the problem under consideration, e.g., the short period response of an aircraft or missile, or its equivalent in a space vehicle. Many of the physical systems which warrant studying on a hybrid computer are of the type which have a small number of natural frequencies. Thus, the system energy will be concentrated there, rather than throughout a broad band of frequencies, as would be the case if stochastic variables alone were present. An additional reason for concentrating attention upon single sine-wave input is that testing and analysis of test data are simplified for simple sine waves.

The input to the quantizer that will be used is

$$x(t) = A \sin \omega t$$

### SINE WAVE PROBABILITY DENSITY

The definition of the probability distribution function is

$$P(X) = \text{probability that } x \leq X$$

Figure 6 shows the sine wave and the level  $X$ .

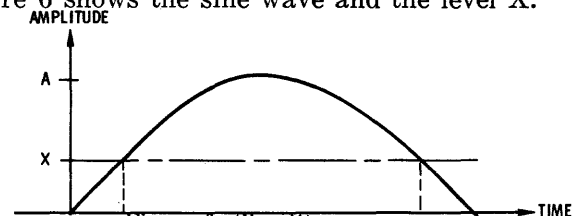


Figure 6. Sine Wave.



From Figure 6, the following can be written

$$\begin{aligned}
 P(X) &= \frac{1}{2} + \frac{1}{\pi} \sin^{-1} \frac{X}{A} & -A \leq X \leq A \\
 &= 0 & X \leq -A \\
 &= 1 & A \leq X \quad (5)
 \end{aligned}$$

The probability density is the derivative of Equation 5

$$\begin{aligned}
 p(X) &= \frac{1}{\pi A} \sqrt{1 - \left(\frac{X}{A}\right)^2} & |X| < A \\
 &= 0 & |X| > A \quad (6)
 \end{aligned}$$

These are plotted in Figure 7.

It is evident that the Fourier transform of  $p(x)$  will not be band limited in the  $\alpha$  domain.

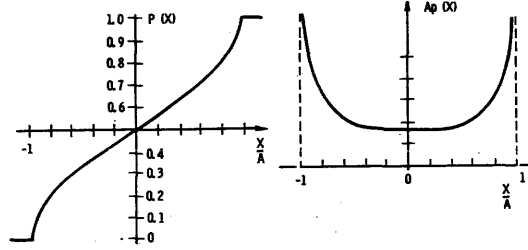


Figure 7. Sine Wave Probability Distribution and Probability Density.

Thus, the quantization theorem reveals that this probability density cannot be recovered after quantization, but does not disclose what will be recovered.

THE TIME DOMAIN

Substituting into Equation 4, we have

$$y(t) = A \sin \omega_0 t + \frac{q}{\pi} \sum_{n=1}^{\infty} \frac{(-1)^n}{n} \sin \left[ \frac{2\pi n A}{q} \sin \omega_0 t \right] \quad (7)$$

but

$$\sin(z \sin \omega t) = 2 \sum_{r=0}^{\infty} J_{2r+1}(z) \sin(2r+1)\omega t \quad (8)$$

therefore

$$y(t) = A \sin \omega_0 t + \frac{2q}{\pi} \sum_{n=1}^{\infty} \sum_{r=0}^{\infty} \frac{(-1)^n}{n} J_{2r+1} \left( \frac{2\pi n A}{q} \right) \sin(2r+1)\omega_0 t$$

or rewriting slightly and interchanging summations

$$y(t) = A \sin \omega_0 t + \frac{2q}{\pi} \sum_{r=0}^{\infty} \left\{ \sum_{n=1}^{\infty} \frac{(-1)^n}{n} J_{2r+1} \left( \frac{2\pi n A}{q} \right) \right\} \sin(2r+1)\omega_0 t \quad (9)$$

It can be seen, therefore, that the quantizer has generated all of the odd harmonics for the input sine wave. If the input signal did not have zero phase, then all the even harmonics would also be generated since in this case there are terms of the form

$$\begin{aligned}
 \sin(z \sin(\omega t + \theta)) &= \\
 &\sin(z \cos \theta \sin \omega t + z \sin \theta \cos \omega t)
 \end{aligned}$$

which is equal to

$$\begin{aligned}
 &\sin(a \sin \omega t) \cos(b \cos \omega t) + \\
 &\cos(a \sin \omega t) \sin(b \cos \omega t)
 \end{aligned}$$

and, for example

$$\cos(a \sin \omega t) = 2 \sum J_{2r}(a) \sin 2r\omega t$$

Because of the product terms, sums and differences of all frequencies are generated. Since there are only harmonics in this signal, however, the sums and differences are again harmonics. If two sine waves with different frequencies constitute the input signal, the above relationships will generate all sum and difference harmonics.

The amplitude of the harmonic terms can be found by expanding Equation 8 into the following

$$y(t) = \sum_{r=0}^{\infty} B_{2r+1} \sin(2r+1)\omega_0 t$$

$$B_1 = A \left[ 1 + \frac{2q}{\pi A} \sum_{n=1}^{\infty} \frac{(-1)^n}{n} J_1 \left( \frac{2\pi n A}{q} \right) \right] \tag{10}$$

$$B_3 = A \left[ \frac{2q}{\pi A} \sum_{n=1}^{\infty} \frac{(-1)^n}{n} J_3 \left( \frac{2\pi n A}{q} \right) \right], \text{ etc.}$$

$$e_1 = \frac{2q}{\pi A} \sum_{n=1}^{\infty} \frac{(-1)^n}{n} J_1 \left( \frac{2\pi n A}{q} \right) \approx \frac{2}{\pi^2} \left( \frac{q}{A} \right)^{3/2} \sum_{n=1}^{\infty} \frac{(-1)^n}{n^{3/2}} \sin \left( \pi \frac{2nA}{q} - \frac{3}{4} \right) \tag{12}$$

In the asymptotic expansion, each successive harmonic adds  $-\pi$  to the angle. This merely multiplies the term by  $-1$ . The following approximation can then be used:

$$y(t) \approx A (1 + e_1) \sin \omega_0 t + A e_1 \sum_{r=1}^{\infty} (-1)^{2r+1} \sin (2r + 1) \omega_0 t \tag{13}$$

This, however, does not apply to all harmonics, because infinite power would be required. Now, the recursion formula gives

$$J_{r+1}(x) = \frac{r}{x} J_r(x) - J_{r-1}(x) \tag{14}$$

Thus, the relationship found above

$$J_{r+1}(x) \sim -J_{r-1}(x) \text{ (odd values)} \tag{15}$$

applies only for

$$x \gg r \tag{16}$$

In a conventional hybrid computer, the number of levels available will be on the order of 1,000 to 10,000. Thus, the previous approximations should be valid for at least the first ten harmonics for large input signals. For small input signals (which can still occur), the approximation cannot be used.

It can be concluded that a *very* large number of harmonics have amplitudes of the same order of magnitude. Thus, a large number of terms in the summation are required to find the harmonic amplitude—even for the fundamental component of the error. Clearly, some other method to find the error would be desirable.

**FOURIER SERIES EXPANSION**

An explicit approach can be used to find the total harmonic amplitude. The output of the quantizer has a Fourier series which is given by

$$y(t) = \sum_{r=0}^{\infty} B_{2r+1} \sin (2r + 1) \omega_0 t$$

Let us assume that the argument of the Bessel function,  $2\pi n A/q$ , is assumed to be large enough to permit the asymptotic expansion to be used.

For large  $x$

$$J_{2r+1}(x) \sim \sqrt{\frac{2}{\pi x}} \sin \left( x - (2r + 1) \frac{\pi}{2} - \frac{\pi}{4} \right) \tag{11}$$

Consider the error term in the fundamental

But  $y(t)$  can be expanded directly in a Fourier series. In Figure 8, horizontal slices are used to characterize the quantized sine wave. The Fourier series can then be written directly as

$$B_{2r+1} = \frac{4}{\pi} \int_0^{\pi/2} y(\theta) \sin (2r + 1) \theta d\theta$$

$$B_{2r+1} = \frac{4}{\pi} \left\{ q \int_{\theta_1}^{\pi/2} d\theta + q \int_{\theta_2}^{\pi/2} d\theta + q \int_{\theta_3}^{\pi/2} d\theta + \dots \right\}$$

$$\sin (2r + 1) = \frac{4q}{\pi} \sum \frac{\cos (2r + 1) \theta_i}{2r + 1}$$

$$y(t) = \sum_{r=0}^{\infty} \left\{ \frac{4q}{\pi} \sum_{i=1}^n \frac{\cos (2r + 1) \theta_i}{2r + 1} \right\} \sin (2r + 1) \omega_0 t \tag{17}$$

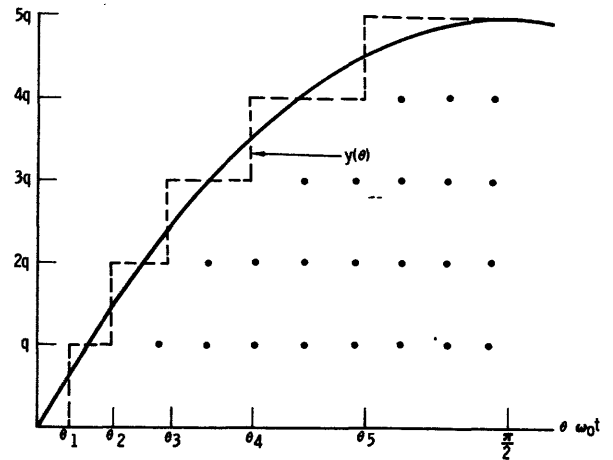


Figure 8. Quantized Sine Wave—Horizontal.

Now

$$\theta_i = \sin^{-1} (2i - 1) \frac{q}{2A} \quad (18)$$

$$N = \frac{A}{q} \text{ rounded to nearest integer}$$

Therefore

$$B_{2r+1} = \frac{4q}{\pi} \sum_{i=1}^{\infty} \cos (2r + 1) \frac{\sin^{-1} (2i - 1) \frac{q}{2A}}{2r + 1} \quad (19)$$

and

$$y(t) = \frac{4q}{\pi} \sum_{r=0}^{\infty} \left[ \sum_{i=1}^N \cos \left[ (2r + 1) \frac{\sin^{-1} (2i - 1) \frac{q}{2A}}{2r + 1} \right] \right] \sin (2r + 1) \omega_0 t \quad (20)$$

Only N terms are required ( $N = A/q$ ) for the determination of the coefficients of the Fourier series. For N not too large, this is a convenient formula to use in practice. It would be difficult to compute the total output power or harmonic distortion. Returning to the original picture of

the process, we this time square the output signal.

It is readily seen from Figure 9 that the power for the output of the quantizer is given by

$$\langle y^2 \rangle = \frac{A^2}{2\pi} \left\{ 4 \left( \frac{q}{A} \right)^2 (\theta_2 - \theta_1) + \dots + 4 \left\{ \left( \frac{(N-1)q}{A} \right)^2 (\theta_N - \theta_{N-1}) + 4 \left( \frac{Nq}{A} \right)^2 \left( \frac{\pi}{2} - \theta_N \right) \right\} \right\} \quad (21)$$

$$\langle y^2 \rangle = \frac{2}{\pi} q^2 \left[ \frac{\pi}{2} N^2 - \sum_{i=1}^N \theta_i \left[ i^2 - (i-1)^2 \right] \right] \quad (22)$$

$$\langle y^2 \rangle = q^2 \left[ N^2 - \frac{2}{\pi} \sum_{i=1}^N (2i-1) \sin^{-1} \left( i - \frac{1}{2} \right) \frac{q}{A} \right] \quad (23)$$

Now, the input power is given by

$$\langle x^2 \rangle = \frac{1}{2} A^2 \quad (24)$$

Thus, the ratio of the output power to input power of the quantizer is

$$\frac{\langle y^2 \rangle}{\langle x^2 \rangle} = 2 \left( \frac{q}{A} \right)^2 \left[ N^2 - \frac{2}{\pi} \sum_{i=1}^N (2i-1) \sin^{-1} \left( i - \frac{1}{2} \right) \frac{q}{A} \right] \quad (25)$$

The harmonic distortion is defined as

$$\text{D. F.} = \sqrt{\frac{\text{Sum of squares of harmonics}}{\text{Square of the fundamental}}} \times 100\%$$

Now

$$\langle y^2 \rangle = \frac{1}{2} \sum_{r=0}^{\infty} B_{2r+1}^2 = \sum (\text{squares of all components})$$

Therefore

$$D. F. = \frac{1}{B_1} \sqrt{2 \langle y^2 \rangle - B_1^2} \quad (26)$$

where  $B_1$  is the amplitude of the fundamental

$$B_1 = \frac{4q}{\pi} \sum_{i=1}^n \cos \sin^{-1} \left( i - \frac{1}{2} \right) \frac{q}{A} \quad (27)$$

Numerical results obtained from the preceding formulas are shown in Table I.

The last column in Table I represents the estimated distortion factor. This was obtained

$A = q$	$B_1$	$B_3$	$B_5$	$B_7$	$B_9$	$B_{11}$	$B_{13}$	$B_{15}$
	110%	0	-22%	-16%	0	10%	9%	0

The amplitudes do not decay rapidly, and there is the tendency for alternating signs, described by Equation 13.

THE FREQUENCY DOMAIN

Equation 9 can be re-written using exponentials and  $\omega_0$  for the input frequency (replacing  $\sin \theta$  by  $\frac{1}{2j} (e^{j\theta} - e^{-j\theta})$ ).

$$y(t) = \frac{1}{2j} A (e^{j\omega_0 t} - e^{-j\omega_0 t}) + \frac{q}{\pi j} \sum_{r=-\infty}^{+\infty} \quad (28)$$

$$\left[ \sum_{n=1}^{\infty} \frac{(-1)^n}{n} J_{2r+1} \left( \frac{2\pi n A}{q} \right) \right] e^{j(2r+1)\omega_0 t}$$

from the estimated mean squared error,  $1/12 q^2$ , multiplied by 2 to give the sum of the squares. It was also assumed that  $B_1 = 1$ . Many of these results are given in Reference 7, but the significance of the last column was unknown.

It can be noted that the amplitude of the fundamental decreases faster than the distortion factor with an increase in the number of levels, again indicating that more and more relative energy appears at the higher frequencies. This effect can be examined in more detail by considering one level:

where the fact that  $J_{-m} = (-1)^m J_m$  has been used. The Fourier transform of  $y(t)$  will then be given by

$$|F(j\omega)| = \sum_{r=-\infty}^{r=+\infty} \frac{1}{2} B_{2r+1} \delta(\omega - \omega_0(2r+1)) \quad (29)$$

$\delta =$  Dirac delta function

where for example

$$\begin{aligned} \frac{1}{2} B_1 &= \frac{A}{2} \left[ 1 + \frac{2q}{\pi A} \sum_{n=1}^{\infty} \frac{(-1)^n}{n} J_1 \left( \frac{2\pi n q}{A} \right) \right] \\ &= \frac{2q}{\pi} \sum_{i=1}^N \cos \left[ \sin^{-1} \left( i - \frac{1}{2} \right) \frac{q}{A} \right] \end{aligned}$$

The harmonics are shown in Figure 10.

TABLE I. QUANTIZATION HARMONICS

(A/q)	$B_1$ (%)	$B_3$ (%)	$B_5$ (%)	D.F. (%)	$\sim D.F. = \left( 100 \frac{q}{\sqrt{6A}} \right)$
1	110	0	-22	33	40.8
$(4\frac{1}{2})^+$	96				
5	100.97	-0.82	+ 0.47	7.55	8.16
$(5\frac{1}{2})^-$	97				
10	100.34			3.8	4.09
20	100.12			2	2.04
50	100.03			0.81	0.816

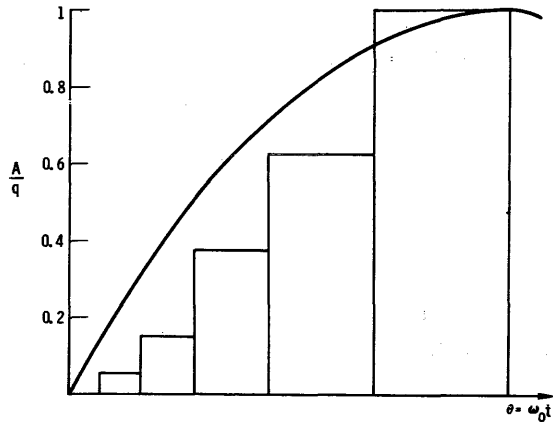


Figure 9. Quantized Sine Wave Power.

It has been observed, however, that approximately the same amplitude is obtained for the first several harmonics. As frequency is increased, the exact behavior of the amplitudes is not clear. Of course, they must decrease.

It should be noted here that in a practical system the sharp edges shown by the error characteristic are realistic. Since digital processes are being used, sharp levels and changes are descriptive. Some uncertainty or jitter at the quantum jump levels will merely contribute additional (high-frequency) noise. It is also evident that since a number system is involved, pure delta function sampling may be assumed, with no attenuation as would be the case with finite width sampling.

Referring to Equation 28, it can be seen that complete fold-overs will occur because the sampling theorem is not obeyed. There are harmonics at frequencies higher than the sampling frequency.

THE EFFECT OF SAMPLING ON THE HARMONIC POWER

In Figure 11, the sampler spectrum is added to the quantizer output spectrum. For simplicity, only the positive sampler frequencies are shown, and the amplitudes are not to scale.

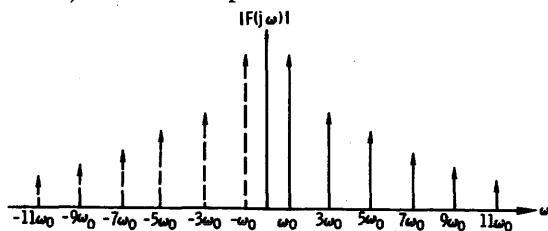


Figure 10. Frequency Domain.

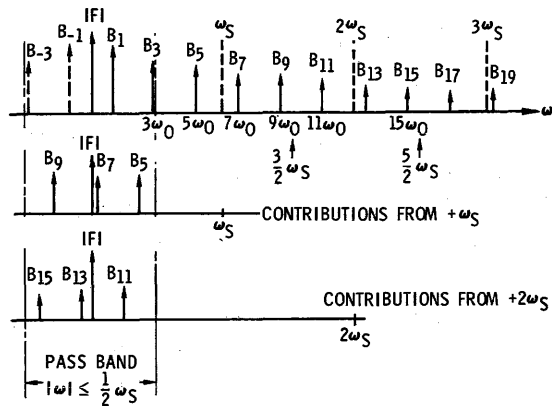


Figure 11. Spectrum of Quantized and Sampled Sine Wave.

Because energy is contained in both side bands, the reflections from  $-\omega_s$  will also contribute energy to the pass band. If only the contributions from  $\pm\omega_s, \pm 2\omega_s$  are summed with the original, the pass band  $0 < \omega < \frac{1}{2}\omega_s$  will have only the harmonics shown in Figure 12. Symmetrical side bands are present for  $-1/2\omega_s < \omega < 0$ . By induction, it can be seen that all of the original harmonics are reflected into the band  $-1/2\omega_s < \omega < 1/2\omega_s$ . Another approach would be to use the following argument:

All harmonics with frequencies  $|\omega| < 1/2\omega_s$  are kept. Harmonics in the range  $1/2\omega_s < |\omega| < 3/2\omega_s$  are reflected by the  $\pm\omega_s$  sampling frequencies.

Harmonics in the range  $(n - 1/2)\omega_s < |\omega| < (n + 1/2)\omega_s$  are reflected into the band by the  $\pm n\omega_s$  sampling frequencies.

Thus, as long as the ratio of the sampling frequency to the input sinusoid frequency is irrational, or essentially so, the error power in the pass band is identical to the error power out of the quantizer. The error power will be increased at some frequencies and decreased at others if the ratio is rational, since then two

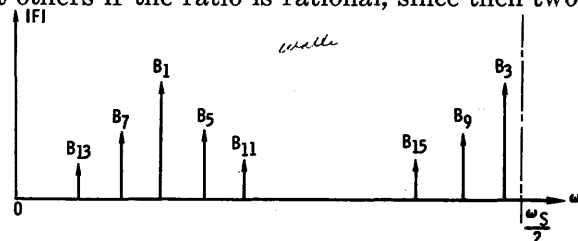


Figure 12. Pass Band Harmonics.

or more harmonics will have the same frequency in the pass band.

$$((B_i + B_j)^2 \neq B_i^2 + B_j^2)$$

If a filter is chosen which has a cut-off much lower than  $1/2 \omega_s$  (e.g.,  $\omega'$ ), the error power will be reduced because some of the reflected frequencies will lie between  $\omega'$  and  $1/2 \omega_s$ . In the previous example, if  $\omega' = 1/4 \omega_s$ , then  $A_3, A_9, A_{13}$  would not be transmitted. In general, the only contributing harmonics will be those lying in the range  $-\omega' + n\omega_s < \omega < n\omega_s + \omega'$ .

It is interesting to note that (error) noise is introduced into the system only by the non-linear operation of quantizing, and not by sampling. It can be concluded, therefore, that the estimate of the quantizing RMS error ( $q^2/12$ ) can be used as a measure of the upper bound to the system error so far. Furthermore, while all of the harmonics appear in the pass band, they appear at different frequencies so the quantizing error signal cannot be recovered exactly.

If an infinite time sampling rate is used, the exact quantized signal can be recovered. Of course, since the output is a staircase function, it does not have the same probability distribution in this case as does the input signal.

The following statement appears to be true for all signals: Since the probability density distribution cannot be negative, a band-limited Fourier transform for probability density can never be derived.

### THE EFFECT OF NOISE

Suppose that the input signal from the analog world is corrupted by noise. Let  $E_q$  be the quantizing error,  $E_n$  the noise, and  $f(t)$  the signal. The output of the quantizer is given by

$$y(t) = f(t) + E_n + E_q$$

with

$$E_q = \frac{q}{\pi} \sum_{n=1}^{\infty} \frac{(-1)^n}{n} \sin \left[ \frac{2\pi n}{q} \left( f(t) + E_n(t) \right) \right] \tag{30}$$

Although  $E_q$  is causally related to  $E_n$ , it is reasonable to assume that they are statistically

independent. Reference 4 indicates that this is a good assumption. What is the expected error?

$$\begin{aligned} \langle y^2 \rangle &= \langle (f + E_n + E_q)^2 \rangle \\ &= \langle f^2 \rangle + \langle E_n^2 \rangle + \langle E_q^2 \rangle \end{aligned}$$

assuming that  $f, E_n, E_q$  are uncorrelated. Then the total expected error squared is

$$\langle E^2 \rangle = \langle E_n^2 \rangle + \langle E_q^2 \rangle \tag{31}$$

If either infinite sampling or a perfect filter (over the range  $-1/2 \omega_s < \omega < 1/2 \omega_s$ ) is used, then

$$\langle E_q^2 \rangle = (1/12)q^2$$

If  $E_n$  is restricted to frequencies less than  $1/2 \omega_s$ , then

$$\langle E^2 \rangle = \langle E_n^2 \rangle \left( 1 + \frac{1}{12} \frac{q^2}{E_n^2} \right)$$

If the total noise is allowed to increase by as much as  $\sqrt{2}$ , then  $q$  can be as large as

$$q = \sqrt{12} \sqrt{\langle E_n^2 \rangle}$$

We then have

$\frac{q}{35 \text{ mv}}$	$\frac{(\langle E_n^2 \rangle)^{1/2}}{10 \text{ mv}}$
139 mv	40 mv

If filtering reduces the pass band to less than  $1/2 \omega_s$ , the quantizing can be even more coarse, since  $1/12 q^2$  is an upper bound. This assumes that the input noise has been filtered to the same pass band.

### THE EFFECT OF THE D-A CONVERSION

For simplicity it is assumed here that a zero-order hold is used to provide the digital to analog conversion. The treatment of higher order holds is similar. The process can be viewed in several different ways. In the frequency domain, the Fourier transform of the ZOH is given by a  $\frac{1}{X} \sin X$  expression, where  $X$  equals  $\pi \omega_0/\omega_s$ . Since this expression is not zero outside of what is termed the pass band ( $|\omega| < \frac{1}{2} \omega_s$ ), additional power will be converted; i.e., additional to the figures that have been used. As has been shown, however, an infinite number of sidebands and reflected harmonics must be considered. In order to simplify the task, consideration will again be given

to the specific example, a sine wave quantized to five levels, to which the sampling and the zero-order hold are now added. Figure 13 compares the signals received from the black box, with and without quantizing.

In keeping with our earlier use, error is defined as

$$\text{error} = \text{output} - \text{input}$$

This definition in this application includes the effect of the phase shift caused by the ZOH. This large effect, which tends to obscure the quantizing effect, can be examined by defining the relative error by

$$e_r = \text{relative error} = \frac{\text{output} - \text{input} \times \text{delay of } \frac{T}{2} \text{ sec}}{\text{output}}$$

The output signal, therefore, can be defined in terms of the fundamental plus an error. Specifically, using the nomenclature of Figure 1, we have

$$Z(t) = X\left(t - \frac{T}{2}\right) + e_r(t) \tag{32}$$

where T is the sampling period which is large enough to cover the conversion and computation time. Consider the single sine wave input. We have

$$Z(t) = A \sin \omega_0 (t - T/2) + e_r(t)$$

Another useful concept is to include the amplitude error of the fundamental separately:

$$Z(t) = A (1 + E_1) \sin \omega_0 (t - T/2) + e_r(t) \tag{33}$$

$$\langle e_r \rangle = 0$$

$$\langle e_r^2 \rangle = \frac{2\omega_0}{\pi} \left[ \int_0^{\pi/\omega_s} \sin^2 \omega_0 t dt + \int_{\pi/\omega_s}^{3\pi/\omega_s} \left( \sin \omega_0 t - \sin \frac{2\pi\omega_0}{\omega_s} \right)^2 dt + \dots \right] \tag{34}$$

In general

$$\langle e_r^2 \rangle = \frac{2\omega_0}{\pi} \left[ \int_0^{\pi/2\omega_0} \sin^2 \omega_0 t dt + \frac{2}{\omega_0} \sum \left[ \cos (2r + 1) \frac{\pi\omega_0}{\omega_s} - \cos (2r - 1) \frac{\pi\omega_0}{\omega_s} \right] \sin 2r \frac{\pi\omega_0}{\omega_s} + \frac{2\pi}{\omega_s} \sum \sin^2 \frac{2r\pi\omega_0}{\omega_s} \right] \tag{35}$$

FIVE-LEVEL QUANTIZATION

$$\omega_s = 18 \omega_0$$

ZERO ORDER HOLD

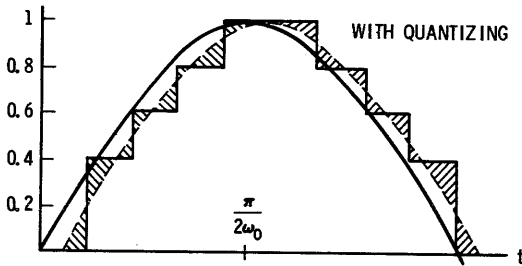
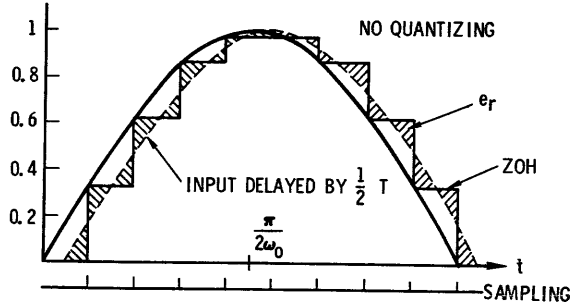


Figure 13. Combined Errors.

In this way, the process can be viewed as one which amplifies and phase shifts the input signal and then adds noise.

The error can be evaluated by examining Figure 13 which indicates the output signal from the zero-order hold with and without quantizing. When the input signal is shifted by 1/2 T seconds, symmetric errors are obtained whenever the sampling rate is an integral multiple of the signal frequency. Without quantization we have

When the quantizing is added to the sample and hold, similar equations are obtained. The combined mean squared error is given by

$$\langle e^2 \rangle = \frac{2\omega_0}{\pi} \left[ \int_0^{\pi/\omega_s} \sin^2 \omega_0 t dt + \sum_{r=1}^{\infty} \int_{(2r-1)\pi/\omega_s}^{(2r+1)\pi/\omega_s} \sin^2 (\omega_0 t - nq) \right] \quad (36)$$

with

$$\sin (2r - 1) \frac{\pi}{\omega_s} < nq < \sin (2r + 1) \frac{\pi}{\omega_s}$$

For the specific sine wave input quantized to five levels at a sampling rate 18 times the sine wave frequency, we find

$$\begin{aligned} E^2 &= 7.1\% A && \text{ZOH only} \\ E^2 &= 8.5\% A && \text{ZOH + quantizing.} \end{aligned}$$

If the estimate,  $E_q = q/\sqrt{12} = 5.77\% A$ , is used and if it is assumed that the errors are statistically independent, the following error is obtained:

$$\langle E^2 \rangle = \langle E_H^2 \rangle + \langle E_q^2 \rangle = 9.16\%$$

This estimate is within 10 percent of the value computed. It would then appear that the two error sources can be viewed as being statistically independent. An error map for various

values of the quantizing level and the sampling rate is given in Figure 14. It should be noted that as the ratio of sampling rate to signal frequency is decreased, more noise power appears.

The quantizing and the sampling and holding operation on the sinusoid  $A \sin \omega_0 t$  can thus be expressed as

$$B \sin (\omega_0 t + \theta) + E(t) \quad (37)$$

A more general expression may be derived for the errors resulting from the zero-order hold.

The parameter,  $\lambda$ , can be defined so that  $\frac{1}{\lambda}$  represents the number of samples per quarter cycle

$$\frac{1}{\lambda} = \frac{\omega_s}{4\omega_0} \text{ or } \lambda = \frac{4\omega_0}{\omega_s} \quad (38)$$

If  $\lambda$  is used and  $\omega_0 t$  is replaced by  $\theta$ , Equation 34 becomes

$$\langle e^2 \rangle = \frac{2}{\pi} \int_0^{\pi\lambda/4} \sin^2 \theta d\theta + \frac{2}{\pi} \int_{\pi/2 - \pi\lambda/4}^{\pi/2} (\sin \theta - 1)^2 d\theta + 2/\pi \sum_{r=1}^{1/\lambda - 1} \int_{\pi\lambda/4 (2r-1)}^{\pi\lambda/4 (2r+1)} \left( \sin \theta - \sin \frac{r\pi\lambda}{2} \right)^2 d\theta \quad (39)$$

$$\begin{aligned} \langle e^2 \rangle &= \frac{2}{\pi} \int_0^{\pi/2} \sin^2 \theta d\theta + \frac{2}{\pi} \left[ \frac{\pi\lambda}{4} + \sum_{r=1}^{1/\lambda - 1} \left( \sin^2 \frac{r\pi\lambda}{2} \right) \frac{\pi\lambda}{2} \right] + \frac{4}{\pi} \int_{\pi/2 - \pi\lambda/4}^{\pi/2} - \sin \theta d\theta + \frac{4}{\pi} \sum_{r=1}^{1/\lambda - 1} \\ &\int_{\pi\lambda/4 (2r-1)}^{\pi\lambda/4 (2r+1)} \left( - \sin \frac{r\pi\lambda}{2} \right) \sin \theta d\theta \end{aligned} \quad (40)$$

Now, one of the expressions resulting from the integration and trigonometric identities is zero:

$$\sum_{r=1}^{1/\lambda - 1} \cos r \pi \lambda = 0 \quad (41)$$

This simply depends upon the fact that

$$\cos (\pi - \theta) = -\cos \theta$$

When evaluated, Equation 40 reduces to

$$\langle e^2 \rangle = 1 - \left( \frac{1}{\pi\lambda} \right) \sin \frac{\pi\lambda}{4} \equiv 1 - \frac{\sin \frac{\pi\omega_0}{\omega_s}}{\frac{\pi\omega_0}{\omega_s}} \quad (42)$$

Using the first term of the expansion gives for small  $\lambda$  (large enough number of levels/quarter cycle)

$$\langle e^2 \rangle \approx \frac{1}{3!} \left( \frac{\pi\lambda}{4} \right)^2 = \frac{\pi^2}{96} \lambda^2 \quad (43)$$



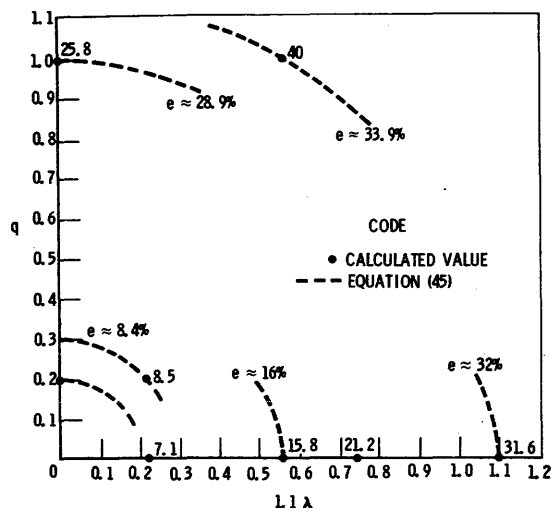


Figure 14. The Combined Error.

Thus, we can write

$$\langle e^2 \rangle \approx \frac{1}{12} \left( \frac{\sqrt{2} \pi \lambda}{4} \right)^2 \quad (44)$$

and treat  $\left( \frac{\sqrt{2} \pi}{4} \right)$  as a form factor, to express the combined error by

$$\langle e_r^2 \rangle = \frac{1}{12} \left( q^2 + (1.11 \lambda)^2 \right) \quad (45)$$

The estimates of the combined error are thus ellipses in the  $\lambda, q$  plane, or circles in a  $q, 1.11 \lambda$  plane. These estimates are plotted in Figure 14, together with calculated combined errors.

CONCLUSIONS

The results observed lead to the following conclusions:

1. The quantizer adds noise power energy to the system, and this energy is spread through a wide range. At even low quantization levels, it was observed that the energy of perhaps the first 50 harmonics was appreciable. As the input signal amplitude is increased, the energy associated with the lower harmonics is reduced, and this energy appears at the higher and higher frequencies.
2. Utilizing sine wave input signals is an effective way of obtaining error estimates for hybrid computation. When four or five bits are used in the quantization,

numerical results can be obtained fairly readily.

3. Much of the dynamic RMS error is the result of the linear phase lag resulting from the data reconstruction. If this phase-shift error is eliminated, the quantization and the sample-hold errors are statistically independent. A sample and hold error parameter,  $\lambda$ , equivalent to the quantizing step  $q$ , can then be defined so that the combined error is an elliptical function of  $q$  and  $\lambda$ .
4. If the noise resulting from the quantizing and data reconstruction is unacceptable, several alternatives are available:
  - a. A band pass filter can be used to eliminate some of the energy at higher frequencies within the pass band.
  - b. The quantizing levels can be increased. Because the RMS noise is directly proportional to the quantizing level, a linear effect will be observed.
  - c. The sampling error parameter,  $\lambda$ , can be decreased by increasing the sampling rate. The error is directly proportional to  $\lambda$  (i.e., inversely proportional to sampling rate).

RECOMMENDATIONS

It is recommended that this line of approach be used to develop general criteria for the performance and specification of hybrid computation systems. In particular, this approach should be used to develop error estimates for first- and second-order hold systems. Test and analysis using the sine waves are not only fairly simple, but most important, are readily comprehensible to the using engineer.

REFERENCES

1. SHEPPARD, W. F., "On the Calculation of the Most Probable Values of Frequency Constants, for Data Arranged According to Equidistant Divisions of a Scale," Proceedings of the London Mathematical Society, Vol. 29 (1898), pp. 353-357.
2. BENNETT, W. R., "Spectra of Quantized Signals," Bell System Technical Journal, Vol. 27 (July, 1948), pp. 446-472.

3. INMAN, S., "The Probability of a Given Error Being Exceeded in Approximate Computations," *Mathematical Gazette*, Vol. 34 (May 1950), pp. 99-101.
4. WIDROW, B., "A Study of Rough Amplitude Quantization by Means of Nyquist Sampling Theory," *IRE, PGCT*, Vol. CT-3, No. 4 (Dec. 1956), pp. 266-276.
5. WIDROW, B., "Statistical Analysis of Amplitude-Quantized Sampled-Data Systems," ONR Technical Report No. 2103-1 (May 10, 1960).
6. KATZENELSON, J., "On Errors Introduced by Combined Sampling and Quantization," *IRE, PGAC*, Vol. AC-7 (April 1962), pp. 58-68.
7. CLAVIER, A. G., PANTER, P. F., and GRIEG, D. D., "PCM Distortion Analysis," *Electrical Engineering*, Vol. 66 (Nov. 1947), pp. 1110-1122.
8. COURANT, R., and HILBERT, D., *Methods of Mathematical Physics*, Vol. 1, Interscience Pub. Inc., N.Y., English Ed. (1953).
9. DAVENPORT, W. B., and ROOT, W. L., *An Introduction to the Theory of Random Signals and Noise* (McGraw-Hill Book Co., Inc., N.Y., 1958).
10. TOU, J. T., *Digital and Sampled-Data Control Systems* (McGraw-Hill Book Co., Inc., N.Y., 1959).
11. TOU, J. T., *Optimum Design of Digital Control Systems* (Academic Press Inc., N.Y., 1963).
12. MONORE, A. J., *Digital Processes for Sampled Data Systems* (John Wiley and Sons, Inc., N.Y., 1962).
13. MIURA, T., and IWATA, J., "Effects of Digital Execution Time in a Hybrid Computer," *AFIPS FJCC Proceedings*, Vol. 24 (1963), pp. 251-266.
14. GELMAN, R., "Corrected Inputs—A Method for Improving Hybrid Simulation," *AFIPS FJCC Proceedings*, Vol. 24 (1963), pp. 267-276.

# REAL TIME RECOGNITION OF HAND-DRAWN CHARACTERS

*Waren Teitelman \**  
*Massachusetts Institute of Technology*  
*Cambridge, Massachusetts*

## 1.0 SUMMARY

This paper describes a system designed to recognize handdrawn characters in real time. The central feature of the system is its use of the time sequence information of the input character.

In its abstract form, the system may be viewed as a collection of discrimination nets, or filters. Each net operates on the input character, or rather on a sequence of property vectors representing the encoding of the input character in the time domain, and produces a set of so-called candidate characters. The system then utilizes reliability estimates for the individual nets to select the final output character.

A program representing a particular implementation of the system has been written for the DEC PDP-1. The user draws the character on the face of a cathode ray tube with a light pen. The program follows the pen and constructs the appropriate sequence of property vectors. The properties used are simple geometrical ones, and the discrimination nets are tree structures which store sequences of property values. Recognition time is of the order of .25 seconds, of which approximately .15 seconds are occupied by drum read-and-write operations required by the small memory size (4K) of the machine.

The user can teach the program to recognize his set of characters. The learning process for the program involves modifying individual decision trees, changing the weights on each tree, and where necessary, introducing new decision trees with their corresponding properties into the system. Because the program was written as an input device to a larger man-machine system, the description of the implementation stresses the human engineering features. A qualitative evaluation of the system as implemented is offered, together with possibilities for expanding and generalizing the program.

## 2.0 INTRODUCTION

Research in pattern recognition may be characterized as a search for invariants.\*\* The problem is to find attributes that all instances of a given pattern have in common that instances of other patterns do not. The particular class of invariants selected will ultimately determine the performance of the pattern recognition system. However, performance is not the sole factor influencing the choice of invariants. Associated with any research project are certain objectives and goals. In

---

\*\* A number of excellent surveys and bibliographies of the voluminous literature on pattern recognition are available.<sup>1, 7, 9</sup> Minsky<sup>8</sup> gives a general survey of the entire area of artificial intelligence, including pattern recognition.

---

\* Consultant, System Development Corporation, Santa Monica, California.

pattern recognition, achievement of these goals is not only a function of *what* the system recognizes, but also *how* it performs the recognition. Thus the selection of invariants may be more influenced by the purposes and philosophy of the research than by a desire for a high rate of recognition. To illustrate this point, let us examine briefly two character recognition schemes which are at opposite ends of a spectrum with respect to the aims of their research. Both perform very well in that their recognition accuracy is high. Their basic difference in purpose and philosophy is reflected in two very diverse sets of invariants. These two schemes will provide a frame of reference for the research described in this paper.

### 2.1 Cyclops-1

Cyclops-1<sup>5</sup> is a sophisticated character recognition program which involves hypothesis generation and testing in the identification process. The system generates a hypothesis concerning the nature of the input; the hypothesis is tested, and if it is correct, it becomes the output. In the present version, the program merely generates hypotheses in a predetermined, fixed order, with the major part of the research going into hypothesis testing.

The hypothesis testing scheme involves a series of questions about characteristics of particular segments in the pattern, or of the relationships between segments, or of the pattern as a whole. There are 42 characteristics, and they are presumably selected because they were the type of things noted by humans. For example, five characteristics used by the system are (1) the predominance coefficient of a segment, (2) the straightness coefficient of a segment, (3) the orientation of a segment, (4) the number and location of inflection points of a segment, and (5) the number and location of the intersections of a segment with other segments. The items to be recognized by the program, i.e., alphabetic characters, are defined in terms of these characteristics.

The system has great generality. New characters may be defined with little difficulty. However, a great deal of computation is necessary to determine the segmentation of a character, and to identify intersection points, inflection points, etc. In operation, the system

receives the input via a pen tracking routine which accepts data written on a cathode ray tube. The system then transforms a table of successive light pen locations into information concerning line segments that comprise the character. The program is thus quite large, and recognition time is slow.\*

### 2.2 The Stylator

A scheme described by Dimond of Bell Labs.,<sup>2</sup> on the other hand, is much simpler. A practical electronic device called the Stylator has been designed and constructed which performs the recognition. The user is constrained to draw the character (a numeral) around two dots which define a set of bipolar coordinates (Fig. 1). The seven lines identified in the figure are actually narrow conductors connected to a source of potential. When the numeral is written with a stylus connected to a source of potential, the stylus energizes the conductors, which in turn cause certain flip-flops to operate and drive the rest of the circuit to indicate the correct numeral. This can be done because the numerals are defined uniquely in terms of their set of line crossings. For example, the numeral '3' is defined as crossing lines A, B, C, D, and not crossing line F. (It may or may not cross lines E and G.) Similarly, the numeral '7' must cross line A and line B, and may not cross line D or F. Admittedly some variations on the construction of a numeral might be conceived which would confuse the machine, and in some cases the user must be trained briefly to familiarize himself with the system. But this causes no great difficulty, and the knack of positioning the character and drawing it to the correct scale is learned readily.

### 2.3 Comparison of Research Approaches

The Cyclops-1 approach to character recognition reflects an attempt at constructing a general system. The search for invariants has therefore led the researchers to examine the way in which humans recognize characters. The resulting properties involve very abstract concepts and considerable computation is necessary in determining them. However, the properties used are those that seem to define

\* Depending on the number of line segments between three and twelve seconds per character.

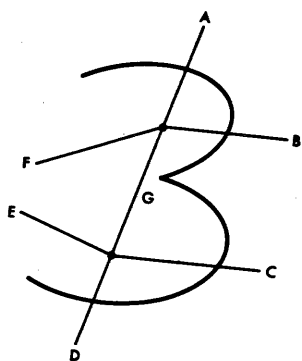


Figure 1. Bipolar Coordinates for Character Recognition.

the character in the everyday sense. Cyclops-1 recognizes an 'A' because it *is* an 'A.' The power in this approach is reflected in the ability Cyclops has demonstrated to perform scene analysis: the system can analyze complex visual inputs consisting of an arbitrary number of characters present simultaneously, where the characters may be of different size and orientations, may overlap or be inside of one another, or may be superimposed on arbitrary backgrounds consisting of meaningless lines or spots or geometric shapes. This is something that is beyond the range of most character recognition schemes, although humans do it fairly well.

In the Stylator scheme, the emphasis is on fast and efficient recognition. The set of invariants for the numerals '0' through '9' is specified as a collection of line crossings only because this is simple and fortunately unique. Note however that the user could merely touch his stylus to the correct lines, without even drawing the character, and have the Stylator recognize it. On the other hand, one could easily construct numerals that most humans would recognize correctly but that would confuse the Stylator. Essentially all the Stylator does is map a set of 128 ( $2^7$ ) possible inputs onto a set of ten possible outputs. To the extent that this correlates with correct character recognition, the device performs well. Its assets are its speed and efficiency.

While the author feels that the type of research embodied in the Cyclops project will be more productive in the long run, the aims of the research reported here are closer to

those of the Stylator project. The research began with an attempt at developing a fast and practical character recognition program to use as input to a larger system. The central theme was to try to use the information of the time sequence of events in constructing the character to establish a simple class of invariants. Since a more flexible scheme than that of Diamond was desired, a simple and rigid definition of alphabetic characters such as the sort utilized by the Stylator would not suffice. As in the work described by Kuhl,\* the properties of the program reflected the "true nature" of the character more closely than that of the Stylator, but were not as general as those used by Cyclops.

As the research proceeded, certain generalizations suggested themselves, and where these did not conflict drastically with the rate of recognition, nor make implementation difficult, they were included in the program. The system presented here arose as the logical abstraction and generalization of the program, rather than the other way around.

### 3.0 THE SYSTEM

#### 3.1 Definitions and Notations

Since the system will treat characters as events in time, a distinction is made between a "form" and a "figure." A figure will be taken to mean a drawing on a piece of paper or on a cathode ray tube, in other words, a picture of a symbol. A form will be considered the sequence of events that produce the drawing, in other words, the figure with its time sequence. A character refers to the symbol that either the figure or the form denotes. Thus, this scheme operates on forms to produce characters, as opposed to more conventional schemes that operate on figures to produce characters.

\* Kuhl's interest is in finding a *convenient* way of separating classes of letters.<sup>4</sup> He makes use of invariants such as the number of free ends of the character, the presence or absence of sharp angles, etc. Since an 'A' can be drawn without any sharp angles, and one can construct a legitimate 'B' with two free ends, the "definition" of a character according to his scheme may not coincide with that used by a human, although his invariants are more suggestive of the characters than the invariants used by Diamond for the Stylator.

The sequence of events that comprise a form is denoted by  $F(t)$ . This notation is meant to suggest the sequence of events in the time interval  $(0,t)$ .\* A property vector is associated with each form  $F(t)$ . This vector is denoted by  $\pi(t)$ , and also changes with time. It may be thought of as describing what is happening at the present instant. Presumably it contains information of use to the recognition function, although it could also indicate some other function—e.g., the phase of the moon or the body temperature of the user. In the implementation of the system,  $\pi(t)$  reflects the position of the pen at time  $t$ .

The system operates on the sequence of property vectors that represent the sequence of events in the construction of the form. Each component of the property vector corresponds to a separate property, with the  $j$ th component of  $\pi(t)$ — $\pi_j(t)$ —corresponding to the  $j$ th property,  $P_j$ . These properties apply to characteristics of the form.  $\pi_j(t)$  is a three-valued function which informs the system whether or not the form has property  $P_j$ .  $\pi_j(t)=1$  is interpreted to mean that the form has property  $P_j$  at time  $t$ .  $\pi_j(t)=0$  means that the form does not have property  $P_j$ . The third value,  $\pi_j(t)=\omega$  indicates that the property is not applicable to the form at time  $t$ . This last value serves to break up the sequence of property values into sub-sequences. In the program, the individual properties indicate whether or not the pen is in a given region, and these sub-sequences of values correspond to strokes of the pen.

### 3.2 Input to the System

As indicated, the system operates on a sequence of property vectors representing the encoding of the input character in the time domain. The particular encoding will depend on exactly what properties are being used by the system, but all encodings are constructed in the same fashion. They are merely the sequence of all property vectors attained by the form from time  $t = 0$  until the form is completed. In other words, the first vector in the sequence informs the system what properties are present, not present, or inapplicable at

\* For convenience, assume all forms begin at time  $t = 0$ .

time  $t = 0$ ; thereafter, whenever the property vector of the form changes, the new vector is added to the encoding sequence. The sequence is thus normalized in time in that there is no information on when the form had a certain property vector or how long it maintained the same vector. The length of the sequence is thus a function of how often the properties vary in the course of constructing the form, and not how quickly or how slowly the form was constructed.\*\*

### 3.3 The Recognition Process

Each property has a discrimination net associated with it. The discrimination net filters out all but those characters that have similar forms. Similarity is defined for a particular discrimination net as having the *same sequence* of property values. Thus, for each property  $P_j$ , the corresponding sequence of property *values* of  $\pi_j$  is extracted from the complete sequence of property *vectors*. This sequence is usually considerably shorter than the entire sequence of property vectors, because the encoding contains all changes of all properties, not only the changes in  $\pi_j$ . The discrimination net then selects as its set of candidate characters all those characters whose sequence of  $\pi_j$  values are identical to the sequence of  $\pi_j$  values for the input form.

The particular properties used affect the performance of the system tremendously. If many characters have the same sequence of property values, then the decision problem will be almost as difficult after the discrimination nets have operated as before. On the other hand, if the sequence of property values is long and complicated, then the chance that it will be duplicated exactly is small, and either the system will have to be provided with an extremely large dictionary of property value sequences or else the discrimination nets will not pass any characters into their candidate set. In one case, the defining criteria are too lax, allowing many characters to have the same definition, and in the other they are too stringent, re-

\*\* There could conceivably be properties that reflected the length of time required to construct the form. For example,  $\pi_k = 1$  if form is completed in less than time  $t_0$ , etc. But there is nothing in the encoding sequence that explicitly represents the passage of time other than the changing property vectors.

quiring the user to construct his character with great precision and consistency.

The final decision procedure that operates on the sets of candidate characters reflects an attempt to recover from either of the above two possibilities. The concept employed here is parallel processing. It has been observed in many pattern recognition projects that a reliable decision may be possible even where individual tests are poor, provided each test contributes some different fractional bit of information. In our case, this is done by having the presence of a character in a candidate set *suggest* that the character might be the correct one and, correspondingly, having the absence of a character in the candidate set *prejudice* the system against it. This is done by weighting the characters with a reliability\* estimate for each net, and selecting the character with the highest score. In the case where all nets are equally reliable, this amounts to selecting the *character* that appears in the plurality of candidate sets. In the implemented version of the system, a small, basic set of properties is used, all of which have reliability estimates of the same order of magnitude. Additional properties, with lower estimates, are considered in the decision only when the basic weights are nearly equal.

To summarize, the system receives a sequence of vectors which represent the changing characteristics of the input form. The components of each vector are three-valued functions which indicate whether or not the form had a particular property at the time the vector was computed, although information on when this occurred is not included with the input sequence.

For each property, the sequence of property values attained by the component of the vector corresponding to that property is extracted from the entire sequence of property vectors. A discrimination net retrieves all characters that have forms with the *same* sequence of property values. This set of characters is called the candidate set. The system then selects a character from among all of the various candidate sets by weighting each char-

acter in proportion to the reliability of the discrimination nets that suggested it, i.e., proportional to the reliability of the particular property.

## 4.0 THE PROGRAM

### 4.1 The Property Vector

If one is asked to visualize a character as a sequence of events in real time, the picture that comes to mind is that of a pen tracing a course over a piece of paper. Similarly the properties first experimented with here were simple geometrical properties that merely reflected the position of the pen at time  $t$ . These were found to be suitable, and more sophisticated properties involving pen velocity, angle of line segment, curvature, etc., were not needed.

Although characters may be drawn to different scales and orientation, let us assume that we have a character normalized and have superimposed it on a rectangle. We will consider this rectangle to be divided into various, possibly overlapping regions. Each region corresponds to a property. A form is said to have the property  $P_j$ , and  $\pi_j = 1$ , if the position of the pen at time  $t$  lies in region  $R_j$ . If the pen is not in region  $R_j$ , then the figure does not have this property, and  $\pi_j = 0$ . When the pen is not touching the page,  $\pi_j = \omega$ , in this case for all  $j$ . Thus, the property vector roughly determines the position of the pen at time  $t$ , and the sequence of property vectors will trace out the page of the pen as it completes a form. Fig. 2 depicts a four-property scheme, and Table I gives the sequences of property vectors for the two different 'E's shown in Fig. 3.

These four properties were first considered because of a desire to divide the character into a left third, center third, and right third, and similarly, upper, lower, and middle third. They are obviously redundant as there are 16 possible property vectors corresponding to only nine subregions of the rectangle. However, they were powerful enough to enable the author to construct a unique definition of all English uppercase letters as the author drew them, with the exception of 'U' and 'V.' While the system allows the user to define his own properties—which for the purposes of the pro-

\* Reliable in the nontechnical sense, meaning valid or trustworthy.

Table I. Sequence of Property Vectors for Two Different 'E's (See Figure 3).

$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$	$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$
1	1	1	1	0	1	1	1
1	1	0	1	0	1	0	1
1	1	0	0	1	1	0	1
0	1	0	0	1	1	0	0
0	0	0	0	0	1	0	0
$\omega$	$\omega$	$\omega$	$\omega$	0	0	0	0
1	1	0	1	$\omega$	$\omega$	$\omega$	$\omega$
0	1	0	1	1	1	0	1
0	0	0	1	0	1	0	1
$\omega$	$\omega$	$\omega$	$\omega$	0	0	0	1
1	1	1	1	$\omega$	$\omega$	$\omega$	$\omega$
0	1	1	1	0	1	1	1
0	0	1	1	0	0	1	1
$\omega$	$\omega$	$\omega$	$\omega$	$\omega$	$\omega$	$\omega$	$\omega$
Type 1				Type 2			

gram means defining his own regions—it was decided that the program, and the user, should start with a useful, basic set of properties. These four properties were selected as a nucleus for the set of properties used by the program, and the sequences of property vectors of Table I would be the actual input to the recognition portion of the program.

#### 4.2 The Discrimination Net

Since the operation of the individual discrimination net is essentially one of retrieval, one possible representation for the net would be a simple table. The program would then merely look up the sequence of property

values in the table and retrieve any and all characters associated with that sequence. However, from a standpoint of conservation of both computer time and storage, this would be very inefficient. In addition, since we would like to have the system learn by adding new forms to its repertoire, we would have to cope with the problem of variable-length tables as well as variable-length entries. (We do not know, a priori, how long a given sequence of property values will be, nor how many characters will have this sequence in their forms.) Some type of list structure is called for.

The structure adopted is the binary tree. This is a special type of list structure consist-



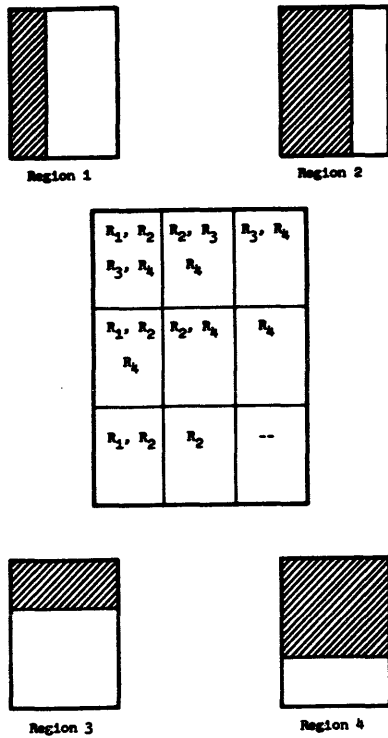


Figure 2. A Four Property Scheme.

ing of a collection of nodes and branches. Each node is connected to one and only one node above it by a single branch. Since the program operates on changes in the property value, and there are three possible values for any property, only two branches are needed at a node to represent a sequence of values; hence the tree is a binary tree. Fig. 4 illustrates a binary tree which has stored the sequences of property values for  $P_1$  (Fig. 2). The algorithm used to construct the tree is that the value 1 is always to the left of the value 0, and the value 0 is always to the left of the value  $\omega$ . The set of candidate characters at a node is actually a list appended to that node. In Fig. 4, the set of candidate characters has been placed adjacent to its corresponding node.

If we return to the two types of 'E's (Fig. 3 and Table I), we can follow the operation of the discrimination net. First, the appropriate sequence of property values must be extracted from the sequence of property vectors. This process is illustrated in Table II. For the first type of 'E', the program then descends the tree and arrives at the node whose candidate set consists of the two characters 'E' and 'A.' This would therefore be the output of the net.

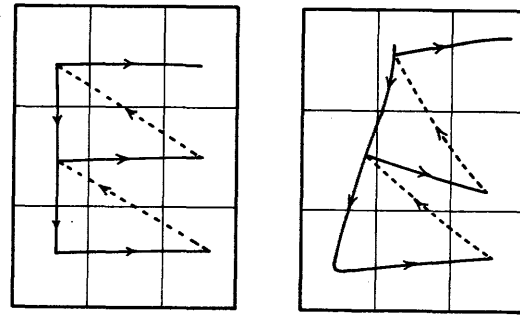


Figure 3. Two Different 'E's (See Table I).

For the second 'E,' the program arrives at the node marked with an asterisk and concludes that no character has a form with this sequence of property values for  $P_1$ ; the candidate set will therefore be empty.

### 4.3 Training the Program

The process of training the program involves modifying and augmenting not only the individual discrimination nets, but also the reliability estimates associated with these nets, and occasionally even creating new nets. The program begins with the four basic discrimina-

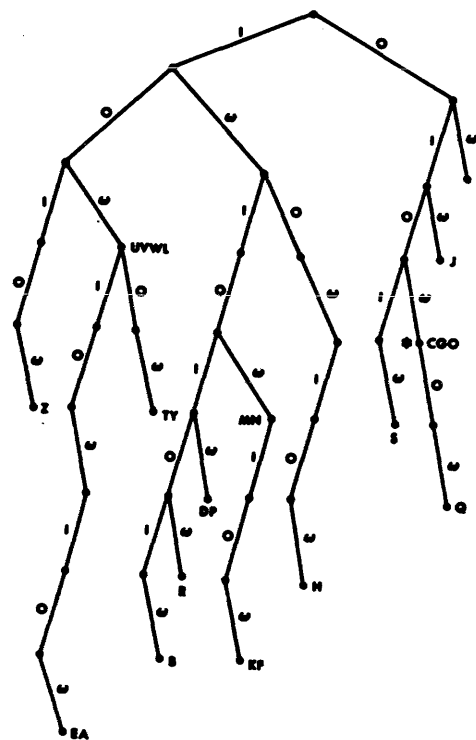


Figure 4. A Binary Tree for Property  $P_1$ .

Table II. Sequence of Property Values for Two Different 'E's.  
The sequence of property values for each individual property was extracted from the sequence of property vectors in Table I. Note that the sequences for  $\pi_2$ ,  $\pi_3$ , and  $\pi_4$  are identical.

$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$	$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$
1	1	1	1	0	1	1	1
0	0	0	0	1	0	0	0
$\omega$	$\omega$	$\omega$	$\omega$	0	$\omega$	$\omega$	$\omega$
1	1	0	1	$\omega$	1	0	1
0	0	$\omega$	$\omega$	1	0	$\omega$	$\omega$
$\omega$	$\omega$	1	1	0	$\omega$	1	1
1	1	$\omega$	$\omega$	$\omega$	1	$\omega$	$\omega$
0	0			0	0		
$\omega$	$\omega$			$\omega$	$\omega$		
Type 1				Type 2			

tion nets corresponding to the four properties discussed earlier. No sequences of property values are stored in any of the nets, and each net is given an initial success number of one. It is this success number which will reflect the reliability of the net.

#### 4.3.1 Modifying the Tree

The user draws a form and the sequence of property vectors is delivered to the program. Since there are no forms stored in the nets, all candidate sets are empty and the program is unable to guess a character. The user then informs the program what the character was, and the correct sequence of property values is stored on each tree along with the name of the character.

As the process continues, the trees grow more complex, but the procedure of growing new branches and/or adding new characters to candidate sets remains the same. Thus, in Fig. 4, when presented with the type-2 'E,' the program would add on the additional branches (1, 0,  $\omega$ , 0,  $\omega$ ) at the node marked with an asterisk and place a candidate set consisting of the single character 'E' at the last node.

After the program has been in operation for some time, correct decisions may be made even though some candidate sets do not include the right character. Similarly, a wrong guess may be made even though some candidate sets do include the correct character. In both cases, one or more trees do not contain the sequence of property values for the input form. If the program is still in training, these sequences are added to the corresponding trees. This means that the next time the program is presented with the same form, not only will the correct decision be made, but the correct character will be in each candidate set, i.e., the "vote" will be unanimous.

#### 4.3.2 Success Numbers—A Reliability Estimate

The whole point of the training is not to provide the program with all possible form it may encounter, but to give it enough experience so that, even on unfamiliar forms, some of the trees may be able to make guesses, though others may generate empty candidate sets. The weighting attached to the candidate sets of each tree reflects the reliability of that tree as

demonstrated during the training period. This is done by incrementing (by one) the success number of a tree every time its candidate set contains the correct character. Similarly, the success number is decreased (by one) each time the candidate set does not contain the correct character. Thus trees that suggest correct characters often are given more weight than those that do not. For the four basic properties, the success numbers usually remain about the same. However if the user tends to be more consistent with respect to the vertical dimensions of his character and varies on his horizontal dimensions, this would be reflected in the larger success numbers for property  $P_3$  and  $P_4$  (Fig. 2).

### 4.3.3 Generating New Properties

Although the four basic properties enable the program to achieve quite a high level of discrimination, they are not foolproof. Suppose in the course of its training, the program were taught the character 'Z' (shown in Fig. 5a and Table III). Later, when confronted with the form in Fig. 5b the program correctly identifies it as a 'Z.' Moreover, it is *sure* that it is a 'Z.' In other words, every tree suggested a 'Z.' The only difficulty is that a human would recognize this form as denoting the character '2.' Telling the program that this form denotes a '2' does not solve the problem. It would merely result in the program identifying subsequent 'Z's as '2's. The two characters are *identical* within the limits of discrimination of the program.

At this juncture there are only two possibilities. Either the user must relent and draw one or the other form differently, e.g., cross

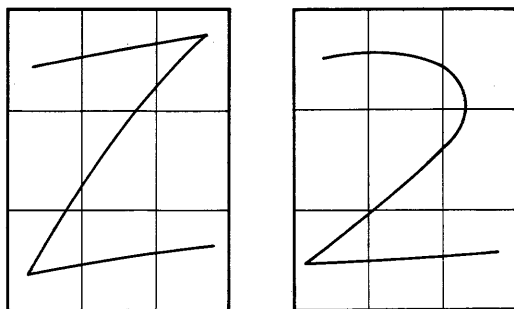


Figure 5. Two Forms Identical Within the Limits of Discrimination of the Program (See Table III).

the 'Z,' or he must extend the discrimination of the program. The latter course is preferable and, for the case of this program, easily achieved.

Observing that the difference between the two forms lies in the fact that a 'Z' enters the upper right hand corner, and a '2' does not, we communicate this fact to the program by activating a new region  $R_5$ , and a corresponding property  $P_5$ . When the program has added this new discrimination net to its collection of existing nets, it has the necessary discrimination to distinguish between these forms (Fig. 6 and Table IV).

There are implications in this procedure beyond the ability to distinguish between two similar characters. First, the new tree is started out with a success number of one. The other trees, by virtue of their training, have built up larger success numbers. Consequently, the new tree will have an effect only in those decision where there is a tie vote from the other trees. In other words, it will only be called upon to discriminate *when it is needed*. For those sequences that the four original trees have been trained to handle, the new tree will not be used.

The second implication is that the new tree can eventually outweigh the old trees if it proves to be more reliable. As the new tree participates in more correct decisions its success number grows. If the user has generated a very effective property, then this new tree can assume a position of dominance in the decision procedure. The user is protected, however, if he has generated a poor property. This will be reflected in its low success number, and it will only be used when there is no other way

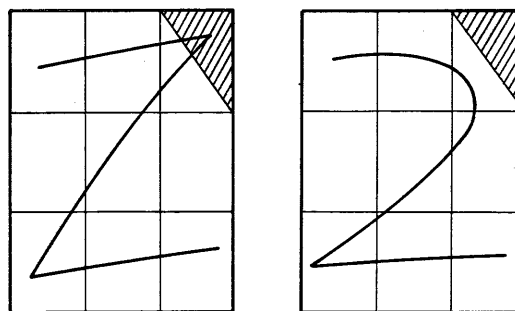


Figure 6. Generating a New Tree (See Table IV).

Table III. Sequences of Property Vectors and Values for Figure 5.

$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$	$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$	$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$	$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0
0	0	1	1	1	1	$\omega$	$\omega$	0	0	1	1	1	1	$\omega$	$\omega$
0	1	1	1	0	0			0	0	0	1	0	0		
0	1	0	1	$\omega$	$\omega$			0	1	0	1	$\omega$	$\omega$		
1	1	0	1					0	1	0	0				
1	1	0	0					1	1	0	0				
0	1	0	0					0	1	0	0				
0	0	0	0					0	0	0	0				
$\omega$	$\omega$	$\omega$	$\omega$					$\omega$	$\omega$	$\omega$	$\omega$				
$\pi$ vectors				$\pi_j$ values				$\pi$ vectors				$\pi_j$ values			

to make a decision. Thus, the generation of new trees proves to be a very powerful heuristic. It enables the clever user to create an extremely powerful character recognizer, and does not penalize the novice, since his new properties can be evaluated in light of the program's experience with him.

#### 4.4 The Program in Operation

The program operates in the following way. The user draws a form within a rectangle of light displayed on the scope; an input routine converts this form to a sequence of property vectors. The particular properties used, as mentioned above, indicate the position of the pen. Hysteresis regions in those areas where properties change have the effect of eliminating the sharp line of demarcation between two regions.

When the user has completed a form and signals to the program, a recognition routine is brought into memory. This routine operates on the property sequence computed by the in-

put routine. It retrieves the candidate sets from the trees, and makes a guess as to the identity of the character that was drawn, using the success number technique discussed above. When in training mode, it uses the feedback information to grow new branches and add characters where necessary, and to modify the success number of existing trees.

##### 4.4.1 The Input Routine

When the program is in operation, a rectangle of light can be seen flickering on the cathode ray tube. This is a 48 x 64 raster of points within which the user must draw his forms.\* Associated with each point in the raster is a property vector. As the user draws on the scope, the pen is tracked by the program. The

\* The user could be allowed to draw his form anywhere on the scope face. However, this would entail saving the entire form and normalizing it after it had been completed, which would be costly in space and time. By limiting the drawing to the rectangle, the property vectors can be computed while the form is being drawn.

property vector for the point in the center of the pen's field is retrieved from the raster table by the input routine. If the property vector differs from the last vector in the sequence, it is appended to the sequence. When the user lifts the pen from the scope, the property vector  $(\omega, \omega, \omega, \omega)$  is added to the sequence. This process continues until the user signals he has completed a form. The  $\pi$  vector sequence is then passed to the recognition routine.

Because of the nature of the properties used by the program, it might be possible to *compute* the property vector directly rather than store a three-thousand word table. There are two reasons why this is not done. First, table lookup is much faster than computing the property vector. This speed is important while tracking the pen. A long computation might

allow the pen to travel far enough that the program would "lose it." This, in turn, would mean that the program would have to display the entire raster to find the pen. This can be annoying to the user. In addition, the program would interpret this as meaning that the pen had left the face of the scope, and add  $(\omega, \omega, \omega, \omega)$  to the property vector sequence.

The second reason for looking up the property vector rather than computing it lies in the generality of the lookup procedure. Since each point has its own property vector, the regions are not restricted to a portion of the rectangle lying to one side of a straight line, as in the basic set of properties. The regions need not even be connected. Although the regions used for the basic set *are* simple, there is no reason why more general regions should not prove

Table IV. Sequences of Property Vectors and Values after Adding New Property (See Figure 6).

$\pi_1$ $\pi_2$ $\pi_3$ $\pi_4$ $\pi_5$	$\pi_1$ $\pi_2$ $\pi_3$ $\pi_4$ $\pi_5$	$\pi_1$ $\pi_2$ $\pi_3$ $\pi_4$ $\pi_5$	$\pi_1$ $\pi_2$ $\pi_3$ $\pi_4$ $\pi_5$
1 1 1 1 0	1 1 1 1 0	1 1 1 1 0	1 1 1 1 0
0 1 1 1 0	0 0 0 0 1	0 1 1 1 0	0 0 0 0 $\omega$
0 0 1 1 0	1 1 $\omega$ $\omega$ 0	0 0 1 1 0	1 1 $\omega$ $\omega$
0 0 1 1 1	0 0 $\omega$	0 0 0 1 0	0 0
0 0 1 1 0	$\omega$ $\omega$	0 1 0 1 0	$\omega$ $\omega$
0 1 1 1 0		0 1 0 0 0	
0 1 0 1 0		1 1 0 0 0	
1 1 0 1 0		0 1 0 0 0	
1 1 0 0 0		0 0 0 0 0	
0 1 0 0 0		$\omega$ $\omega$ $\omega$ $\omega$ $\omega$	
0 0 0 0 0			
$\omega$ $\omega$ $\omega$ $\omega$ $\omega$			
$\pi$ vectors	$\pi_j$ values	$\pi$ vectors	$\pi_j$ values

useful for certain types of characters. In fact, it is this table lookup that permits the generation of new properties (Fig. 6). This is done by the simple expedient of pointing the light pen at the corresponding region of the rectangle as it is displayed. The program then stores the correct values for the new property (one for points inside the region, zero for points outside) in the 48 x 64 property vector table. From then on, the program will act as though the new property were one of the basic set.

#### 4.4.2 Hysteresis Areas

As mentioned above, when certain generalizations suggested themselves, and they were not difficult to implement and did not detract from recognition efficiency, these were carried out by the investigator. One such modification consists of a hysteresis area that lies between  $R_j$  (the region corresponding to  $P_j$ ) and its complement. In this hysteresis area, no changes in the property  $P_j$  can occur. In other words, if the pen enters the hysteresis area from the region where  $\pi_j = 1$ , even if it crosses the line between the region and its complement,  $\pi_j$  remains equal to one until the pen leaves the hysteresis area. Similarly if the pen enters the area from the region where  $\pi_j = 0$ ,  $\pi_j$  remains zero as long as the pen is in the hysteresis area. Thus, the hysteresis area acts as a buffer between two regions of the rectangle.

The effect of the hysteresis is to standardize certain forms. In Fig. 7 and Table V, a form denoting the character 'e' is shown, with and without hysteresis areas. Without hysteresis property vector sequences for this form probably would not agree with that of another form

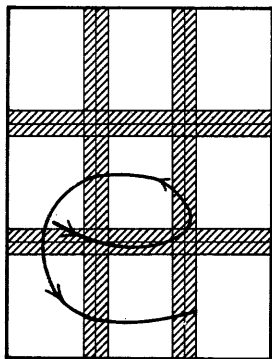


Figure 7. Use of Hysteresis Areas (See Table V)

denoting the same character. This is because slight variations in the path of the pen would result in an entirely different sequence. Sometimes the property vector sequence would record more than one crossing of certain boundaries, and other times no crossings. Consequently, the program would find it difficult to learn this character. With the hysteresis area, a certain amount of "slop" can be tolerated regardless of the position of the form.

In the program, the width of these hysteresis areas is set at five points (the raster is 48 points wide and 64 points high). The hysteresis areas are drawn approximately to scale in Fig. 7.

#### 4.4.3 Binary Trees and the Recognition Routine

For each  $j$ , the recognition routine extracts a sequence of  $\pi_j$  values from the sequence of property vectors. The program descends  $T_j$ , and retrieves the candidate set. It then forms the union of all of the candidate sets. Each candidate is credited with the success number of the trees that suggested it, and the candidate with the highest score is chosen by the decision procedure.

When the program is in training mode, the recognition routine indicates its guess and waits for feedback. If the correct character is in the candidate set of  $T_j$ , the  $j$ th tree, the success number of that tree is incremented. Otherwise, the success number is decremented, and the correct character is stored at the terminal node for the  $\pi_j$  sequence, new branches being grown where necessary.

The implementation of this process involves extensive use of list structure; the binary tree and the candidate set are each represented as a list of nodes. In addition, there is also a free storage list which contains all available space.

The nodes of the binary tree consist of three pointers. One pointer points to the node at the end of the left-hand branch, and another pointer points to the node at the end of the right-hand branch. The third pointer points to the list of characters (the candidate set) stored at the node. The nodes of the candidate

Table V. Property Vectors with and without Hysteresis (See Figure 7).

$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$	$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$
1	1	0	1	1	1	0	1
0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	1
0	1	0	1	1	1	0	0
0	0	0	1	0	1	0	0
0	1	0	1	$\omega$	$\omega$	$\omega$	$\omega$
1	1	0	1				
1	1	0	0				
0	1	0	0				
0	0	0	0				
$\omega$	$\omega$	$\omega$	$\omega$				
<b>without hysteresis</b>				<b>with hysteresis</b>			

set consist of the name of a character, represented as a nine-bit binary code, and a pointer to the next node of the list. The end of a list is indicated by a special symbol, called the null symbol instead of a pointer. Fig. 8 depicts such a structure. 0 represents the null symbol in the figure.

When the program must add a node to the binary tree or the candidate set, it obtains the necessary cells from the free storage list and constructs the node. The node is then tied into the list structure at the appropriate place by changing a null symbol to the address of the new node.

#### 4.4.4 Human Engineering Considerations

The program under discussion was originally designed to serve as a subroutine for a larger system. As such, its only function was to allow a user to input information to the larger

system by means of the cathode ray tube, and to do this in a way that seemed convenient and natural to him. Since the system must deal with users who would be complete strangers not only to this program, but also to the entire field of computers, a great deal of thought has gone into the design of the "physical appearance" of the program. This section describes and illustrates by means of photographs, etc., exactly how the user interacts with the program.

One of the first problems encountered concerned the many options the program offered. These would be of no value if the onus of remembering what these options were and how they were to be selected fell on the user. The solution adopted is the use of a control panel. This panel may be called at any time by means of a small light button displayed in the upper right-hand corner of the screen. When called,

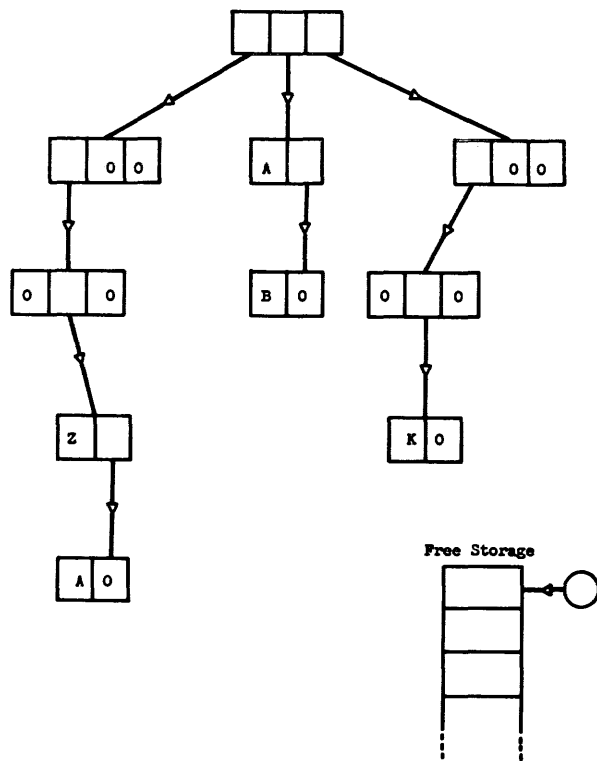


Figure 8. Internal Representation of Data Structure 0 represents the Null Symbol.

the panel displays all the options (Fig. 9). The user can select an option simply by pointing the light pen at the corresponding light button. The button lights up and the appropriate action is taken. When the panel is no longer needed, it may be dismissed by pointing the light pen at the appropriate button. This procedure has been found to be quite acceptable and intuitive to persons unfamiliar with the program.

Unless the user has previously prepared a set of trees, the first step in using the program is to place it in training mode (the program is usually operated in hysteresis mode). The user then proceeds to train the program to his style of handwriting. He does this by drawing the character on the raster, and signaling the program to recognize it (Fig. 10). The program displays its guess in the upper left-hand corner of the screen, while displaying the two-dimensional projection of the form in the center of the screen (Fig. 11), and waits for feedback.

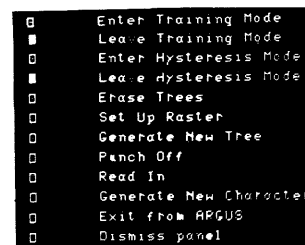


Figure 9. The Control Panel.

The control panel is called to display the various options of the program. The user selects an option by pointing at the corresponding light button, which then lights up. Note that the program is currently not in training mode and not in hysteresis mode.

If the program has guessed correctly, the user signals approval with the light pen. If the program is wrong, or could not make a guess, the user signals disapproval, and the program's alphabet is displayed so that the user can indicate the correct character (Fig. 12). This process continues until the user is satisfied with the program's proficiency (Fig. 13). He may then take the program out of training mode and write characters on the raster. When the program is asked to recognize each character, it inserts its guess ('?' if no guess) in the input string of characters at the pointer (Fig. 13). The user may move the pointer so that he can insert (or delete) characters at any point of his input string. If the program evidences weakness or certain characters, the user can return to training mode. These processes continue until the input string is exactly what the user wishes to communicate to the larger program, and the user then exits from the character recognizer. If the user wishes to retain what he has taught the program for

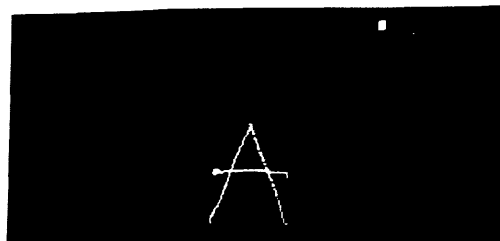


Figure 10. Drawing a Character.

The user has just completed drawing the form which denotes the character 'A.' If this is acceptable, he signals to the program to recognize it. If not, he may erase the entire form and start over. Note the light button situated in the upper right hand corner of the screen for calling the control panel. Note also the lighter hysteresis areas in the raster (Figure 7).



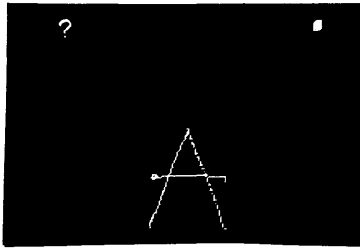


Figure 11. Indicating a Guess.

The program has been asked to identify the form whose projection is displayed in the center of the screen. The '?' indicates that it was unable to make a guess. Since the program is in training mode, the user will presumably tell it that the form denotes the character 'A.'

future use, he may request a tape of the trees and property table through the control panel. The trees and property table can then be read into the program via the control panel the next time the user wishes to use the program, thus avoiding the training process.

Although the character set basic to the program is a large one, it is quite possible that a particular user may need some special-purpose character or set of characters. These characters may be generated and are assigned nine-bit codes. They are subsequently treated exactly like the original characters. Fig. 14 shows a user generating a 'Σ.'

One other valuable option is the ability to generate new trees, which may also be done via the control panel. Fig. 15 shows the generation of a new tree needed to distinguish between 'U' and 'V.' The user has merely shaded in the corners of the raster. Upon signalling his approval, the program will modify the property vectors in the 48 x 64 table to indicate the new region. When the user calls for the PUNCH option at the end



Figure 12. Feedback.

The user has indicated disapproval, and will now communicate the correct character to the program by pointing at it with the light pen. The character set displayed is only part of the basic alphabet of the program; there are more "pages" of symbols. In addition, the user can generate new characters (Figure 14).

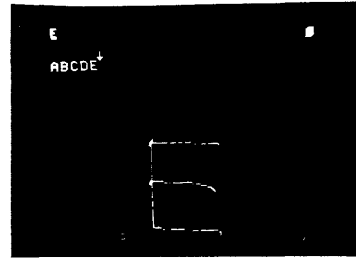


Figure 13. Correct Identification.

The program has guessed that the form denotes the character 'E.' The character displayed is underlined to indicate that the program is *sure*, i.e., all trees voted for 'E.' Below the character is the input string. The next character will be inserted at the pointer.

of a training run, all new trees are punched along with the modified property table.

The other options on the control panel are the "Enter Hysteresis Mode," "Leave Hysteresis Mode," "Erase," and "Set Up Raster." The first two of these are exactly what they suggest. In hysteresis mode, the program uses the hysteresis areas; otherwise not. The erase feature enables the user to erase all of the four original trees, destroy any new trees, and cause the property table to revert to its original form. The erase feature is used when new trees are read in with the READ option, or the user may call it if he wishes to start over and train the program differently. The set up raster option is used to determine the size and position of the raster. This allows individuals to write as large or as small as they please, and also enables them to position the raster anywhere on the scope.

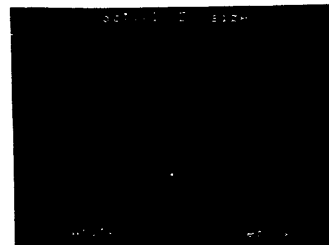


Figure 14. Generating a New Character.

The user has found a need for a character not offered by the program. He is generating a new character by pointing at the points that will make up the character. The partially formed character is displayed in the upper center of the screen between "actual" and "size." When satisfied, he indicates approval and the new character is added to the program's alphabet. The Russian and Hebrew alphabets used (Figures 16 and 17) were generated in this fashion.



Figure 15. Generating a New Tree.

The user is generating a new tree to extend the discrimination of the program. This particular tree might be useful in distinguishing between 'U' and 'V.' When the user approves, the property table will be modified accordingly.

## 5.0 DISCUSSION

The usual criterion applied to evaluating a pattern recognition program is its rate of recognition. With the program presented above, this is not too meaningful. Mermelstein and Eden,<sup>6</sup> who have also experimented with using the time sequence information in their system, report that their recognition depends on the "correspondence between the script of test samples and that of the ensemble on which the machine's representation of handwriting is based." That is certainly true here. By the nature of the program's training, if it were presented with a form encountered before, it would recognize it correctly 100 percent of the time. If it were presented with a form similar to one previously encountered, its recognition accuracy would depend on how similar this form was, and the measure of similarity would have to be circularly defined in terms of the program's properties. In particular, a form that looked identical to the human eye with another form might be very different as far as the program was concerned. This is especially true if the number of separate strokes of the pen used were not the same.

Essentially the only things that can be said about a program of this type relate to its convenience and usefulness. The program has been taught to recognize, on separate occasions, Russian characters (Fig. 16), Hebrew characters (Fig. 17), Greek characters, upper- and lower-case English characters, and a large collection of mathematical symbols (Fig. 18). It has been used by many different people, and the combination of control panel and punch-out and read-in features make it at the very least enjoyable to operate. With a little experience, after the user becomes accustomed to the idiosyncracies of the light pen, and trains the program in the variations of his hand-

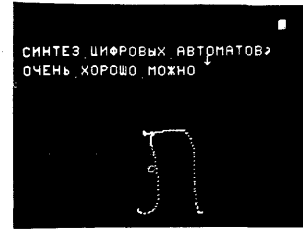


Figure 16. Recognizing Russian.

The program has been trained to recognize capital Russian letters, and an appropriate character set has been generated. The figure shows the program in operation. The letters in the input string displayed were recognized correctly by the program.

writing (surprisingly many people are not aware of the fact that they vary their style of handwriting from sample to sample), the resulting man-machine system becomes quite effective. Again no quantitative results can be offered here other than observations on use of the program by a considerable number of individuals.

The results seem to indicate that the use of the time sequence is a powerful tool in pattern recognition. It enables a program using fairly simple properties to achieve a high rate of recognition. One way to improve the program might be to include more sophisticated properties other than the position of the pen. An immediate improvement would be to include a property that detected sharp corners by noticing changes in the velocity of the pen. Other properties might note curvature or angle. Developing a language that described a wide class of properties would generalize the program even further. The user would then be able to communicate subtle differences in forms by means of fairly abstract concepts.

The final goal of this type of approach would be to have the system generate new properties to help distinguish between similar forms without user intervention.<sup>10</sup> One possible way

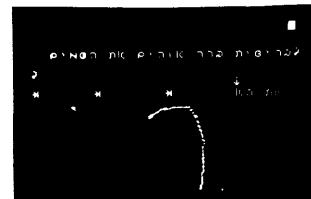


Figure 17. Recognizing Hebrew.

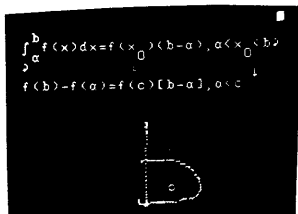


Figure 18. Recognizing Mathematics.

of doing this would be to give the program a large pool of potentially useful properties and have the program select those applicable.

Another extension of the program's applicability lies in relaxing the real time restriction. If line tracing is used on input characters, the program could recover a canonical time sequence from a figure. This would allow these techniques to be applied to recognizing printed characters.

Similarly, if a technique were used similar to Freeman's<sup>3</sup> to encode the character as it was being drawn, the normalization problem might also be avoided. This would allow the user complete freedom of the scope. In this case, the system would begin to rival the typewriter for convenience, and would make computers very accessible to any and all potential users.

REFERENCES

1. DAVID, E. E. and SELFRIDGE, O. G. Eyes and Ears for Computers. *Proceedings of the IRE*, 1962, 50(5), 1-9.
2. DIMOND, T L. Devices for Reading Handwritten Characters. *Proceedings of*

- the Eastern Joint Computer Conference*, 1957, 232-237.
3. FREEMAN, H. On the Encoding of Arbitrary Geometric Configurations. *IRE Transactions on Electronic Computers*, 1961, EC-10(2), 260-268.
4. KUHL, F. Classification and Recognition of Hand Printed Characters. *IEEE International Convention Record*, 1963, Part 4, 75-93.
5. MARILL, T. *et al.* Cyclops-1: A Second Generation Recognition System. *Proceedings of the Fall Joint Computer Conference*, 1963, 24, 27-33.
6. MERMELSTEIN, P. and EDEN, M. Experiments on Computer Recognition of Connected Handwritten Words. *Information and Control*, 1964, 7(2), 255-270.
7. MINSKY, M. L. A Selected Descriptor-Indexed Bibliography to the Literature on Artificial Intelligence. *IRE Transactions on Human Factors in Electronics*, 1961, HFE-2(1), 39-55.
8. MINSKY, M. L. Steps toward Artificial Intelligence. *Proceedings of the IRE*, 1961, 49(1), 8-30.
9. UHR, L. Pattern Recognition Computers as Models for Form Perception. *Psychological Bulletin*, 1963, 60(1), 40-73.
10. UHR, L. and VOSSLER, C. A Pattern Recognition Program that Generates, Evaluates, and Adjusts its own Operators. *Proceedings of the Western Joint Computer Conference*, 1961, 19, 555-569.



# A COMPUTER PROGRAM WHICH "UNDERSTANDS"\*

*Bertram Raphael*  
*University of California, Berkeley, California*

## I. INTRODUCTION

This paper describes a computer program which demonstrates one approach to building an "intelligent" machine. The computer system called SIR—Semantic Information Retriever—exhibits some humanlike conversational behavior and appears to have certain cognitive abilities. The conversation presented in Figure 1 between a person (identified by "\*\*\*\*.") and SIR illustrates some of the system's capabilities.

"Understanding" is difficult to define. A basic assumption here is that understanding can be demonstrated by dialogue; *i.e.*, a computer should be considered able to "understand" if it can remember what it is told, answer questions, and make other responses which a human observer considers reasonable.

In order to make "reasonable" responses the computer must not only be able to echo, upon request, facts it has been given; it must be able to select (from a large store) facts which are relevant to a particular question, and must be able to recognize the logical implications of those facts. One way to satisfy these requirements is to utilize a suitable internal representation, a "model", for stored information.

This paper describes SIR, a prototype of a general-purpose "understanding" machine. SIR demonstrates how conversational and deductive abilities can be obtained through use of a model

which can represent semantic content from a variety of subject areas.

## II. QUESTION-ANSWERING SYSTEMS

Several computer programs have previously been written whose behaviors are somewhat similar to that of SIR. The following are among the best-known existing systems:

1. "Baseball: An Automatic Question-Answerer".<sup>1</sup> This system draws answers to a wide range of English questions from a predetermined, fixed, hierarchical data structure.
2. Phillips' "Question-Answering Routine".<sup>2</sup> This early example of a question-answerer contains the important feature that its corpus of data is automatically extended with each new input sentence. However, the answering process involves a statement-by-statement search of the entire corpus.
3. "SYNTHEx".<sup>3</sup> The SYNTHEx approach is to find answers in an indexed natural language text by means of grammatical dependencies. The ability of the system can be improved only by increasing the complexity of the associated grammar.
4. Lindsay's "Semantic Analyzing Machine".<sup>4</sup> Lindsay takes advantage of special properties of a particular semantic relationship by using a form of "family tree" as the internal computer model for

\* The work reported herein was supported in part by the MIT Computation Center and, in part, by Project MAC, an MIT research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01). This report is based on a doctoral thesis for the MIT Mathematics Department, June, 1964.

```

(***. EVERY BOY IS A PERSON)
(I UNDERSTAND)

(***. A FINGER IS PART OF A HAND)
(I UNDERSTAND)

(***. EACH PERSON HAS TWO HANDS)
(THE ABOVE SENTENCE IS AMBIGUOUS ** PLEASE RE-PHRASE IT)

(***. THERE ARE TWO HANDS ON EACH PERSON)
(I UNDERSTAND)

(***. HOW MANY FINGERS DOES JOHN HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I DON'T KNOW WHETHER FINGER IS PART OF JOHN)

(***. JOHN IS A BOY)
(I UNDERSTAND)

(***. HOW MANY FINGERS DOES JOHN HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(HOW MANY FINGER PER HAND Q)

(***. EVERY HAND HAS 5 FINGERS)

(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I UNDERSTAND)

(***. HOW MANY FINGERS DOES JOHN HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(THE ANSWER IS 10)

(***. HOW MANY AUTOMOBILES DOES JOHN HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** PLEASE RE-PHRASE IT)

(***. WHO IS PRESIDENT OF THE UNITED STATES Q)
(STATEMENT FORM NOT RECOGNIZED)

(***. THE BOY IS JUST TO THE LEFT OF THE TABLE)
(G02840 IS A TABLE)
(I UNDERSTAND)

(***. THE LAMP IS JUST TO THE LEFT OF THE TABLE)
(G02841 IS A LAMP)
(THE ABOVE STATEMENT IS IMPOSSIBLE)

(***. THE TABLE IS TO THE RIGHT OF THE CHAIR)
(G02842 IS A CHAIR)
(I UNDERSTAND)

(***. WHAT IS THE RELATIVE POSITION OF A PERSON Q)
(THE LEFT-TO-RIGHT ORDER IS AS FOLLOWS)
(CHAIR (BOY TABLE))

```

Figure 1. Sample Conversation.

kinship information. Of course, this "tree" model is not a good representation for other kinds of relationships.

5. Darlington's translation to symbolic logic.<sup>5</sup> This program shows how certain English problems can be translated into logical terms and attacked by formal methods. As with Lindsay's system, the approach used here has limited applicability.
6. Bennett's computer program for word relations.<sup>6</sup> This system, in which words are related by means of the English formats in which they appear, extracts and stores semantic information in a limited and somewhat awkward way.

7. Bobrow's "STUDENT".<sup>7</sup> This new system deals with linguistic problems which arise while analyzing those English statements which describe mathematical relationships. The program translates "word problems" directly into algebraic expressions.

Each of the above systems has certain interesting features, some of which have influenced the design of SIR. The goal of the present work, however, has been to find and demonstrate a question-answering mechanism which is highly automatic, widely applicable, and reasonably efficient; one which overcomes many of the limitations of the above systems. An appropriate *model* is the key to this mechanism.

### III. THE SIR MODEL

The SIR model is the collection of data which SIR subprograms may refer to in the course of question-answering. It is a "semantic" model in two senses: 1) The *information* stored in the model is intended to approximate the linguistic "semantic content" or "meaning" of English text; 2) The *structure* of the model is derived from the "semantic" model structures of mathematical logic.

Many linguists and logicians have considered problems of "semantics" and offered definitions of "meaning".<sup>8,9,10,11,12,13</sup> Few of these discussions are sufficiently concrete to be suitable for computer implementation. However, the idea of representing meaning by *word associations*, suggested by several previous authors, has been adapted as the basis of the SIR model. The resulting word-association structure of the model is general enough to be useful in a wide variety of subject areas; yet, the stored information is specific enough to provide convenient accessibility of relevant facts and, therefore, efficient question-answering.

#### STRUCTURE OF THE MODEL

The SIR model is structured by means of *property-lists* (sometimes called *description-lists*). A property-list is a sequence of pairs of elements, and the entire list is associated with a particular object. The first element of each pair is an *attribute* applicable to a class of ob-

jects, and the second element of the pair is the *value* of that attribute for the object described.

If an English statement asserts that relation  $R$  holds between objects or classes named  $x$  and  $y$ , or equivalently, that the word "x" is associated with the word "y" in a manner specified by the word "R", then this relationship is represented in the SIR model by attribute-value pairs placed on the property-lists of both  $x$  and  $y$ . Each attribute specifies a relation, and the value paired with the attribute (the value of the attribute) indicates which other objects are related to the described object by means of the specified relation, *i.e.*, which other words are associated with the described word in the way specified by the attribute word.

Since, in general, relations are not symmetric, relation  $R$  must be factored into two relations  $R1$  and  $R2$  so that, if  $R$  holds between  $x$  and  $y$ , one can say that  $y$  stands in relation  $R1$  to  $x$  and  $x$  stands in the inverse relation  $R2$  to  $y$ .

For example, consider the sentence, "Every boy is a person." SIR takes this sentence to mean that a *set-inclusion* relation holds between "BOY" and "PERSON". The factor relations  $R1$  and  $R2$  are "SUPERSET" and "SUBSET", respectively. The fact specified by the above sentence can be represented in the model by the attribute-value pair "(SUPERSET, PERSON)" on the property-list of "BOY", and the pair "(SUBSET, BOY)" on the property-list of "PERSON".

An attribute can only appear once on a given property-list. However, the *value* of an attribute may be a *list* containing several object names. For added flexibility, elements of these value lists may themselves be in property-list format so that they can hold descriptive information as well as object-names. The first item on such sub-property-lists is the flag "PLIST" which indicates that a property-list follows.

For example, after the system learns that "A person has two hands" and "A finger is part of a person", the property-list of "PERSON" would contain the attribute-value pair:

```
(SUBPART, ((PLIST (NAME, HAND)
(NUMBER, 2)) (PLIST (NAME, FIN-
GER))))
```

These general links are the principal mechanism for structuring the model.

#### IV. NATURAL LANGUAGE INPUT

The language which is most convenient to the users of SIR is natural English; therefore, the input and response languages of the SIR system should be reasonably close to natural English. Since its internal information representation is a relational model, SIR is faced with the difficult problem of extracting word associations from natural language.

The work reported in this paper is concerned with the ability of a computer to utilize relational information in order to produce intelligent behavior. The linguistic problem of transforming natural language into a usable predicate form is only of peripheral interest here. The following outlines a superficial but adequate method for solving this linguistic problem in the present context. Reference<sup>14</sup> contains further details of this method.

SIR recognizes only a small number of sentence forms, each of which corresponds to a particular relation. The input language is defined by a list of *formats*. Each format is a string of constants and variables, and an applicability test is associated with each variable. An input sentence is *recognized* if it is matched by the constants in some format, and if its substrings corresponding to each variable in that format satisfy the corresponding tests. Special functions associated with each format then extract appropriate word associations from recognized sentences, and perform the desired storage or retrieval operations in the model.

For example, the sentence, "Every boy is a person", is recognized by the format, " $x$  is  $y$ ". Applicability tests check that the substrings corresponding to  $x$  and  $y$  are each two words long, the first of which is a member of the set { a, an, every, any, each }. The associated function then creates set-inclusion links between "BOY" and "PERSON" in the model.

Some formats do not uniquely determine word relations. As an example of how such ambiguity may be treated, SIR considers the verb "to have" as meaning either "to have attached as parts" or "to own", *e.g.*, "John has ten fingers" *vs.* "John has three automobiles". The function associated with the format " $x$  has  $n$   $y$ " must resolve this semantic ambiguity before it can operate on the model. The ambiguity

is resolved, as described in Section V, on the basis of word-associations in the model which were created because of previous, unambiguous input sentences.

SIR always makes reports of its actions. The conversation of Figure 1 was produced by an abbreviated-response mode in which only directly relevant responses were printed. Alternatively, the system can provide a running commentary of its activities. Although less reada-

ble, this full-response mode was a significant debugging aid. Figure 2 shows the dialogue of Figure 1 in that mode.

## V. THE SIR PROGRAM

SIR is programmed in the LISP language,<sup>15</sup> a list-processing computer language<sup>16</sup> well suited for model building and searching procedures. The operation of the program is described in detail in<sup>14</sup>. Here we shall observe

```
(THE NEXT SENTENCE IS . .)
(EVERY BOY IS A PERSON)

(THE FUNCTION USED IS . .)
SETR-SELECT
((GENERIC . BOY) (GENERIC . PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETR
(BOY PERSON)
(ITS REPLY . .)
(I UNDERSTAND THE SUPERSET RELATION BETWEEN PERSON AND BOY)
(I UNDERSTAND THE SUBSET RELATION BETWEEN BOY AND PERSON)

(THE NEXT SENTENCE IS . .)
(A FINGER IS PART OF A HAND)

(THE FUNCTION USED IS . .)
PARTR-SELECT
((GENERIC . FINGER) (GENERIC . HAND))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
PARTR
(FINGER HAND)
(ITS REPLY . .)
(I UNDERSTAND THE SUBPART-OF-EACH RELATION BETWEEN FINGER AND HAND)
(I UNDERSTAND THE SUPERPART-OF-EACH RELATION BETWEEN HAND AND FINGER)

(THE NEXT SENTENCE IS . .)
(EACH PERSON HAS TWO HANDS)

(THE FUNCTION USED IS . .)
HASH-RESOLVE
(2 . HAND) (GENERIC . PERSON)
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** PLEASE RE-PHRASE IT)

(THE NEXT SENTENCE IS . .)
(THERE ARE TWO HANDS ON EACH PERSON)

(THE FUNCTION USED IS . .)
PARTRN-SELECT
((GENERIC . PERSON) (2 . HAND))
(THE REPLY . .)
(I UNDERSTAND THE SUPERPART-OF-EACH RELATION BETWEEN PERSON AND HAND)
(I REALIZE THE NUMBER RELATION BETWEEN 2 AND (PLIST NAME PERSON))
(I UNDERSTAND THE SUBPART-OF-EACH RELATION BETWEEN HAND AND PERSON)
(I REALIZE THE NUMBER RELATION BETWEEN 2 AND (PLIST NAME HAND))

(THE NEXT SENTENCE IS . .)
(HOW MANY FINGERS DOES JOHN HAVE Q)

(THE FUNCTION USED IS . .)
HAVE-RESOLVE
(FINGER (UNIQUE . JOHN))
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I DON'T KNOW WHETHER FINGER IS PART OF JOHN)

(THE NEXT SENTENCE IS . .)
(JOHN IS A BOY)

(THE FUNCTION USED IS . .)
SETR-SELECT
((UNIQUE . JOHN) (GENERIC . BOY))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
SETRN
(JOHN BOY)
(ITS REPLY . .)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN JOHN AND BOY)
(I UNDERSTAND THE MEMBER RELATION BETWEEN BOY AND JOHN)

(THE NEXT SENTENCE IS . .)
(HOW MANY FINGERS DOES JOHN HAVE Q)

(THE FUNCTION USED IS . .)
HAVE-RESOLVE
(FINGER (UNIQUE . JOHN))

(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I KNOW THE SUPERPART-OF-EACH RELATION BETWEEN HAND AND FINGER)
(HOW MANY FINGER PER HAND Q)

(THE NEXT SENTENCE IS . .)
(EVERY HAND HAS 5 FINGERS)

(THE FUNCTION USED IS . .)
HASH-RESOLVE
(5 . FINGER) (GENERIC . HAND)
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))

(I KNOW THE SUPERPART-OF-EACH RELATION BETWEEN HAND AND FINGER)
(I REALIZE THE NUMBER RELATION BETWEEN 5 AND (PLIST NAME HAND))
(I KNOW THE SUBPART-OF-EACH RELATION BETWEEN FINGER AND HAND)
(I REALIZE THE NUMBER RELATION BETWEEN 5 AND (PLIST NAME FINGER))

(THE NEXT SENTENCE IS . .)
(HOW MANY FINGERS DOES JOHN HAVE Q)

(THE FUNCTION USED IS . .)
HAVE-RESOLVE
(FINGER (UNIQUE . JOHN))
(THE REPLY . .)
(THE ABOVE SENTENCE IS AMBIGUOUS ** PLEASE RE-PHRASE IT)

(THE NEXT SENTENCE IS . .)
(HOW IS PRESIDENT OF THE UNITED STATES Q)

(STATEMENT FORM NOT RECOGNIZED)

(THE NEXT SENTENCE IS . .)
(THE BOY IS JUST TO THE LEFT OF THE TABLE)

(THE FUNCTION USED IS . .)
JRIGHT-SELECT
((SPECIFIC . TABLE) (SPECIFIC . BOY))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
JRIGHT
(TABLE BOY)
(ITS REPLY . .)
(G02040 IS A TABLE)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02040 AND TABLE)
(I UNDERSTAND THE MEMBER RELATION BETWEEN TABLE AND G02040)
(I REALIZE THE JRIGHT RELATION BETWEEN TABLE AND BOY)
(I REALIZE THE JLEFT RELATION BETWEEN BOY AND TABLE)

(THE NEXT SENTENCE IS . .)
(THE LAMP IS JUST TO THE LEFT OF THE TABLE)

(THE FUNCTION USED IS . .)
JRIGHT-SELECT
((SPECIFIC . TABLE) (SPECIFIC . LAMP))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
JRIGHT
(TABLE LAMP)
(ITS REPLY . .)
(G02041 IS A LAMP)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02041 AND LAMP)
(I UNDERSTAND THE MEMBER RELATION BETWEEN LAMP AND G02041)
(THE ABOVE STATEMENT IS IMPOSSIBLE)

(THE NEXT SENTENCE IS . .)
(THE TABLE IS TO THE RIGHT OF THE CHAIR)

(THE FUNCTION USED IS . .)
RIGHT-SELECT
((SPECIFIC . TABLE) (SPECIFIC . CHAIR))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
RIGHT
(TABLE CHAIR)
(ITS REPLY . .)
(G02042 IS A CHAIR)
(I UNDERSTAND THE ELEMENTS RELATION BETWEEN G02042 AND CHAIR)
(I UNDERSTAND THE MEMBER RELATION BETWEEN CHAIR AND G02042)
(I UNDERSTAND THE RIGHT RELATION BETWEEN TABLE AND CHAIR)
(I UNDERSTAND THE LEFT RELATION BETWEEN CHAIR AND TABLE)

(THE NEXT SENTENCE IS . .)
(WHAT IS THE RELATIVE POSITION OF A PERSON Q)

(THE FUNCTION USED IS . .)
LOC-SELECT
((GENERIC . PERSON))
(THE REPLY . .)
(THE SUB-FUNCTION USED IS . .)
LOCATEP
(PERSON)
(ITS REPLY . .)
(THE LEFT-TO-RIGHT ORDER IS AS FOLLOWS)
(CHAIR (BOY TABLE))
```

Figure 2. Sample Conversation in Full-Response Mode.



the system's behavior by means of annotated examples.

Each part of Figure 3 is a conversation between a person and SIR designed to illustrate SIR's ability to "understand" a different relation or group of relations. Each part starts with a "clean" memory, *i.e.*, an empty model and no knowledge of vocabulary except for format constants. The following notes, keyed to Figure 3, should help clarify some of the responses:

- a1. The response, "I UNDERSTAND", indicates that some desired link has been created or discovered in the model.
- a2. "IS" and "IS AN EXAMPLE OF" are equivalent formats in this context.
- a3. In general, question-answering routines in SIR can perform all necessary logical deductions. "Q" should be read as a question mark.
- a4. The program responds "SOMETIMES" to the question, "Is an  $x$  a  $y$ ?" if it can deduce, from existing linkages, that  $y$  is a subclass of  $x$ .
- a5. "Insufficient Information" is the most common report of failure. The present system does not handle deductions from negative premises, *e.g.*, "Every boy is not a girl."
- b1. Absence of an indefinite article indicates that MAX is an individual and, therefore, is an element, rather than subclass, of IBM-7094.
- b2. "THE BOY" requires the existence of a unique element of the class "BOY". SIR assigns "G" names to anonymous individuals.
- b3. "THE BOY" is assumed to be G02840.
- b4. JOHN and G02840 may be different boys, making "THE BOY" ambiguous.
- c1. Two words are linked by the attribute "EQUIV" if they are different names for the same object. Deduction procedures must allow for possible equivalences.
- d1. In these cases attributes on the property-list of the name of a set may describe properties of every element of the set, rather than of the set as a whole. The hyphens in "PAIR-OF-RED-SUS-

- PENDERS" are necessary to avoid confusing the format recognition scheme.
- d2. Each question-answering program may check for certain special cases before attempting standard deduction procedures.
- e1. "Specific" information appears on the property-lists of individuals rather than of sets. Deduction procedures must use both general and specific information.
- f1. See d1.
- f2. See d2.
- f3. The system discovers that a NOSTRIL is part of a PERSON, and then answers the question, "Is a living-creature a person?"
- h1. In this and future sentences, the system reports that the sentence form is ambiguous (because "have" may mean either "have as parts" or "own"), but it has been able to make a "reasonable" assumption about the intended meaning and is proceeding from there. (See description of Figure 4b.) In this case, "How many" cannot be answered unless a part-whole relationship can be established first.
- h2. "NUMBER" is an attribute which is placed on sub-property-lists associated with part-whole relationships.
- h3. Here a "NUMBER" attribute is missing along the chain of links which established the part-whole relationship.
  - i1. "JUST. . ." requires adjacency.
  - i2. In this section new links are created only if they are consistent with already "known" facts.
  - i3. A "WHERE" question requests location information obtainable from direct links only.
  - i4. "WHAT IS THE POSITION" requests the construction of a linear ordering of objects, as far as available linkage information permits. Inner parentheses in the order list indicate adjacent objects.

#### SPECIAL FEATURES

Figure 4 illustrates three special features of the SIR system.

- a) Exception principle: General information about "all the elements" of a set is con-

a. SET-INCLUSION	c. EQUIVALENCE	f. PART-WHOLE, GENERAL	g. PART-WHOLE, SPECIFIC
<p>1 [***. EVERY KEYPUNCH-OPERATOR IS A GIRL] (I UNDERSTAND)</p> <p>2 [***. ANY GIRL IS AN EXAMPLE OF A PERSON] (I UNDERSTAND)</p> <p>3 [***. IS A KEYPUNCH-OPERATOR A PERSON Q] YES [***. IS A PERSON A PERSON Q] YES [***. IS A PERSON A GIRL Q] SOMETIMES</p> <p>4 [***. IS A MONKEY A KEYPUNCH-OPERATOR Q] (INSUFFICIENT INFORMATION)</p> <p>5 [***. IS A MONKEY A KEYPUNCH-OPERATOR Q] (INSUFFICIENT INFORMATION)</p>	<p>[***. THE MAN IS A JERK] (GO2840 IS A MAN) (I UNDERSTAND)</p> <p>[***. JACK IS A DOPE] (I UNDERSTAND)</p> <p>[***. JOHN IS JACK] (I UNDERSTAND)</p> <p>[***. IS JOHN A DOPE Q] YES</p> <p>[***. IS THE MAN A DOPE Q] (INSUFFICIENT INFORMATION)</p> <p>[***. JOHN IS THE MAN] (I UNDERSTAND)</p> <p>[***. IS THE MAN A DOPE Q] YES</p> <p>[***. JIM IS A MAN] (I UNDERSTAND)</p> <p>[***. IS THE MAN A DOPE Q] (WHICH MAN . . . (GO2840 JIM))</p>	<p>1 [***. A NOSE IS PART OF A PERSON] (I UNDERSTAND) [***. A NOSTRIL IS A PART OF A NOSE] (I UNDERSTAND) [***. A PROFESSOR IS A TEACHER] (I UNDERSTAND) [***. A TEACHER IS A PERSON] (I UNDERSTAND) [***. IS A NOSTRIL PART OF A PROFESSOR Q] YES [***. IS A NOSE PART OF A NOSE Q] 2 (NO PART MEANS PROPER SUBPART) [***. A PERSON IS A LIVING-CREATURE] (I UNDERSTAND) 3 [***. IS A NOSTRIL PART OF A LIVING-CREATURE Q] SOMETIMES [***. IS A LIVING-CREATURE PART OF A NOSE Q] (NO NOSE IS SOMETIMES PART OF LIVING-CREATURE)</p>	<p>[***. A VAN-DYKE IS PART OF FERREN] (I UNDERSTAND) [***. A VAN-DYKE IS A BEARD] (I UNDERSTAND) [***. IS A BEARD PART OF FERREN Q] YES [***. A CRT IS A DISPLAY-DEVICE] (I UNDERSTAND) [***. A CRT IS PART OF THE PDP-1] (GO2840 IS A PDP-1) (I UNDERSTAND) [***. SAN IS THE PDP-1] (I UNDERSTAND) [***. A SCREEN IS PART OF EVERY DISPLAY-DEVICE] (I UNDERSTAND) [***. IS A SCREEN PART OF SAN Q] YES [***. A BEARD IS PART OF A BEATNIK] (I UNDERSTAND) [***. EVERY COFFEE-HOUSE-CUSTOMER IS A BEATNIK] (I UNDERSTAND) [***. BUZZ IS A COFFEE-HOUSE-CUSTOMER] (I UNDERSTAND) [***. IS A BEARD PART OF BUZZ Q] YES</p>
<p>b. SET-MEMBERSHIP</p> <p>1 [***. MAX IS AN IBM-7094] (I UNDERSTAND) [***. AN IBM-7094 IS A COMPUTER] (I UNDERSTAND) [***. IS MAX A COMPUTER Q] YES</p> <p>2 [***. THE BOY IS AN MIT-STUDENT] (GO2840 IS A BOY) (I UNDERSTAND) [***. EVERY MIT-STUDENT IS A BRIGHT-PERSON] (I UNDERSTAND) 3 [***. IS THE BOY A BRIGHT-PERSON Q] YES [***. JOHN IS A BOY] (I UNDERSTAND) 4 [***. IS THE BOY A BRIGHT-PERSON Q] (WHICH BOY . . . (GO2840 JOHN))</p>	<p>d. OWNERSHIP, GENERAL</p> <p>1 [***. EVERY FIREMAN OWNS A PAIR-OF-RED-SUSPENDERS] (I UNDERSTAND) 2 [***. DOES A PAIR-OF-RED-SUSPENDERS OWN A PAIR-OF-RED-SUSPENDERS Q] (NO ** THEY ARE THE SAME) [***. DOES A DOCTOR OWN A PAIR-OF-RED-SUSPENDERS Q] (INSUFFICIENT INFORMATION) [***. A FIRECHIEF IS A FIREMAN] (I UNDERSTAND) [***. DOES A FIRECHIEF OWN A PAIR-OF-RED-SUSPENDERS Q] YES</p>	<p>i. LEFT-TO-RIGHT POSITION</p> <p>[***. THE TELEPHONE IS JUST TO THE RIGHT OF THE BOOK] (GO2840 IS A TELEPHONE) (GO2841 IS A BOOK) (I UNDERSTAND) [***. THE TELEPHONE IS JUST TO THE LEFT OF THE PAD] (GO2842 IS A PAD) (I UNDERSTAND) 1 [***. IS THE PAD JUST TO THE RIGHT OF THE BOOK Q] NO [***. IS THE BOOK TO THE LEFT OF THE PAD Q] YES [***. THE PAD IS TO THE RIGHT OF THE TELEPHONE] (THE ABOVE STATEMENT IS ALREADY KNOWN) 2 [***. THE PAD IS TO THE LEFT OF THE TELEPHONE] (THE ABOVE STATEMENT IS IMPOSSIBLE) [***. THE ASH-TRAY IS TO THE LEFT OF THE BOOK] (GO2843 IS A ASH-TRAY) (I UNDERSTAND) [***. THE PENCIL IS TO THE LEFT OF THE PAD] (GO2844 IS A PENCIL) (I UNDERSTAND) [***. THE PAPER IS TO THE RIGHT OF THE TELEPHONE] (GO2845 IS A PAPER) (I UNDERSTAND) 3 [***. WHERE IS THE PAD Q] (JUST TO THE RIGHT OF THE TELEPHONE) (SOMEWHERE TO THE RIGHT OF THE FOLLOWING . . . (PENCIL)) [***. WHAT IS THE POSITION OF THE PAD Q] 4 (THE LEFT-TO-RIGHT ORDER IS AS FOLLOWS) (ASH-TRAY (BOOK TELEPHONE PAD) PENCIL) (TO FURTHER SPECIFY THE POSITIONS YOU MUST INDICATE WHERE THE PENCIL IS WITH RESPECT TO THE ASH-TRAY) [***. THE BOOK IS JUST TO THE RIGHT OF THE ASH-TRAY] (I UNDERSTAND) [***. WHAT IS THE POSITION OF THE PAD Q] (THE LEFT-TO-RIGHT ORDER IS AS FOLLOWS) (PENCIL (ASH-TRAY BOOK TELEPHONE PAD) PAPER) [***. A TELEPHONE IS AN AUDIO-TRANSDUCER] (I UNDERSTAND) [***. A DIAPHRAGM IS PART OF AN AUDIO-TRANSDUCER] (I UNDERSTAND) [***. WHERE IS A DIAPHRAGM Q] (JUST TO THE LEFT OF THE PAD) (JUST TO THE RIGHT OF THE BOOK) (SOMEWHERE TO THE LEFT OF THE FOLLOWING . . . (PAPER))</p>	<p>e. OWNERSHIP, SPECIFIC</p> <p>1 [***. ALFRED OWNS A LOG-LOG-DECITRIG] (I UNDERSTAND) [***. A LOG-LOG-DECITRIG IS A SLIDE-RULE] (I UNDERSTAND) [***. DOES ALFRED OWN A SLIDE-RULE Q] YES [***. EVERY ENGINEERING-STUDENT OWNS A SLIDE-RULE] (I UNDERSTAND) [***. VERNON IS A TECH-MAN] (I UNDERSTAND) [***. A TECH-MAN IS AN ENGINEERING-STUDENT] (I UNDERSTAND) [***. DOES VERNON OWN A SLIDE-RULE Q] YES [***. DOES AN ENGINEERING-STUDENT OWN THE LOG-LOG-DECITRIG Q] (GO2840 IS A LOG-LOG-DECITRIG) (INSUFFICIENT INFORMATION) [***. ALFRED IS A TECH-MAN] (I UNDERSTAND) [***. DOES AN ENGINEERING-STUDENT OWN THE LOG-LOG-DECITRIG Q] YES</p>

Figure 3. Selected Conversations.

## a. EXCEPTION PRINCIPLE

```

(***.  THERE ARE 5 FINGERS ON EVERY HAND)
(I UNDERSTAND)

(***.  THERE ARE TWO HANDS ON A PERSON)
(I UNDERSTAND)

(***.  A BOY IS A PERSON)
(I UNDERSTAND)

(***.  TOM IS A BOY)
(I UNDERSTAND)

(***.  DICK IS A BOY)
(I UNDERSTAND)

(***.  HARRY IS A BOY)
(I UNDERSTAND)

(***.  TOM HAS NINE FINGERS)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I UNDERSTAND)

(***.  DICK HAS ONE HAND)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I UNDERSTAND)

(***.  HOW MANY FINGERS DOES TOM HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(THE ANSWER IS 9)

(***.  HOW MANY FINGERS DOES DICK HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(THE ANSWER IS 5)

(***.  HOW MANY FINGERS DOES HARRY HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(THE ANSWER IS 10)

(***.  HOW MANY FINGERS DOES JOE HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I DON'T KNOW WHETHER FINGER IS PART OF JOE)

```

## b. RESOLVING AMBIGUITIES

```

(***.  JOHN IS A PERSON)
(I UNDERSTAND)

(***.  DICK IS A PERSON)
(I UNDERSTAND)

(***.  A CHAIN IS PART OF A BICYCLE)
(I UNDERSTAND)

(***.  THE POWER-SAW HAS A CHAIN)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(602840 IS A POWER-SAW)
(I UNDERSTAND)

(***.  JOHN OWNS A CHAIN)
(I UNDERSTAND)

(***.  DICK HAS A CHAIN)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (OWNS))
(I UNDERSTAND)

(***.  THE CUCKOO-CLOCK HAS A CHAIN)
(THE ABOVE SENTENCE IS AMBIGUOUS ** PLEASE RE-PHRASE IT)

```

## c. STREAMLINING LINKAGES

```

(***.  JOHN IS A PERSON)
(I UNDERSTAND)

(***.  JOHN IS A TECH-MAN)
(I UNDERSTAND)

(***.  JOHN IS A BOY)
(I UNDERSTAND)

(***.  JOHN IS A STUDENT)
(I UNDERSTAND)

(***.  JOHN IS A BRIGHT-PERSON)
(I UNDERSTAND)

(***.  EVERY BOY IS A PERSON)
(I UNDERSTAND)

(***.  EVERY TECH-MAN IS A PERSON)
(I UNDERSTAND)

(***.  EVERY TECH-MAN IS A BRIGHT-PERSON)
(I UNDERSTAND)

(***.  EVERY TECH-MAN IS A STUDENT)
(I UNDERSTAND)

(***.  EVERY BRIGHT-PERSON IS A PERSON)
(I UNDERSTAND)

(***.  EVERY STUDENT IS A BRIGHT-PERSON)
(I UNDERSTAND)

(***.  EVERY STUDENT IS A PERSON)
(I UNDERSTAND)

END OF EVALQUOTE, VALUE IS **
(NO MORE INPUT SENTENCES)

FUNCTION  EVALQUOTE  HAS BEEN ENTERED, ARGUMENTS..
STREAMLINE
(JOHN)
(I FORGET THE MEMBER-ELEMENTS RELATIONS BETWEEN PERSON AND JOHN)
(I FORGET THE MEMBER-ELEMENTS RELATIONS BETWEEN STUDENT AND JOHN)
(I FORGET THE MEMBER-ELEMENTS RELATIONS BETWEEN BRIGHT-PERSON AND JOHN)
(I FORGET THE SET-INCLUSION RELATION BETWEEN PERSON AND TECH-MAN)
(I FORGET THE SET-INCLUSION RELATION BETWEEN BRIGHT-PERSON AND TECH-MAN)
(I FORGET THE SET-INCLUSION RELATION BETWEEN PERSON AND STUDENT)
END OF EVALQUOTE, VALUE IS **
NIL

```

Figure 4. Special Features.

sidered to apply to particular elements only in the absence of more specific information about those elements. Thus, it is not necessarily contradictory to learn that "Mammals are land animals"; and, yet, "A whale is a mammal which always lives in water." In the program, this idea is implemented by always referring for desired information to the property-list of the individual concerned *before* looking at the descriptions of sets to which the individual belongs.

True "understanding" is frequently characterized as an ability to reason appropriately with facts which appear contradictory or paradoxical—an ability generally considered to be beyond formal logical procedures. Here, on the other hand, we see that a simple algorithm used in conjunction with the SIR model is sufficient to produce reasonable, human-like conversational behavior in just such a paradoxical situation.

- b) Resolving ambiguities: The criteria used by the program to decide whether "has" in the format "x has y" should be interpreted "has as parts" or "owns" are the following:

Let P be the proposition, "either  $y$  is known to be part of something, or  $y$  is an element of some set whose elements are known to be parts of something."

Let N be the proposition, "either  $y$  is known to be owned by something, or  $y$  is an element of some set whose elements are known to be owned by something."

- 1) If  $P \wedge \sim N$ , assume "has" means "has as parts."
- 2) If  $\sim P \wedge N$ , assume "has" means "owns."
- 3) If  $\sim P \wedge \sim N$ , give up and ask for re-phrasing.

Let P' be the proposition,

$$(\ u) [[ [y \text{ is known to be part of } u] \vee [y \text{ is an element of some set whose elements are known to be parts of the elements of } u]] \wedge (\ w) [[ [u \in w \vee u \subset w] \wedge [x \in w \vee x \subset w]]].$$

Let N' be the proposition,

$$(\ u) [[ [y \text{ is known to be owned by } u] \vee [y \text{ is an element of some set whose elements are known to be owned by the elements of } u]] \wedge (\ w) [[ [u \in w \vee u \subset w] \wedge [x \in w \vee x \subset w]]].$$

- 4) If  $P' \wedge \sim N'$ , assume "has" means "has as parts."
- 5) If  $\sim P' \wedge \sim N'$ , assume "has" means "owns."
- 6) Otherwise, give up and ask for re-phrasing.

These criteria are simple, yet they are sufficient to enable the program to make quite reasonable decisions about the intended purpose in various sentences of the ambiguous word "has". Of course, the program can be fooled into making mistakes; *e.g.*, in case the sentence, "Dick has a chain", had been presented before the sentence "John owns a chain", in the above dialogue. However, a human being exposed to a new word in a similar situation would make a similar error. The point here is that it is feasible to automatically resolve ambiguities in sentence meaning by referring to the descriptions of the words in the sentence—descriptions which can automatically be created through proper prior exposure to unambiguous sentences.

- c) Streamlining linkages: All question-answering (model-searching) functions which involve references to set-inclusion or set-membership relations must "know" about the basic properties of those relations; *i.e.*, those functions must have built into them the ability to apply theorems like

$$x \subset y \wedge y \subset z = x \subset z \quad \text{and} \\ \alpha \in x \wedge x \subset y = \alpha \in y;$$

otherwise, the functions would not be able to make full use of the usually limited information available in the form of explicit links. On the other hand, since the functions involved will be "aware" of these theorems, then the set of questions which can be answered is independent of the presence or absence of explicit links

which provide the information to the right of the "=", provided the information to the left of the "=" is available. The "STREAMLINE" operation starts with the object  $x$  which is its argument, and considers all objects linked to  $x$ , directly or indirectly, through set-inclusion or set-membership. All explicit links among these objects which can also be deduced by use of the above-known theorems are deleted. A response of the form "(I FORGET THE SET-INCLUSION RELATION BETWEEN  $y$  and  $z$ )" indicates that whatever links were created by some sentence of a form similar to "(EVERY  $z$  IS A  $y$ )" are being deleted, and the space they occupied is being made available for other use.

## VI. EXTENSIONS OF SIR

Four major obstacles prevent the immediate expansion of a SIR-like system into a large, practical question-answering system: input language, memory size, search time, and subprogram interaction.

SIR's input language consists of sentences which match a number of simple formats. For convenient use, a more general system should accept a larger subset of natural English. Various research groups are studying this difficult problem of translating from English into a formal language.<sup>5,17,18</sup> The research effort described here concentrates, instead, on the representation and retrieval problems which will remain after the input language has been formalized.

The immediate reason for terminating the current project was the lack of computer memory for additional programs and data. However, at that point problems involving search time and subprogram interaction had also become significant. Progress in solving these latter problems is necessary so that we may be ready to use profitably the larger, better organized memories which will undoubtedly become available before long.

In SIR, the time required to search for the existence of particular relational links in the model increases rapidly with both the number of individual elements which can be linked and

the number of different relations which can do the linking. Of course, this exponential growth of tree structured search space is a familiar phenomenon in theorem proving, game playing, and other problem solving areas. General tree searching heuristics such as partitioning a tree into non-intersecting branches, searching for a path simultaneously from two endpoints toward an unknown common point, and generating intermediate "stepping stones" to use in a long search, must be further developed for application to the SIR model.

The most basic obstacle to enlarging SIR is the problem of subprogram interaction. The present system contains a separate subprogram for performing each different information retrieval task. This diffuse program structure was convenient during the early development of the system because it facilitated modifying component programs to experiment with different model structures and different search and deduction schemes. However, each new relation added to such a system may affect the question answering procedures associated with those relations already included. Since these procedures are buried in various subprograms, the addition of the subprograms associated with the new relation frequently necessitates program changes throughout the system. Because of these "interactions," the resulting program becomes more awkward and more difficult to generalize as its size increases. Future versions of semantic question answering systems must avoid this increasingly complex program organization. One solution might be based on a proposed "formalized" question answerer, outlined below and described in more detail in reference 14, which increases the question answering power as well as simplifies the program structure of the system.

### A FORMALIZED QUESTION ANSWERER

The formalized question answerer consists of the following components:

1. a *formal system* whose sentences correspond in a well defined way to "yes or no" questions; and a theorem-proving program which can determine whether well-formed sentences are "true," *i.e.*, whether the corresponding questions should be

answered "yes," on the basis of information in the model.

2. a *model* similar to the SIR model but containing, in addition to links relating objects, axioms and deduction rules of the formal system for the use of the theorem proving program.
3. a *programming language* for specifying question-answering procedures which are more complex than truth-testing.

The formal system has the same structure as the first order predicate calculus except that the domain of all quantifiers is the set of objects described in the model. Since this set is finite, the system is logically equivalent to the propositional calculus. The basic predicates of the formal system can be initially just those which are needed, along with quantifiers and sentential connectives, to express any question which the SIR system is capable of answering. It can be shown<sup>14</sup> that only a few basic predicates are needed to enable the formal system to express a great many questions. Also many interactions between predicates, which created programming difficulties in SIR, are automatically represented by the structure of the sentences in the formal system.

The formal system is decidable; therefore a program could be written which would answer any well-defined question on the basis of axioms and facts in the model. However, since the model might be large, such a program might be quite slow. On the other hand, a heuristic program could be written which would attempt to answer questions only on the basis of the most *relevant* data, where "most relevant" is defined in terms of the structure of the model. Such a program could conceivably be as efficient as the special purpose question answering subprograms of SIR.

The model in the formalized question answerer is the same as the SIR model except for containing an additional class of data. The described objects in this model can be names of real objects or classes of objects, or *names of basic predicates* in the formal system. The property list of a predicate contains all axioms or special deductive procedures associated with that predicate. The theorem proving program would have to perform in accordance with this

data in the model. Thus the user could "tell" the system how to change its question answering procedures, whenever such changes were desired, simply by changing the content of the model.

A theorem proving program can only answer "yes or no" questions. However, some of the questions which SIR can answer require the system to perform more elaborate information retrieval tasks. The power of a general purpose symbol manipulation computer language such as LISP<sup>15</sup> must be available for specifying these computational procedures since new question types may require, in the answering process, unanticipated kinds of data manipulation. However, these procedures should be made as easy to write and to understand as possible. In particular, the full power of the theorem proving program should be available within the procedure specification language.

For example, suppose the theorem prover is represented by the LISP function "theorem [x]," whose value is TRUE if  $x$  is a sentence in the formal system corresponding to a question whose answer is "yes," and FALSE otherwise. Then suppose that in the course of the procedure for answering the question, "what is the relative position of  $x$ ?" it is determined that  $y$  is to the right of  $x$  and also that a  $z$  is to the right of  $x$ . The procedure could then contain the statement, *if*

$$\text{theorem}[(\alpha)[\alpha \in Z \wedge \text{right}[\alpha; x] \wedge \text{right}[y; \alpha]]]$$

*then go[A] else go[B]* where A and B are appropriate further instructions in the procedure. The procedure writer need not consider how to answer the question, "is a  $z$  between  $x$  and  $y$ ?" for the *theorem* function, i.e., the theorem proving program, will do that for him.

Space does not permit a more detailed description of the proposed formalized question answerer. However, it should now be clear that such a system has all the question answering ability of SIR and accepts a much larger class of questions. More important, new relations can be added to the formalized system and the axioms of its proof procedure can be modified without any significant reprogramming, thus overcoming the "subprogram interaction" obstacle to the expansion of SIR.

## VII. CONCLUSIONS

### THE MODEL

The power of SIR's question answering subprograms is due primarily to the flexible property-list structure of the *model*.

SIR is not unique in permitting facts to be automatically added to or excised from the system. Several existing computer systems, *e.g.*, airline reservation systems, permit dynamic fact storage and retrieval. However, the existing systems generally depend upon the use of a fixed, unique representation for the information involved. A response generally is determined by the explicit presence or absence of a particular item of data.

In SIR, on the other hand, although the model is general enough to contain a wide class of data, it is organized so that subprograms may search it for *any* facts from which an appropriate answer may be deduced. Because of the way the deduction subprograms use the model, a fact may be represented in many different equally effective ways. *E.g.*, the system "knows" that the statement, "a finger is part of John" is true if (a) there is an explicit part-whole link from FINGER to JOHN; or if (b) there are links by means of which the retrieval programs can deduce that a finger is part of a person, and John is a person; or if (c) there are links by means of which the retrieval programs can deduce that a finger is part of a hand, and a hand is part of John; *etc.* Thus the use of this model facilitates question answering even in the absence of complete, explicit data. In addition, the system can automatically translate from one representation to another having some advantages. *E.g.*, the "streamline" operation described in Section V reduces storage space requirements by removing redundancy in the representation, without necessitating any changes in the operation of the system.

### VALUE OF PROGRAMMING

Many of the results and conclusions written after the development of a large computer program such as SIR frequently appear as if they could have been established without the tedious effort of programming. This is rarely true, and in fact, new systems which are described as complete "except for the programming" usually

require fundamental modifications if and when they are translated into operating programs. The reasons for the importance of actually writing the program include the following:

- (a) Without a program it is difficult to tell whether the specifications for a system are really complete and consistent. The process of building an operating system makes one aware of major problems which might otherwise remain unnoticed.
- (b) Programming not only turns up fallacies in the specifications for a system, but also usually suggests ways for avoiding them and improving the system. A completed "debugged" programmed system usually turns out to be a compromise between the system as it was originally specified, a simpler system which was more feasible to actually construct, and a more elaborate system whose new features were thought of during programming. This resulting system is frequently as useful as and certainly more reliable than the originally specified system, and in addition it may suggest the design of even more advanced systems. With SIR, for example, methods for implementing the "exception principle" and the resolution of ambiguities arose from the design of the basic question answerer, and the specifications for the formalized system of Section VI were based largely on properties of the final, working SIR system.
- (c) The programming process frequently turns up insights which might not otherwise be discovered.
- (d) Finally, the resulting program provides at the same time a demonstration of the feasibility of the ideas upon which it is based, a measure of the practicality of the system in terms of time and space requirements, and an experimental device for testing variations in the original specifications.

### NEXT STEPS

The present SIR system, and its formalized version discussed in Section VI, are proposed as first steps toward a true "understanding" machine. Further steps will involve developing

better means by which a computer system can add to its store of "knowledge" and can "intelligently" select relevant data from that store to use in particular problem solving tasks. These goals are embodied in the "advice taker" problem,<sup>19</sup> which is that of designing a machine whose operation is controlled by "advising" it, in a suitable English-like language, of desired procedures or results.

One type of "advice taker" is a program which can do any of a particular class of problems, such as writing other computer programs, in accordance with simple instructions. Simon<sup>20</sup> is working on such a program-writing program which accepts a broad range of descriptive English sentences as its input.

SIR represents a different approach. Instead of developing various special purpose advice takers, we attempt to build a single, general program which can do any task provided that program is properly controlled by information in its model. "Giving advice" then requires only the process of inserting control information into the model. In a sense, this program is simply an interpreter of information provided in the easily changeable model.

The SIR model provides its programs with information about the truth of particular relations between specific objects. The model in the formalized system of Section VI also provides the "theorem proving program" with axioms which describe properties of relations and interactions between relations. A next generalization would involve adding to the model information which specifies and controls theorem proving and model searching procedures for the program.

Ultimately the "intelligent" machine will have to be able to abstract from the information in its model, "realize" the necessity for additional action, and create the necessary instructions for itself. The design of such an "artificial intelligence" awaits the development of automatic concept formation and inductive inference systems as well as the generalizations of SIR described above.

#### REFERENCES

1. GREEN, B. F., Jr., et al., "Baseball: an automatic question answerer," in reference 21.
2. PHILLIPS, A. V., "A question-answering routine," Master's Thesis, Mathematics Department, MIT, Cambridge, Mass., 1960.
3. SIMMONS, R. F., et al., "Toward the synthesis of human language behavior," SP-466, Systems Development Corp., Santa Monica, California.
4. LINDSAY, R. K., "Inferential memory as the basis of machines which understand natural language," in reference 21.
5. DARLINGTON, J., "Translating ordinary language into symbolic logic," Memo MAC-M-149, Project MAC, MIT, Cambridge, Mass., 1964.
6. BENNETT, J. L., "A Computer program for word relations," Memo 1961-1, Mechanical Translation Group, MIT, Cambridge, Mass., 1961.
7. BOBROW, D. G., "Natural language input for a computer problem solving system," Ph.D. Thesis, Mathematics Department, MIT, Cambridge, Mass., 1964.
8. CARNAP, R., *Meaning and Necessity* (U. of Chi. Press, Chicago, Ill., 1947).
9. QUILLIAN, R., "A revised design for an understanding machine," Mechanical Translation, Vol. 7, No. 1 (1962).
10. QUINE, W., *Word and Object* (MIT Press, Cambridge, Mass., 1960).
11. REICHENBACH, H., *Elements of Symbolic Logic* (The Macmillan Co., New York, 1947).
12. SOMMERS, F. T., "Semantic structures and automatic clarification of linguistic ambiguity," International Electric Corp., Paramus, N. J., 1961.
13. WALPOLE, H. R., *Semantics: The Nature of Words and Their Meanings* (W. W. Norton and Co., New York, 1941).
14. RAPHAEL, B., "SIR: a computer program for semantic information retrieval," Ph.D. Thesis, Mathematics Department, MIT, Cambridge, Mass., 1964.
15. MCCARTHY, J., et al., *LISP 1.5 Programmers Manual* (MIT Press, Cambridge, Mass., 1963).



16. BOBROW, D. G., and B. RAPHAEL, "A comparison of list-processing computer languages," *Comm. ACM*, Vol. 7, No. 4 (1964).
17. COHEN, D., "Picture processing in a picture language machine," Report 7885, National Bureau of Standards, Washington, D. C., 1962.
18. ACF Industries, Avion Div., "Translating from ordinary discourse into formal logic—a preliminary study," Scientific Report AF CRC-TN-56-770.
19. MCCARTHY, J., "Programs with common sense," in *Proc. Symposium on Mechanization of Thought Processes*, National Physics Laboratory, Teddington, England, Her Majesty's Stationery Office, London, 1959.
20. SIMON, H. A., "Experiments with a heuristic compiler," Paper P-2349, RAND Corp., Santa Monica, California, 1961.
21. FEIGENBAUM, E. A., and J. FELDMAN, *Computers and Thought* (McGraw-Hill, New York, 1963).



# A QUESTION-ANSWERING SYSTEM FOR HIGH SCHOOL ALGEBRA WORD PROBLEMS\*

Daniel G. Bobrow

Massachusetts Institute of Technology, Cambridge, Massachusetts

## I. INTRODUCTION

The aim of the research reported here was to discover methods for building computer programs which can understand and communicate with people in a non-trivial subset of English. A computer program *understands* a subset of English if it accepts input sentences which are members of this subset, and answers questions based on information contained in the input. We describe in this paper a semantic theory of discourse, and utilize a first approximation to the analytical portion of this theory in the STUDENT question-answering system, a program which understands a subset of English in the sense defined above.

The STUDENT system, programmed in METEOR<sup>1</sup> and LISP<sup>2</sup>, accepts as input high school algebra word problems expressed within a restricted but comfortable subset of English. For example, STUDENT will accept the following problem statement:

“The price of a radio is \$69.70. If this price is 15 percent less than the marked price, find the marked price.”

After some computation STUDENT will respond:

“The marked price is 82 dollars.”

If needed, STUDENT has access to a store of “global” information not specific to any one problem, and can retrieve relevant facts and equations from this store of information. For example, when solving the problem:

“If 1 span equals 9 inches, and 1 fathom equals 6 feet, how many spans equals 1 fathom?”

STUDENT retrieves and uses the fact that 1 foot equals 12 inches, and prints the answer:

“1 fathom is 8 spans.”

STUDENT is embedded in the M.I.T. Project MAC time-sharing system<sup>3</sup>. Therefore, as a last resort, when it can not solve a problem, STUDENT requests and can obtain immediate help from the questioner.

A number of other English language question-answering systems have been constructed; the most closely related work was that of Green<sup>4</sup>, Lindsay<sup>5</sup>, and Raphael<sup>6</sup>. A critical analysis of this related work and criteria for evaluating a question-answering system may be found in the author's thesis<sup>\*</sup>. Simmons<sup>7</sup> gives a descriptive survey of systems which answer English questions.

\* The work reported here was supported in part by the M.I.T. Computation Center, in part by the M.I.T. Research Laboratory of Electronics, and in part by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under ONR Contract Number Nonr-

\* We shall not mention it again, but the reader who wants more extensive background material or a more detailed exposition of ideas given throughout this paper should refer to the thesis.

There are a number of reasons why I chose the context of algebra story problems in which to embed an English language question-answering system. First, we know a good type of data structure in which to store information needed to answer questions in this context, namely, algebraic equations. There exist well known algorithms for deducing information implicit in the equations, that is, values for particular variables which satisfy the set of equations.

In addition, I felt that there was a manageable subset of English in which many types of algebra story problems were expressible. A large number of these story problems are available in first year high school text books, and I have transcribed some of them into STUDENT's input English. Since this question-answering task is one performed by humans, and since the entire process from input to solution of the equations was programmed, we can obtain a measure of comparison between the performance of STUDENT and of a human on the same problems. In fact, this program on an IBM 7094, answers most questions that it can handle as fast as or faster than humans trying the same problem. In judging this comparison, one should remember the base speed of the IBM 7094, which can perform over one hundred thousand additions per second.

## II. SEMANTIC GENERATION AND ANALYSIS OF DISCOURSE

The purpose of this section is to put the techniques of analysis embedded in the STUDENT program into a wider context, and indicate how they would fit into a more general language processing system. We will describe a theory of semantic generation and analysis of discourse. STUDENT can then be considered a first approximation to a computer implementation of the analytic portion of the theory, with certain restrictions on the interpretation of a discourse to be analyzed. It will be evident from the theory why analysis is so greatly simplified by the imposed restrictions.

### A. *Language as Communication*

Language is an encoding used for communication between a speaker and a listener (or writer and reader). To transmit an "idea", the

speaker must first encode it in a message, as a string in the transmission language. In order to understand this message, a listener must decode it and extract its meaning. The coding of a particular message,  $M$ , is a function of both its global context and local context. The global context of a message is the background knowledge of the speaker and the listener, including some knowledge of possible universes of discourse, and codings for some simple ideas.

The local context of a message,  $M$ , is the set of messages temporally adjacent to  $M$ .  $M$  may refer back to earlier messages.  $M$  may even be just a modification of a previous message, and only understandable in this context. For example, consider the second sentence of the following discourse: "How many chaplains are in the U.S. Army? How many are in the Navy?"

In order for communication to take place, the information map of both the listener and the speaker must be approximately the same, at least for the universe of discourse; also the decoding process of the listener must be an approximate inverse of the encoding process of the speaker. Education in language is, in large part, an attempt to force the language processors of different people into a uniform mold to facilitate successful communication. We are not proposing that identity in detail is achieved, but as Quine<sup>9</sup> put it:

"Different persons growing up in the same language are like different bushes trimmed and trained to take the shape of identical elephants. The anatomical details of twigs and branches will fulfill the elephantine form differently from bush to bush, but the overall outward results are alike."

As a speaker transmits successive messages concerning some portion of his information map, the listener who understands the messages constructs a model of a "situation". The relation between the listener's model and the speaker's information map is that from each can be extracted the transmitted information relevant to the universe of discourse, including information deducible from the entire set of messages. The internal structure of the listener's model need bear no resemblance to that of the speaker, and may in general contain far less detail.

### B. Definition of Coherent Discourse

The theory of language generation and analysis which we shall describe below is designed to handle what we call *coherent discourse*. A discourse is a sequence of sentences, and the meaning of a discourse for a listener is the model of a situation he derives from this discourse. Determination of the meaning of each sentence of the discourse may sometimes involve knowledge of the meanings of other sentences of the discourse. A discourse is coherent if it has a complete and consistent interpretation within the model of the situation being built by the listener. A listener understands the discourse if his model of the situation is isomorphic to the speaker's model.

A listener's ability to build a model of a situation from a discourse is dependent on information available to him from his general store of knowledge. Therefore it is quite possible for a discourse to seem coherent to one listener and not another. A writer, reading his own writing, may feel that he has generated a coherent sequence of sentences, which, in fact, is incoherent to all other readers. This is, unfortunately, not a rare occurrence in the scientific literature. Conversely, a listener who is a psychiatrist, for example, may find coherence in a sequence of remarks which a patient thinks are entirely unrelated.

The STUDENT system utilizes an expandable store of general knowledge to build a model of a situation described in a member of a limited class of discourses. The form of the model of a situation built by STUDENT will be discussed in detail below.

### C. The Use of Kernel Sentences in Our Theory

A basic postulate of our theory of language analysis is that a listener understands a discourse by transforming it into an equivalent (in meaning) sequence of simpler kernel sentences. A *kernel sentence* is one which the listener can understand directly; that is, one for which he knows a transformation into his information store. Conversely, a speaker generates a set of kernel sentences from his information map, and utilizes a sequence of transformations on this set to yield his spoken discourse. This set of kernel sentences is not invariant from person

to person, and even varies for a single individual as he learns.

Although we are not proposing our theory as a basis for a psychological model, it has been useful, to avoid circumlocutions, to describe the theory in terms of the properties and actions of a hypothetical speaker and listener. All statements about speakers and listeners should be interpreted as referring to computer programs which respectively, generate and analyze coherent discourse.

### D. Generation of Coherent Discourse

1) *The Speaker's Model of the World*. We assume that a speaker has some model of the world in his information store. We shall not be concerned here with how this model was built, or its exact form. Different forms for the model will be useful for different language tasks, but they must all have the properties described below.

The basic components of the model are a set of objects,  $\{O_i\}$ , a set of functions  $\{F_i^n\}$ , a set of relations  $\{R_i^n\}$ , a set of propositions  $\{P_i\}$ , and a set of semantic deductive rules. A *function*  $F_i^n$  is a mapping from ordered sets of  $n$  objects, called the arguments of  $F_i^n$ , into the  $\{O_i\}$ . The mapping may be multivalued and is defined only if the arguments satisfy a set of conditions associated with  $F_i^n$ . A condition is essentially membership in a class of objects, but is defined more precisely below. A relation  $R_i^n$  is a special type of object in the model, and consists of a label (a unique identifier), and an ordered set of  $n$  conditions, called the argument conditions for the relation. Functions of relations are again relations.

An *elementary proposition*  $P_i$  consists of a label associated with some relation,  $R_i^n$ , and an ordered set of  $n$  objects satisfying the argument conditions for this relation. One may think of these propositions as the beliefs of a speaker about what relationships between objects he has noticed are true in the world. *Complex propositions* are logical combinations (in the usual sense) of elementary propositions.

The *semantic deductive rules* give procedures for adding new propositions to the model based on the propositions now in the model. In addi-

tion to the ordinary rules of logic, these rules include axioms about the relationships of the relations in the model. The semantic deductive rules also include links to the senses of the speaker. For example, one such deductive rule for adding a proposition to the model might be (loosely speaking) "Look in the real world and see if it is true." These rules essentially determine how the model is to be expanded, and are the most complex part of a complete system. However, from our present point of view, we need only consider these rules as a black box which can extend the set of propositions in the model.

A *closed question* is a relational label for some  $R_i^n$  and an ordered set of  $n$  objects. The *answer* to this question is affirmative if the proposition, consisting of this label and the  $n$  objects, is in the model (or can be added to it according to the semantic deductive rules). If the negation of this proposition is in the model (or can be added), the answer is negative. Otherwise the answer is undefined.

An *open question* consists of a relational label for an  $n$ -argument relation,  $R_i^n$ , and a set of objects corresponding to  $n-k$  of these arguments, where  $n \geq k \geq 1$ . An answer to an open question is an ordered set of  $k$  objects, such that if these objects are associated with the  $k$  unspecified arguments of  $R_i^n$ , the resulting proposition is in the model, or can be added to it. An open question may have no answers, or may have one or more answers. A *condition* is an open question with  $k=1$ , and an object satisfies a condition if it is an answer to the question.

2) *Generation of Kernel Sentences.* We have described the logical properties of the speaker's model of the world. We shall now consider how strings in a language, words, phrases, and sentences, are associated with the model. Corresponding to the set of objects  $\{O_i\}$  there is a set  $\{N_{ij}\}$  of strings (in English in our case), called the *names* of the objects. There is a many-one mapping from  $\{N_{ij}\}$  onto  $\{O_i\}$ . It is many-one because one object may have more than one name, e.g. frankfurter and hot dog both map back into the same object in the model.

Recall that functions map  $n$ -tuples of objects into objects. Thus a function name and an  $n$ -

tuple can specify an object. We can derive a name for this object from the function name and the names of its  $n$  arguments. Associated with each function is at least one linguistic form, a string of words with blanks in which names of arguments of the function must be inserted. Examples of linguistic forms associated with a model are "number of \_\_\_\_\_", "father of \_\_\_\_\_", and "the child of \_\_\_\_\_ and \_\_\_\_\_". There is a many-one mapping from the set of linguistic forms  $\{L_{ij}^n\}$  onto the set of functions. Two examples of multiple linguistic forms for the same function are: "father of \_\_\_\_\_" and "\_\_\_\_\_’s father"; and "\_\_\_\_\_ plus \_\_\_\_\_" and "the sum of \_\_\_\_\_ and \_\_\_\_\_". Thus, if objects  $x$  and  $y$  have names "the first number" and "the second number" and associated with the function "\*" is the linguistic form "the product of \_\_\_\_\_ and \_\_\_\_\_", then the name of the object produced by applying the function "\*" to  $x$  and  $y$  is "the product of the first number and the second number". A parsing of a name must decompose it into the part which is the linguistic form, and the parts which are names of arguments of the corresponding function. We shall call objects defined in terms of a function and an  $n$ -tuple of objects a *functionally defined object*, and those which are not functionally defined we shall call *simple objects*. Simple objects have *simple names* and functionally defined objects have *composite names*.

In addition to linguistic forms associated with functions, there are linguistic forms associated with relations. For an  $n$  argument relation there are  $n$  blanks in the linguistic form. Examples of relational linguistic forms are: "\_\_\_\_\_ equals \_\_\_\_\_", "\_\_\_\_\_ gave \_\_\_\_\_ to \_\_\_\_\_" and "\_\_\_\_\_ speaks". These linguistic forms, corresponding to the relations in the model, serve as frames for the kernel sentences.

In a manner similar to the way composite names are built, a kernel sentence corresponding to an elementary proposition is constructed by inserting names corresponding to each argument in the appropriate blank. Names may be simple or composite. An example of a kernel sentence for a proposition built from such a relational linguistic form is "John's father gave .3 times the salary of Bill to Jack." which con-

tains the simple names "John", ".3", "bill", and "Jack". It contains the functional linguistic forms "\_\_\_\_\_'s father", "\_\_\_\_ times \_\_\_\_" and "salary of \_\_\_\_" and the relational linguistic form "\_\_\_\_ gave to \_\_\_\_".

A kernel sentence corresponding to a complex proposition is constructed recursively from the kernel sentences corresponding to its elementary propositional constituents by placing them in the corresponding places in the linguistic forms "\_\_\_\_ and \_\_\_\_", "\_\_\_\_ or \_\_\_\_", "not \_\_\_\_" etc.

The kernel sentence corresponding to a closed question is constructed from the kernel of the corresponding proposition by placing it in the linguistic form "It is true that \_\_\_\_?" For an open question, dummy objects are placed in the open argument positions to complete a propositional form. These dummy arguments have names "who", "what", "where", etc., and which dummy objects are used depends on the condition on that argument position. A question mark is placed at the end of the kernel sentence constructed in the usual way from the relational linguistic form and the names of the arguments.

In generating a coherent discourse, a speaker chooses a number of propositions in his model and/or some open or closed questions. He then uses linguistic information associated with the model to construct the set of kernel sentences corresponding to this set of chosen propositions. In the next section we will discuss how he generates his discourse from this set of kernels.

3) *Transformations on Kernel Sentences.* The set of kernel sentences is the base of the coherent discourse. The meaning of a kernel sentence is the proposition into which it maps, and similarly, the meaning of any name is the object which is its image under the mapping. To this set of kernels we apply a sequence of meaning preserving transformations to get the final discourse. We use the word "transformation" in its broad general sense, not in the narrow technical sense defined by Chomsky.<sup>10</sup>

There two distinct types of transformations, structural and definitional. A structural or syntactic transformation is only dependent on the structure of the kernel string(s) on which it

operates. For example, one syntactic transformation takes a kernel in the active voice to one in the passive voice. Another combines two sentences into a single complex coordinate sentence.

One large class of syntactic transformations is used to substitute pronominal phrases for names. Pronominal phrases may be ordinary pronouns such as "he", "she", or "it". They may be referential phrases such as "the latter", "the former" or "this quantity". They may also be truncations of a full name such as "the distance" for "the distance between New York and Los Angeles". In cases where such pronominal reference is made, the coherence of the final discourse is dependent on the order in which the resultant strings appear.

The second type of transformation is definitional. It involves substitutions of linguistic strings and forms for ones appearing in the kernel sentences. For example, for any appearance of "2 times" we may substitute "twice", and for ".5 times" substitute "one half of". In addition to this string substitution, some transformations perform form substitution and rearrangement. For example, for a kernel sentence of the form " $x$  is  $y$  more than  $z$ ", where  $x$ ,  $y$ , and  $z$  are any names, one definitional transformation can substitute " $x$  exceeds  $z$  by  $y$ ".

Some transformations are optional, and some may be mandatory if certain forms are present in the kernel set. Certain transformations are used by a speaker for stylistic purposes, for example, to emphasize certain objects; other syntactic transformations, such as those which perform pronominal substitutions, are used because they decrease the depth of a construction, in the sense defined by Yngve.<sup>11</sup>

Let us review the steps in the generation of a coherent discourse. The speaker chooses a set of propositions, the "ideas" he wishes to transmit. He then encodes them as language strings called kernel sentences in the manner described above. He then chooses a sequence of structural and definitional transformations which are defined on this set of kernels or on the ordered set of sentences which result from applications of the first transformations. The resulting sequence of sentences will be a co-

herent discourse to a listener if he knows all the definitional transformations applied. In addition, every pair of distinct names which the speaker maps back into the same object, the listener must also map into a single object.

#### E. *Analysis of Coherent Discourse*

Generation of coherent discourse consists of two distinguishable steps. From propositions in the speaker's model of the world, he generates an ordered set of kernel sentences. He then applies a sequence of transformations to this kernel set. The resulting discourse is a coded message which is to be analyzed and decoded by a listener. The listener's problem can be loosely characterized as an attempt to answer the question, "What would I have meant if I said that?"

To analyze a discourse the listener must find the set of kernel sentences from which it was generated; one way to do this is to find a set of inverse transformations which when applied to the input discourse yield a sequence of kernel sentences. The listener must then transform these kernel sentences to an appropriate representation in his information store. The appropriateness of a representation is a function of what later use the listener expects to make of the information contained in the discourse. The listener may simultaneously transform a given kernel sentence into a number of different representations in his information store. On a level of pragmatic analysis, statements require only storage of information. Questions and imperatives require appropriate responses from the listener. The difficulties in analysis dichotomize into those associated with finding the kernel sentences which are the base of the discourse, and those associated with transforming the kernel sentences into representations in the information store.

STUDENT'S analytical program utilizes a set of inverse analytic transformations. If  $T_i$  is a transformation that may be used in generating a discourse, and  $T_i(S) = S$ , where  $S$  and  $S$  are sets of sentences, then the analytic transformation  $T_i^{-1}$  is the inverse of  $T_i$  if and only if  $T_i^{-1}(S) = S$ . The choice of which inverse transformations to apply and the order of their application may be restricted by utiliz-

ing heuristics concerned with features of the input.

Once the base set of kernel sentences for a given discourse is determined, there remains the problem of entering representations of these sentences in the listener's information store. The major problem in accomplishing this step involves the separation of those words which are part of linguistic forms for relations, and those which are part of a name. This is difficult because the same word (lexicographic symbol) may have multiple uses in a language. Having separated the relational form from the names which represent the arguments of this relation, one can then analyze the name in terms of components which are functional linguistic forms and others which are simple names. From this parsing in terms of relational linguistic forms, functional linguistic forms and simple names, the discourse can be transformed into a canonical representation in the information store of the listener.

#### F. *Limited Deductive Models*

A complete understanding of a discourse by a listener would imply that the representation of the discourse in his information store is essentially isomorphic to the speaker's model of the world, at least for the universe of discourse. The listener's representation must preserve all information implicit in the discourse.

If the listener is only interested in certain aspects of the discourse, he need only preserve information relevant to his interest, and discard the rest. Within his area of interest the listener's model is isomorphic to the speaker's model in the sense that all relevant deductions which can be made by the speaker on the basis of the discourse can also be made by the listener. Outside this area of interest, the listener will be unable to answer any questions. We call such restricted information stores *limited deductive models*.

The question-answering programs of Lindsay and Raphael, and the STUDENT system, all utilize limited deductive models. For the area of interest in each of these programs there is a "natural" representation for the information in the allowable input. These representations are natural in that they facilitate the deduction of implicit information. For example, Lindsay's



family tree representation makes it easy to compute the relationship of any two individuals in the tree, independent of the number of sentences necessary to build the tree.

Because the number of relations and functions expressible in the models in all three systems is very limited, there is a corresponding limitation on the number of linguistic forms that may appear in the input. This greatly simplifies the parsing problem by restricting alternatives for forms in the input text.

### G. The STUDENT Deductive Model

The STUDENT system is an implementation of the analytic portion of our theory. STUDENT performs certain inverse transformations to obtain a set of kernel sentences and then transforms these kernel sentences to expressions in a limited deductive model. Utilizing the power of this deductive model, within its limited domain of understanding, it is able to answer questions based on information implicit in the input information.

The analytic and transformational techniques utilized in STUDENT are described in detail in the next section. We shall describe here the canonical representation of objects, relations, and functions within the model. STUDENT is restricted to answering questions framed in the context of algebra story problems. Algebraic equations are a natural representation for information in the input.

The objects in the model are numbers, or numbers with an associated dimension. The only relation in the model is equality, and the only functions represented directly in the model are the arithmetic operations of addition, negation, multiplication, division and exponentiation. Other functions are defined in terms of these basic functions, by composition, and/or substitution of constants for arguments of these functions. For example, the operation of squaring is defined as exponentiation with "2" as the second argument of the exponential function; subtraction is a composition of addition and negation.

Within the computer, a parenthesized prefix notation is used for a standard representation of the equations implicit in the English input. The arithmetic operation to be expressed is

made the first element of a list, and the arguments of the function are succeeding list elements. The exact notation is given in Figure 1. In the figure, A, B, and C are any representations of objects in the model, either composite or simple names. The usual infix notation for these functional expressions is given for comparison. Because this is a fully parenthesized notation, no ambiguity of operational order arises, as it does, for example, for the unparenthesized infix notation expression  $A*B+C$ , or its corresponding natural language expression "A times B plus C". Note also that in this prefix notation *plus* and *times* are not strictly binary operators. Indeed, in the model they may have any finite number of arguments, e.g. (TIMES A B C D) is a legitimate expression in the STUDENT model.

Representations of objects in the STUDENT deductive model are taken from the input. Any strings of words *not* containing a linguistic form associated with an arithmetic function expressible in the model are considered simple names for objects. Thus, "the age of the child of John and Jane" is considered a simple name because it contains no functional linguistic forms associated with functions represented in STUDENT's limited deductive model. In a more general model it would be considered a composite name, and the functional forms "age of \_\_\_\_\_" and "child of \_\_\_\_\_ and \_\_\_\_\_" would be mapped into their corresponding functions in the model.

Figure 1: Notation Within the STUDENT Deductive Model

Operation	Infix	
	Notation	Prefix Notation
Equality	$A = B$	(EQUAL A B)
Addition	$A + B$	(PLUS A B)
	$A + B + C$	(PLUS A B C)
Negation	$- A$	(MINUS A)
Subtraction	$A - B$	(PLUS A
		(MINUS B))
Multiplication	$A * B$	(TIMES A B)
	$A * B * C$	(TIMES A B C)
Division	$A / B$	(QUOTIENT A B)
Exponentiation	$A^B$	(EXPT A B)

Figure 1. Flow Chart of the Student Program

Because such complex strings are considered simple names in the model, and objects are distinguished only by their names, it is important to determine when two distinct names actually refer to the same object. In fact, answers to questions in the STUDENT system are statements of the identity of the object referenced by two names. However, one of the names (the desired one) must satisfy a certain lexical condition. Most often this condition is just that the name be a numeral. For a more general model this restriction could be stated as requiring a simple name corresponding to some functionally defined name—because, for example, “number of \_\_\_\_\_” would be a functional linguistic form in the general model, and the only simple name for such an object would be the numeral corresponding to this number. An answer consists of a statement of identity e.g. “The number of customers Tom gets is 162.”

The other lexical restriction on answers sometimes used in the STUDENT system is insistence that a certain unit (corresponding to a dimension associated with a number) appear in the desired answer. For example, *spans* is the unit specified by the question “How many spans equals 1 fathom?”, and the answer given by STUDENT is “1 fathom is 8 spans.”

The deductive model described here is useful for answering questions because we know how to extract implicit information from expressions in this model. More explicitly, we know how to solve sets of algebraic equations to find numerical values which satisfy these equations.

### III. Transformation of English to the STUDENT Deductive Model

Our question-answering system contains two main programs which process English input. One is a program called STUDENT which accepts the statement of an algebra story problem and attempts to find the solution to the particular problem. STUDENT does not store any information, nor “remember” any thing from problem to problem. The information obtained by STUDENT is the local context of the question.

The other program is called REMEMBER and it processes and stores facts not specific to any one problem. These facts make up STU-

DENT's store of “global information” as opposed to the “local information” contained in the statement of any one problem. This information is accepted in a subset of English which overlaps but is different from the subset of English accepted by STUDENT. REMEMBER accepts statements in certain fixed formats, and for each format the information is stored in a way that makes it convenient for retrieval and use within the STUDENT program. The following examples indicate some of the acceptable formats for global information:

“Distance equals speed times time.”

“Feet is the plural of foot.”

“Bill is a person.”

“Times is an operator of level 1.”

“One half always means .5.”

We will not give any details of the processing done by REMEMBER, except to note that some of the global information is stored by actually adding statements to the STUDENT program, and other information is stored on dictionary entries for the individual words.

#### A. Outline of the Operation of STUDENT

To provide perspective by which to view the detailed heuristic techniques used in the STUDENT program, we shall first give an outline of the operation of the STUDENT program when given a problem to solve. This outline is a verbal description of the flow chart of the program found in the appendix.

STUDENT is asked to solve a particular problem. We assume that all necessary global information has been stored previously. STUDENT will now transform the English input statement of this problem into expressions in its limited deductive model, and through appropriate deductive procedures attempt to find a solution. More specifically, STUDENT finds the kernel sentences of the input discourse, and transforms this sequence of kernels into a set of simultaneous equations, keeping a list of the answers required, a list of the units involved in the problem (e.g. dollars, pounds) and a list of all the variables (simple names) in the equations. Then STUDENT invokes the SOLVE program to solve this set of equations for the desired unknowns. If a solution is found, STUDENT prints the values of the un-

knowns requested in a fixed format, substituting in “(variable IS value)” the appropriate phrases for *variable* and *value*. If a solution cannot be found, various heuristics are used to identify two variables (i.e. find two slightly different phrases that refer to the same object in the model). If two variables, A and B, are identified, the equation  $A = B$  is added to the set of equations. In addition, the store of global information is searched to find any equations that may be useful in finding the solution to this problem. STUDENT prints out any assumptions it makes about the identity of two variables, and also any equations that it retrieves because it thinks they may be relevant. If the use of global equations or equations from identifications leads to a solution, the answers are printed out in the format described above.

If a solution is not found, and certain idioms are present in the problem (a result of a definitional transformation used in the generation of the problem), a substitution is made for each of these idioms in turn and the transformation and solution process is repeated. If the substitutions for these idioms do not enable the problem to be solved by STUDENT, then STUDENT requests additional information from the questioner, showing him the variables being used in the problem. If any information is given, STUDENT tries to solve the problem again. If none is given, it reports its inability to solve this problem and terminates. If the problem is ever solved, the solution is printed and the program terminates.

### B. Categories of Words in a Transformation.

The words and phrases (strings of words) in the English input can be classified into three distinct categories on the basis of how they are handled in the transformation to the deductive model. The first category consists of strings of words which name objects in the model; I call such strings, *variables*. Variables are identified only by the string of words in them, and if two strings differ at all, they define distinct variables. One important problem considered below is how to determine when two distinct variables refer to the same object.

The second category of words and phrases are what I call *substitutors*. Each substitutor may be replaced by another string. Some substitutions are mandatory; others are optional

and are only made if the problem cannot be solved without such substitutions. An example of a mandatory substitution is “2 times” for the word “twice”. “Twice” always means “2 times” in the context of the model, and therefore this substitution is mandatory. One optional “idiomatic” substitution is “twice the sum of the length and width of the rectangle” for “the perimeter of the rectangle”. The use of these substitutions in the transformation process is discussed below. These substitutions are inverses of definitional transformations as defined earlier.

Members of the third category of words indicate the presence of functional linguistic forms which represent functions in the deductive model. I call members of this third class *operators*. Operators may indicate operations which are complex combinations of the basic functions of the deductive model. One simple operator is the word “plus”, which indicates that the objects named by the two variables surrounding it are to be added. An example of a more complex operator is the phrase “percent less than”, as in “10 percent less than the marked price”, which indicates that the number immediately preceding the “percent” is to be subtracted from 100, this result divided by 100, and then this quotient multiplied by the variable following the “than”.

Operators may be classified according to where their arguments are found. A prefix operator, such as “the square of . . . .” precedes its argument. An operator like “. . . .percent” is a suffix operator, and follows its argument. Infix operators such as “. . . .plus. . . .” or “. . . .less than . . . .” appear between their two arguments. In a split prefix operator such as “difference between . . . . and . . . .”, part of the operator precedes, and part appears between the two arguments. “The sum of . . . .and. . . .and. . . .” is a split prefix operator with an indefinite number of arguments.

Some words may act as operators conditionally, depending on their context. For example, “of” is equivalent to “times” if there is a fraction immediately preceding it; e.g., “.5 of the profit” is equivalent to “.5 times the profit”; however, “Queen of England” does not imply

a multiplicative relationship between the Queen and her country.

### C. Transformational Procedures.

Let us now consider in detail the transformation procedure used by STUDENT, and see how the three categories of phrases interact. Let us consider the following example, which has been solved by STUDENT.

(THE PROBLEM TO BE SOLVED IS)  
 (IF THE NUMBER OF CUSTOMERS TOM GETS IS *TWICE* THE *SQUARE OF* 20 *PER CENT* OF THE NUMBER OF ADVERTISEMENTS HE RUNS, AND THE NUMBER OF ADVERTISEMENTS HE RUNS IS 45, WHAT IS THE NUMBER OF CUSTOMERS TOM GETS Q.)

Shown below are copies of actual printout from the STUDENT program, illustrating stages in the transformation and the solution of the problem. The parentheses are an artifact of the LISP programming language, and "Q." is a replacement for the question mark not available on the key punch.

The first stage in the transformation is to perform all mandatory substitutions. In this problem only the three phrases underlined (by the author, not the program) are substitutors: "twice" becomes "2 times", "per cent" becomes the single word "percent", and "square of" is truncated to "square". Having made these substitutions, STUDENT prints:

(WITH MANDATORY SUBSTITUTIONS  
 THE PROBLEM IS)  
 (IF THE NUMBER OF CUSTOMERS TOM GETS IS 2 *TIMES* THE *SQUARE* 20 *PERCENT* OF THE NUMBER OF ADVERTISEMENTS HE RUNS, AND THE NUMBER OF ADVERTISEMENTS HE RUNS IS 45, WHAT IS THE NUMBER OF CUSTOMERS TOM GETS Q.)

From dictionary entries for each word, the words in the problem are tagged by their function in terms of the transformation process, and STUDENT prints:

(WITH WORDS TAGGED BY FUNCTION  
 THE PROBLEM IS)  
 (IF THE NUMBER (OF / OP) CUSTOMERS (GETS / VERB) IS 2 (TIMES / OP 1) THE (SQUARE / OP 1) 20 (PERCENT

/ OP 2) (OF/OP) THE NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS, AND THE NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS IS 45, (WHAT / QWORD) IS THE NUMBER (OF / OP) CUSTOMERS TOM (GETS / VERB) (QMARK / DLM))

If a word has a tag, or tags, the word followed by "/", followed by the tags, becomes a single unit, and is enclosed in parentheses. Some typical taggings are shown above. "(OF/OP)" indicates that "OF" is an operator and other taggings show that "GETS" is a verb, "TIMES" is an operator of level 1 (operator levels will be explained below), "SQUARE" is an operator of level 1, "PERCENT" is an operator of level 2, "HE" is a pronoun, "WHAT" is a question word, and "QMARK" (replacing Q.) is a delimiter of a sentence. These tagged words will play the principal role in the remaining transformation to the set of equations implicit in this problem statement.

The next stage in the transformation is to break the input sentences into "kernel sentences". As in the example, a problem may be stated using sentences of great grammatical complexity; however, the final stage of the transformation is only defined on a set of kernel sentences. The simplification to kernel sentences as done in STUDENT depends on the recursive use of format matching. If an input sentence is of the form "IF" followed by a substring, followed by a comma, a question word and a second substring then the first substring (between the IF and the comma) is made an independent sentence, and everything following the comma is made into a second sentence. In the example, this means that the input is resolved into the following two sentences, (where tags are omitted for the sake of brevity):

"The number of customers Tom gets is 2 times the square 20 percent of the number of advertisements he runs, and the number of advertisements he runs is 45." and "What is the number of customers Tom gets?"

This last procedure effectively resolves a problem into declarative assumptions and a question sentence. A second complexity resolved by STUDENT is illustrated in the first

sentence of this pair. A coordinate sentence consisting of two sentences joined by a comma immediately followed by an "and" will be resolved into these two independent sentences. The first sentence is therefore resolved into two simpler sentences.

Using these two inverse syntactic transformations, this problem statement is resolved into "simple" kernel sentences. For the example, STUDENT prints

(THE SIMPLE SENTENCES ARE)  
 (THE NUMBER (OF / OP) CUSTOMERS  
 TOM (GETS / VERB) IS 2 (TIMES / OP  
 1) THE (SQUARE / OP 1) 20 (PERCENT  
 / OP 2) (OF / OP) THE NUMBER (OF  
 / OP) ADVERTISEMENTS (HE / PRO)  
 RUNS (PERIOD / DLM)  
 (THE NUMBER (OF / OP) ADVERTISE-  
 MENTS (HE / PRO) RUNS IS 45 (PE-  
 RIOD / DLM)  
 ((WHAT / QWORD) IS THE NUMBER  
 (OF / OP) CUSTOMERS TOM (GETS /  
 VERB) (QMARK / DLM))

Each simple sentence is a separate list, i.e., is enclosed in parentheses, and each ends with a delimiter (a period or question mark). Each of these sentences can now be transformed directly to its interpretation in the model.

#### D. From Kernel Sentences to Equations

The transformation from the simple kernel sentences to equations uses three levels of precedence for operators. Operators of higher precedence level are used earlier in the transformation. Before utilizing the operators, STUDENT looks for linguistic forms associated with the equality relation. These forms include the copula "is" and transitive verbs in certain contexts. In the example we are considering, only the copula "is" is used to indicate equality. The use of transitive verbs as indicators of equality, that is, as relational linguistic forms, will be discussed in connection with another example. When the relational linguistic form is identified, the names which are the arguments of the form are broken down into variables and operators (functional linguistic forms). In the present problem, the two names are those on either side of the "is" in each sentence.

The word "is" may also be used meaningfully within algebra story problems as an auxiliary verb (*not* meaning equality) in such verbal phrases as "is multiplied by" or "is divided by". A special check is made for the occurrence of these phrases before proceeding on to the main transformation procedure. The transformation of sentences containing these special verbal phrases will be discussed later. If "is" does not appear as an auxiliary in such a verbal phrase, a sentence of the form "P1 is P2" is interpreted as indicating the equality of the objects named by phrases P1 and P2. No equality relation will be recognized within these phrases, even if an appropriate transitive verb occurs within either of them. If P1\* and P2\* represent the arithmetic transformations of P1 and P2, then "P1 is P2" is transformed into the equation "(EQUAL P1\* P2\*)".

The transformation of P1 and P2 to give them an interpretation in the model is performed recursively using a program equivalent to the table in Figure 2. This table shows all the operators and formats currently recognized by the STUDENT program. New operators can easily be added to the program equivalent of this table.

In performing the transformation of a phrase P, a left to right search is made for an operator of level 2 (indicated by subscripts of "OP" and 2). If there is none, a left to right search is made for a level 1 operator (indicated by subscripts "OP" and 1), and finally another left to right search is made for an operator of level 0 (indicated by a subscript "OP" and no numerical subscript). The first operator found in this ordered search determines the first step in the transformation of the phrase. This operator and its context are transformed as indicated in column 4 in the table. If no operator is present, delimiters and articles (*a*, *an* and *the*) are deleted, and the phrase is treated as an indivisible entity, a variable.

In the example, the first simple sentence is  
 (THE NUMBER (OF/OP) CUSTOMERS  
 TOM (GETS/VERB) IS 2 (TIMES/OP  
 1) THE (SQUARE/OP 1) 20 (PER-  
 CENT/OP 2) (OF/OP) THE NUMBER  
 (OF/OP) ADVERTISEMENTS (HE/  
 PRO) RUNS (PERIOD/DLM))

Figure 2

Operator	Precedence Level	Context	Interpretation in the Model	
PLUS	2	P1 PLUS P2	(PLUS P1* P2*)	(a)
PLUSS	0	P1 PLUSS P2	(PLUS P1* P2*)	(b)
MINUS	2	P1 MINUS P2 MINUS P2	(PLUS P1* (MINUS P2*)) (MINUS P2*)	(c)
MINUSS	0	P1 MINUSS P2	(PLUS P1* (MINUS P2*))	(b)
TIMES	1	P1 TIMES P2	(TIMES P1* P2*)	
DIVBY	1	P1 DIVBY P2	(QUOTIENT P1* P2*)	
SQUARE	1	SQUARE P1	(EXPT P1* 2)	(d)
SQUARED	0	P1 SQUARED	(EXPT P1* 2)	
**	0	P1 ** P2	(EXPT P1* P2*)	
LESSTHAN	2	P1 LESSTHAN P2	(PLUS P2* (MINUS P1*))	
PER	0	P1 PER K P2 P1 PER P2	(QUOTIENT P1* (K P2))* (QUOTIENT P1* (1 P2))*	(e) (f)
PERCENT	2	P1 K PERCENT P2	(P1 (K/100) P2)*	(f) (g)
PERLESS	2	P1 K PERLESS P2	(P1 ((100-K)/100) P2)*	(f) (g)
SUM	0	SUM P1 AND P2 AND P3 SUM P1 AND P2	(PLUS P1* (SUM P2 AND P3))* (PLUS P1* P2*)	
DIFFERENCE	0	DIFFERENCE BETWEEN P1 AND P2	(PLUS P1* (MINUS P2*))	
OF	0	K OF P2 P1 OF P2	(TIMES K P2*) (P1 OF P2)*	

- (a) If P1 is a phrase, P1\* indicates its interpretation in the model.  
 (b) *PLUSS* and *MINUSS* are identical to PLUS AND MINUS except for precedence level.  
 (c) When two possible contexts are indicated, they are checked in the order shown.  
 (d) SQUARE P1 and SUM P1 are idiomatic shortenings of SQUARE OF P1 and SUM OF P1.  
 (e) \* outside a parenthesized expression indicates that the enclosed phrase is to be transformed.  
 (f) K is a number.  
 (g) / and - imply that the indicated arithmetic operations are actually performed.

This is of the form "P1 is P2", and is transformed to (EQUAL P1\* P2\*). P1 is "(THE NUMBER (OF/OP) CUSTOMERS TOM (GETS/VERB))". The occurrence of the verb "gets" is ignored because of the presence of the "is" in the sentence, meaning "equals". The only operator found is "(OF/OP)". From the table we see that if "OF" is immediately preceded by a number (*not* the word "number") it is treated as if it were the infix "TIMES". In this case, however, "OF" is not preceded by a number; the subscript OP, indicating that "OF" is an operator, is stripped away, and the transformation process is repeated on the phrase with "OF" no longer act-

ing as an operator. In this repetition, no operators are found, and P1\* is the variable

(NUMBER OF CUSTOMERS TOM (GETS/VERB)).

To the right of "IS" in the sentence is P2:  
 (2 (TIMES/OP 1) THE (SQUARE/OP 1)  
 20 (PERCENT/OP 2) (OF/OP) THE  
 NUMBER (OF/OP) ADVERTISE-  
 MENTS (HE/PRO) RUNS (PERIOD/  
 DLM)) The first operator found in P2 is PERCENT, an operator of level 2.

From the table in Figure 2, we see that this operator has the effect of dividing the number immediately preceding it by 100. The "PER-

CENT" is removed and the transformation is repeated on the remaining phrase. In the example, the "...20 (PERCENT/OP 2) (OF/OP)..." becomes "...2000(OFF/OP)..."

Continuing the transformation, the operators found are, in order, TIMES, SQUARE, OF and OF. Each is handled as indicated in the table. The "OF" in the context "...2000(OFF/OP) THE ..." is treated as an infix TIMES, while at the other occurrence of "OF", the operator marking is removed. The resulting transformed expression for P2 is:

```
(TIMES 2 (EXPT (TIMES .2 (NUMBER
OF ADVERTISEMENTS (HE/PRO)
RUNS)) 2))
```

The transformation of the second sentence of the example is done in a similar manner, and yields the equation:

```
(EQUAL (NUMBER OF ADVERTISE-
MENTS (HE/PRO) RUNS) 45)
```

The third sentence is of the form "What is P1?". It starts with a question word and is therefore treated specially. A unique variable, a single word consisting of an  $X$  of  $G$  followed by five integers, is created, and the equation (EQUAL  $X_{nnnnn}$  P1\*) is stored. For this example, the variable X00001 was created, and this last simple sentence is transformed to the equation:

```
(EQUAL X00001 (NUMBER OF CUSTOM-
ERS TOM (GETS/VERBS))
```

In addition, the created variable is placed on the list of variables for which STUDENT is to find a value. Also, this variable is stored, paired with P1, the untransformed right side, for use in printing out the answer. If a value is found for this variable, STUDENT prints the sentence (P1 is *value*) with the appropriate substitution for *value*. Below we show the full set of equations, and the printed solution given by STUDENT for the example being considered. For ease in solution, the last equations created are put first in the list of equations.

```
(THE EQUATIONS TO BE SOLVED ARE)
(EQUAL X00001 (NUMBER OF CUSTOM-
ERS TOM (GETS/VERB))) (EQUAL
(NUMBER OF ADVERTISEMENTS (HE/
PRO) RUNS) 45) (EQUAL (NUMBER OF
CUSTOMERS TOM (GETS/VERB))
```

```
(TIMES 2 (EXPT (TIMES .2000 (NUM-
BER OF ADVERTISEMENTS (HE/PRO)
RUNS)) 2))) (THE NUMBER OF CUS-
TOMERS TOM GETS IS 162)
```

In the example just shown, the equality relation was indicated by the copula "is". In the problem shown below, solved by STUDENT, equality is indicated by the occurrence of a transitive verb in the proper context.

```
(THE PROBLEM TO BE SOLVED IS)
(TOM HAS TWICE AS MANY FISH AS
MARY HAS GUPPIES. IF MARY HAS 3
GUPPIES, WHAT IS THE NUMBER OF
FISH TOM HAS Q.)
(THE NUMBER OF FISH TOM HAS IS 6)
```

The verb in this case is "has". The simple sentence "Mary has 3 guppies" is transformed to the "equivalent" sentence "The number of guppies Mary has is 3" and the sentence is processed as before. This transformation rule may be stated generally as: anything (a subject) followed by a verb followed by a number followed by anything (the unit) is transformed to a sentence starting with "THE NUMBER OF" followed by the unit, followed by the subject and the verb, followed by "IS" and then the number. In "Mary has 3 guppies" the subject is "Mary", the verb "has", and the units "guppies". Similarly, the sentence "The witches of Firth brew 3 magic potions" would be transformed to

```
"The number of magic potions the witches of
Firth brew is 3."
```

In addition to a declaration of number, a single-object transitive verb may be used in a comparative structure, such as exhibited in the sentence "Tom has twice as many fish as Mary has guppies." STUDENT transforms this sentence into the equivalent sentence.

```
"The number of fish Tom has is twice the
number of guppies Mary has."
```

Transformations of new sentence formats to formats previously "understood" by the program can be easily added to the program, thus extending the subset of English "understood" by STUDENT.

The word "is" indicates equality only if it is not used as an auxiliary. The example below shows how verbal phrases containing "is", such

as “is multiplied by”, and “is increased by” are handled in the transformation.

(THE PROBLEM TO BE SOLVED IS)  
(A NUMBER IS MULTIPLIED BY 6.  
THIS PRODUCT IS INCREASED BY 44.  
THIS RESULT IS 68. FIND THE NUM-  
BER.)

(THE EQUATIONS TO BE SOLVED ARE)  
(EQUAL X00001 (NUMBER))  
(EQUAL (PLUS (TIMES (NUMBER) 6)  
44) 68)

(THE NUMBER IS 4)

The sentence “A number is multiplied by 6” only indicates that two objects in the model are related multiplicatively, and does not indicate explicitly any equality relation. The interpretation of this sentence in the model is the prefix notation product “(TIMES (NUMBER 6))”. This latter phrase is stored in a temporary location for possible later reference. In this problem, it is referenced in the next sentence, with the phrase “this produce”. The important word in this last phrase is “this”—STUDENT ignores all other words in a variable containing the key word “this”. The last temporarily stored phrase is substituted for the phrase containing “this”. Thus, the first three sentences in the problem shown above yield only one equation, after two substitutions for “this” phrases. The last sentence “Find the number.” is transformed as if it were “What is the number Q.”, and yields the first equation shown.

The word “this” may occur in a context where it is not referring to a previously stored phrase. Below is an example with such a context.

(THE PROBLEM TO BE SOLVED IS)  
(THE PRICE OF A RADIO IS 69.70 DOL-  
LARS. IF THIS PRICE IS 15 PERCENT  
LESS THAN THE MARKED PRICE, FIND  
THE MARKED PRICE.)

(THE MARKED PRICE IS 82 DOLLARS)

In such contexts, the phrase containing “THIS” is replaced by the left half of the last equation created. In this example, STUDENT breaks the last sentence into two simple sentences, deleting the “IF”. Then the phrase “THIS PRICE” is replaced by the variable “PRICE OF RADIO”, which is the left half of the previous equation.

This problem illustrates two other features of the STUDENT program. The first is the action of the complex operator “percent less than”. It causes the number immediately preceding it, i.e., 15, to be subtracted from 100, this result divided by 100, to give .85. Then this operator becomes the infix operator “TIMES”. This is indicated in the table in Figure 2.

This problem also illustrates how units such as “dollars” are handled by the STUDENT program. Any word which immediately follows a number is labeled as a special type of variable called a *unit*. A number followed by a unit is treated in the equation as a product of the number and the unit, e.g., “69.70 DOLLARS” becomes “(TIMES 69.70 (DOLLARS))”. Units are treated as special variables in solving the set of equations; a unit may appear in the answer though variables cannot. If the value for a variable found by the solver is the product of a number and a unit, STUDENT concatenates the number and the unit. For example, the solution for “(MARKED PRICE)” in the problem above was (TIMES 82 (DOLLARS)) and STUDENT printed out:

(THE MARKED PRICE IS 82 DOLLARS)

There is an exception to the fact that any unit may appear in the answer, as illustrated in the problem below.

(THE PROBLEM TO BE SOLVED IS)  
(IF 1 SPAN EQUALS 9 INCHES, AND 1  
FATHOM EQUALS 6 FEET, HOW MANY  
SPANS EQUALS 1 FATHOM Q.)

(THE EQUATIONS TO BE SOLVED ARE)  
(EQUALS X00001 (TIMES 1 (FATH-  
OMS))) (EQUAL (TIMES 1 (FATH-  
OMS)) (TIMES 6 (FEET))) (EQUAL  
(TIMES 1 (SPANS)) (TIMES 9 (INCH-  
ES)))

THE EQUATIONS WERE INSUFFICIENT  
TO FIND A SOLUTION (USING THE  
FOLLOWING KNOWN RELATIONSHIPS)  
((EQUAL (TIMES 1 (YARDS)) (TIMES  
3 (FEET))) (EQUAL (TIMES 1 (FEET))  
(TIMES 12 (INCHES))))

(1 FATHOM IS 8 SPANS)

If the unit of the answer is specified, in this problem by the phrase “how many *spans*”—then *only* that unit, in this problem “spans”,



may appear in the answer. Without this restriction, STUDENT would blithely answer this problem with "(1 FATHOM IS 1 FATHOM)".

In the transformation from the English statement of the problem to the equations, "9 INCHES" became (TIMES 9 (INCHES)). However, "1 FATHOM" became "(TIMES 1 (FATHOMS))". The plural form for fathom has been used instead of the singular form. STUDENT always uses the plural form if known, to ensure that all units appear in only one form. Since "fathom" and "fathoms" are different, if both were used STUDENT would treat them as distinct, unrelated units. The plural form is part of the global information that can be made available to STUDENT, and the plural form of a word is substituted for any singular form appearing after "1" in any phrase. The inverse operation is carried out for correct printout of the solution.

Notice that the information given in the problem above was insufficient to allow solution of the set of equations to be solved. Therefore, STUDENT looked in its glossary for information concerning each of the units in this set of equations. It found the relationships "1 foot equals 12 inches." and "1 yard equals 3 feet." Using only the first fact, and the equation it implies, STUDENT is then able to solve the problem. Thus, in certain cases where a problem is not *analytic*, in the sense that it does not contain, explicitly stated, all the information needed for its solution, STUDENT is able to draw on a body of facts, picking out relevant ones, and use them to obtain a solution.

In certain problems, the transformation process does not yield a set of solvable equations. However, within this set of equations there exists a pair of variables (or more than one pair) such that the two variables are only "slightly different", and really name the same object in the model. When a set of equations is unsolvable, STUDENT searches for relevant global equations. In addition, it uses several heuristic techniques for identifying two "slightly different" variables in the equations. The problem below illustrates the identification of two variables where in one variable a pronoun has been substituted for a noun phrase in the other variable. This identification is made by

checking all variables appearing *before* one containing the pronoun, and finding one which is identical to this pronoun phrase, with a substitution of a string of any length for the pronoun.

(THE PROBLEM TO BE SOLVED IS)

(THE NUMBER OF SOLDIERS THE RUSSIANS HAVE IS ONE HALF OF THE NUMBER OF GUNS THEY HAVE. THE NUMBER OF GUNS THEY HAVE IS 7000. WHAT IS THE NUMBER OF SOLDIERS THEY HAVE Q.)

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION (ASSUMING THAT)

((NUMBER OF SOLDIERS (THEY/PRO) (HAVE/VERB)) IS EQUAL TO (NUMBER OF SOLDIERS RUSSIANS (HAVE/VERB)))

(THE NUMBER OF SOLDIERS THEY HAVE IS 3500)

If two variables match in this fashion, STUDENT assumes the two variables are equal, prints out a statement of this assumption, as shown, and adds an equation expressing this equality to the set to be solved. The solution procedure is tried again, with this additional equation. In this example, the additional equation was sufficient to allow determination of the solution.

The example below is again a non-analytic problem. The first set of equations developed by STUDENT is unsolvable. Therefore, STUDENT tries to find some relevant equations in its store of global information.

(THE PROBLEM TO BE SOLVED IS)

(THE GAS CONSUMPTION OF MY CAR IS 15 MILES PER GALLON. THE DISTANCE BETWEEN BOSTON AND NEW YORK IS 250 MILES. WHAT IS THE NUMBER OF GALLONS OF GAS USED ON A TRIP BETWEEN NEW YORK AND BOSTON Q.)

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION (USING THE FOLLOWING KNOWN RELATIONSHIPS) ((EQUAL (DISTANCE) (TIMES (SPEED) (TIME))) (EQUAL (DISTANCE) (TIMES (GAS CONSUMPTION)

((NUMBER OF GALLONS OF GAS USED)))

(ASSUMING THAT)

((DISTANCE) IS EQUAL TO (DISTANCE BETWEEN BOSTON AND NEW YORK)

(ASSUMING THAT)

((GAS CONSUMPTION) IS EQUAL TO (GAS CONSUMPTION OF MY CAR))

(ASSUMING THAT)

((NUMBER OF GALLONS OF GAS USED) IS EQUAL TO (NUMBER OF GALLONS OF GAS USED ON TRIP BETWEEN NEW YORK AND BOSTON))

(THE NUMBER OF GALLONS OF GAS USED ON A TRIP BETWEEN NEW YORK AND BOSTON IS 16.66 GALLONS)

It uses the first word of each variable string as a key to its glossary. The one exception to this rule is that the words "number of" are ignored if they are the first two words of a variable string. Thus, in this problem, STUDENT retrieved equations which were stored under the key words *distance*, *gallons*, *gas*, and *miles*. Two facts about *distance* had been stored earlier; "distance equals speed times time" and "distance equals gas consumption times number of gallons of gas used". The equations implicit in these sentences were stored and retrieved now—as possibly useful for the solution of this problem. In fact, only the second is relevant.

Before any attempt is made to solve this augmented set of equations, the variables in the augmented set are matched, to identify "slightly different" variables which refer to the same object in the model. In this example "(DISTANCE)," "(GAS CONSUMPTION)," and "(NUMBER OF GALLONS OF GAS USED)" are all identified with "similar" variables. The following conditions must be satisfied for this type of identification of variables P1 and P2:

- 1) P1 must appear later in the problem than P2.
- 2) P1 is completely contained in P2 in the sense that P1 is a contiguous substring within P2.

This identification reflects a syntactic phenomenon where a truncated phrase, with one or more modifying phrases dropped, is often

used in place of the original phrase. For example, if the phrase "the length of a rectangle" has occurred, the phrase "the length" may be used to mean the same thing. This type of identification is distinct from that made using pronoun substitution.

In the example above, a stored schema was used by identifying the variables in the schema with the variables that occur in the problem. This problem is solvable because the key phrases "distance", "gas consumption" and "number of gallons of gas used" occur as substrings of the variables in the problem. Since STUDENT identifies each generic key phrase of the schema with a particular variable of the problem, any schema can be used only once in a problem. Because STUDENT handles schema in this *ad hoc* fashion it cannot solve problems in which a relationship such as "distance equals speed times time" is needed for two different values of distance, speed, and time.

#### E. Possible Idiomatic Substitutions

There are some phrases which have a dual character, depending on the context. In the example below, the phrase "perimeter of a rectangle" becomes a variable with no reference to its meaning, or definition, in terms of the length and width of the rectangle. This definition is unneeded for solution.

(THE PROBLEM TO BE SOLVED IS)

(THE SUM OF THE PERIMETER OF A RECTANGLE AND THE PERIMETER OF A TRIANGLE IS 24 INCHES. IF THE PERIMETER OF THE RECTANGLE IS TWICE THE PERIMETER OF THE TRIANGLE, WHAT IS THE PERIMETER OF THE TRIANGLE Q.)

(THE PERIMETER OF THE TRIANGLE IS 8 INCHES)

However, the following problem is stated in terms of the perimeter, length and width of the rectangle. Transforming the English into equations is not sufficient for solution. Neither retrieving and using an equation about "inches", the unit in the problem, nor identifying "length" with a longer phrase serve to make the problem solvable. Therefore, STUDENT looks in its dictionary of possible idioms, and

finds one which it can try in the problem. STUDENT

(THE PROBLEM TO BE SOLVED IS)  
(THE LENGTH OF A RECTANGLE IS 8 INCHES MORE THAN THE WIDTH OF THE RECTANGLE. ONE HALF OF THE PERIMETER OF THE RECTANGLE IS 18 INCHES. FIND THE LENGTH AND THE WIDTH OF THE RECTANGLE.)

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION TRYING POSSIBLE IDIOMS

(THE PROBLEM WITH AN IDIOMATIC SUBSTITUTION IS)

(THE LENGTH OF A RECTANGLE IS 8 INCHES MORE THAN THE WIDTH OF THE RECTANGLE. ONE HALF OF TWICE THE SUM OF THE LENGTH AND WIDTH OF THE RECTANGLE IS 18 INCHES. FIND THE LENGTH AND THE WIDTH OF THE RECTANGLE.)

(ASSUMING THAT)

((LENGTH) IS EQUAL TO (LENGTH OF RECTANGLE))

(THE LENGTH IS 13 INCHES)

(THE WIDTH OF THE RECTANGLE IS 5 INCHES)

actually had two possible idiomatic substitutions which it could have made for "perimeter of a rectangle"; one was in terms of the length and width of the rectangle and the other was in terms of the shortest and longest sides of the rectangle. When there are two possible substitutions for a given phrase, one is tried first, namely the one STUDENT has been told about most recently. In this problem, the correct one was fortuitously first. If the other had been first, the revised problem would not have been solvable, and eventually the second (correct) substitution would have been made. Only one non-mandatory idiomatic substitution is ever made at one time, although the substitution is made for all occurrences of the phrase chosen.

In this problem, the idiomatic substitution made allows the problem to be solved, after identification of the variables "length" and "length of rectangle". The retrieved equation about inches was not needed. However, its presence in the set of equations to be solved did not sidetrack the solver in anyway.

This use of possible, but non-mandatory idiomatic substitutions can also be used to give STUDENT a way to solve problems in which two phrases denoting one particular variable are quite different. For example, the phrase, "students who passed the admissions test" and "successful candidates" might be describing the same set of people. However, since STUDENT knows nothing of the "real world" and its value system for success, it would never identify these two phrases. However, if told that "successful candidates" sometimes means "students who passed the admissions test", it would be able to solve a problem using these two phrases to identify the same variable. Thus, possible idiomatic substitutions serve the dual purpose of providing tentative substitutions of definitions, and identification of synonymous phrases.

#### F. *Special Heuristics.*

The methods thus far discussed have been applicable to the entire range of algebra problems. However, for special classes of problems, additional heuristics may be used which are needed for members of the class, but not applicable to other problems. An example is the class of age problems, as typified by the problem below.

(THE PROBLEM TO BE SOLVED IS)  
(BILL S FATHER S UNCLE IS TWICE AS OLD AS BILL S FATHER. 2 YEARS FROM NOW BILL S FATHER WILL BE 3 TIMES AS OLD AS BILL. THE SUM OF THEIR AGES IS 92. FIND BILL S AGE.)  
(BILL S AGE IS 8)

Before the age problem heuristics are used, a problem must be identified as belonging to that class of problems. STUDENT identifies age problems by any occurrence of one of the following phrases, "as old as", "years old" and "age". This identification is made immediately after all words are looked up in the dictionary and tagged by function. After the special heuristics are used the modified problem is transformed to equations as described previously.

The need for special methods for age problems arises because of the conventions used for denoting the variables, all of which are ages. The word age is usually not used explicitly,

but is implicit in such phrases as “as old as”. People’s names are used where their ages are really the implicit variables. In the example, for instance, the phrase “Bill’s father’s uncle” is used instead of the phrase “Bill’s father’s uncle’s age”.

STUDENT uses a special heuristic to make all these ages explicit. To do this, it must know which words are “person words” and therefore, may be associated with an age. For this problem STUDENT has been told that *Bill*, *father*, and *uncle* are person words. The “spaces -s” following a word is the STUDENT representation for possessive, used instead of “apostrophe -s” for programming convenience. STUDENT inserts a “S AGE” after every person word *not* followed by a “S” (because this “S” indicates that the person word is being used in a possessive sense, not as an independent age variable). Thus, as indicated, the phrase “BILL S FATHER S UNCLE” becomes “BILL S FATHER S UNCLE S AGE”.

In addition to changing phrases naming people to ones naming ages, STUDENT makes certain special idiomatic substitutions. For the phrase “their ages”, STUDENT substitutes a conjunction of all the age variables encountered in the problem. In the example, for “THEIR AGES” STUDENT substitutes “BILL S FATHER S UNCLE S AGE AND BILL S FATHER S AGE AND BILL S AGE”. The phrases “as old as” and “years old” are then deleted as dummy phrases not having any meaning, and “will be” and “was” are changed to “is”. There is no need to preserve the tense of the copula, since the sense of the future or past tense is preserved in such prefix phrases as “2 years from now”, or “3 years ago”.

The remaining special age problem heuristics are used to process the phrases “in 2 years”, “5 years ago” and “now”. The phrase “2 years from now” is transformed to “in 2 years” before processing. These three time phrases may occur immediately after the word “age”, (e.g., Bill’s age 3 years ago”) or at the beginning of the sentence. If a time phrase occurs at the beginning of the sentence, it implicitly modifies all ages mentioned in the sentence, *except* those followed by their own time phrase. For example, “In 2 years Bill’s father’s age will be 3

times Bill’s age” is equivalent to “Bill’s father’s age in 2 years will be 3 times Bill’s age in 2 years”. However, “3 years ago Mary’s age was 2 times Ann’s age now” is equivalent to “Mary’s age 3 years ago was 2 times Ann’s age now”. Thus prefix time phrases are handled by distributing them over all ages not modified by another time phrase.

After these prefix phrases have been distributed, each time phrase is translated appropriately. The phrase “in 5 years” causes 5 to be added to the age it follows, and “7 years ago” causes 7 to be subtracted from the age preceding this phrase. The word “now” is deleted.

Only the special heuristics described thus far are necessary to solve the first age problem. The second age problem, given below, requires one additional heuristic not previously mentioned. This is a substitution for the phrase “was when” which effectively decouples the two facts combined in the first sentence. For “was when”, STUDENT substitutes “was K years ago. K years ago” where K is a new variable created for this purpose.

(THE PROBLEM TO BE SOLVED IS)  
(MARY IS TWICE AS OLD AS ANN WAS  
WHEN MARY WAS AS OLD AS ANN IS  
NOW. IF MARY IS 24 YEARS OLD, HOW  
OLD IS ANN Q.)

(THE EQUATIONS TO BE SOLVED ARE)  
(EQUAL X00008 ((ANN / PERSON) S  
AGE))  
(EQUAL ((MARY / PERSON) S AGE)  
24)  
(EQUAL (PLUS ((MARY / PERSON) S  
AGE) (MINUS X00007))) ((ANN / PER-  
SON) S AGE))  
(EQUAL ((MARY / PERSON) S AGE)  
(TIMES 2 (PLUS ((ANN / PERSON) S  
AGE) (MINUS X00007))))  
(ANN S AGE IS 18)

In the example, the first sentence becomes the two sentences: “Mary is twice as old as Ann X00007 years ago. X00007 years ago Mary was as old as Ann is now.” These two occurrences of time phrases are handled as discussed previously. Similarly the phrase “will be when” would be transformed to “in K years. In K years”.

These decoupling heuristics are useful not only for the STUDENT program but for people trying to solve age problems. The classic age problem about Mary and Ann, given above, took an MIT graduate student over 5 minutes to solve because he did not know this heuristic. With the heuristic he was able to set up the appropriate equations much more rapidly. As a crude measure of STUDENT's speed, note that STUDENT took less than one minute to solve this problem.

#### G. *When All Else Fails.*

For all the problems discussed thus far, STUDENT was able to find a solution eventually. In some cases, however, necessary global information is missing from its store of information, or variables which name the same object cannot be identified by the heuristics of the program. Whenever STUDENT cannot find a solution for any reason, it turns to the questioner for help. As in the problem below, it prints out "(DO YOU KNOW ANY MORE RELATIONSHIPS BETWEEN THESE VARIABLES)" followed by a list of the variables in the problem. The questioner can answer "yes" or "no". If he says "yes", STUDENT says "TELL ME", and the questioner can append another sentence to the statement of the problem.

(THE PROBLEM TO BE SOLVED IS)  
 (THE GROSS WEIGHT OF A SHIP IS  
 20000 TONS. IF ITS NET WEIGHT IS  
 15000 TONS, WHAT IS THE WEIGHT OF  
 THE SHIPS CARGO Q.)  
 THE EQUATIONS WERE INSUFFICIENT  
 TO FIND A SOLUTION  
 (DO YOU KNOW ANY MORE RELATION-  
 SHIPS AMONG THESE VARIABLES)  
 (GROSS WEIGHT OF SHIP)  
 (TONS)  
 (ITS NET WEIGHT)  
 (WEIGHT OF SHIPS CARGO)  
 yes  
 TELL ME  
 (the weight of a ships cargo is the difference  
 between the gross weight and the net weight)  
 THE EQUATIONS WERE INSUFFICIENT  
 TO FIND A SOLUTION  
 (ASSUMING THAT)  
 ((NET WEIGHT) IS EQUAL TO (ITS  
 NET WEIGHT)

(ASSUMING THAT)  
 ((GROSS WEIGHT) IS EQUAL TO  
 (GROSS WEIGHT OF SHIP))  
 (THE WEIGHT OF THE SHIPS CARGO  
 IS 5000 TONS)

In this problem, the additional information typed in (in lower case letters) was sufficient to solve the problem. If it was not, the question would be repeated until the questioner said "no", or provides sufficient information for solution of the problem.

In the problem below, the solution to the set of equations involves solving a quadratic equation, which is beyond the mathematical ability of the present STUDENT system. Note that in this case STUDENT reports that the equations were unsolvable, not simply insufficient for solution. STUDENT still requests additional information from the questioner. In the example, the questioner says "no", and STUDENT states that "I CANT SOLVE THIS PROBLEM" and terminates.

(THE PROBLEM TO BE SOLVED IS)  
 (THE SQUARE OF THE DIFFERENCE  
 BETWEEN THE NUMBER OF APPLES  
 AND THE NUMBER OF ORANGES ON  
 THE TABLE IS EQUAL TO 9. IF THE  
 NUMBER OF APPLES IS 7, FIND THE  
 NUMBER OF ORANGES ON THE TA-  
 BLE.)  
 UNABLE TO SOLVE THIS SET OF EQUA-  
 TIONS  
 TRYING POSSIBLE IDIOMS  
 (DO YOU KNOW ANY MORE RELATION-  
 SHIPS AMONG THESE VARIABLES)  
 (NUMBER OF APPLES)  
 (NUMBER OF ORANGES ON TABLE)  
 no  
 I CANT SOLVE THIS PROBLEM

#### H. *Summary of the STUDENT Subset of English*

The subset of English understandable by STUDENT is built around a core of sentence and phrase formats which can be transformed into expressions in the STUDENT deductive model. On this basic core is built a larger set of formats. Each of these are first transformed into a string built on formats in this basic set and then this string is transformed into an ex-

pression in the deductive model. For example, the format (\$ IS EQUAL TO \$) is changed to the basic format (\$ IS \$), and the phrase "IS CONSECUTIVE TO" is changed to "IS 1 PLUS". The constructions discussed earlier involving single object transitive verbs could have been handled this way, though for programming convenience they were not.

The basic linguistic form which is transformed into an equation is one containing "is" as a copula. The phrases "is equal to" and "equals" are both changed to the coupla "is". The auxiliary verbal constructions "is multiplied by", "is divided by" and "is increased by" are also acceptable as principal verbs in a sentence. As discussed in detail earlier, a sentence with no occurrence of "is" can have as a main verb a transitive verb immediately followed by a number. This number must be an element of the phrase which is the direct object of the verb, as in "Mary has three guppies". This type of transitive verb can also have a comparative structure as direct object, e.g., "Mary has twice as many guppies as Tom has fish".

This completes the repertoire of declarative sentence formats. Any number of declarative sentences may be conjoined, with ", and" between each pair, to form a new (complex) declarative sentence. A declarative sentence (even a complex declarative) can be made a presupposition for a question by preceding it with "IF" and following it with a comma and the question.

Questions, that is, requests for information from STUDENT, will be understood if they match any of the following patterns (where \$ will match any string, and \$1 any one word):

(WHAT ARE \$ AND \$)	(WHAT IS \$)
(FIND \$ AND \$)	(FIND \$)
(HOW MANY \$ DO \$ HAVE)	(HOW MANY \$ DOES \$ HAVE)
(HOW MANY \$1 IS \$)	

This completes the summary of the set of input formats presently understood by STUDENT. This set can be enlarged in two distinct ways. One is to enlarge the set of basic formats, using standard subroutines to aid in de-

fining, for each new basic format, its interpretation in the deductive model. The other method of extending the range of STUDENT input is to define transformations from new input formats to previously understood basic or extension formats.

Even if a story problem is stated within the subset of English acceptable to STUDENT, this is not a guarantee that this problem can be solved by STUDENT (assuming it to be solvable). Two phrases describing the object must be at worst only "slightly different" by the criteria prescribed earlier. Appropriate global information must be available to STUDENT, and the algebra involved must not exceed the abilities of the solver. However, though most algebra story problems found in the standard texts cannot be solved by STUDENT exactly as written, the author has usually been able to find some paraphrase of almost all such problems which is solvable by STUDENT.

#### I. *Limitations of the STUDENT Subset of English*

The techniques presented here are general and can be used to enable a computer program to accept and understand a fairly extensive subset of English for a fixed semantic base. However, the current STUDENT system is experimental and has a number of limitations.

STUDENT's interpretation of the input is based on format matching. If each format is used to express the meaning understood by STUDENT, no misinterpretation will occur. However, these formats occur in English discourse even in algebra story problems, in semantic contexts not consistent with STUDENT's interpretation of these formats. For example, a sentence containing ", and" is always interpreted by STUDENT as the conjunction of two declarative statements. Therefore, the sentence "Tom has 2 apples, 3 bananas, and 4 pears." would be incorrectly divided into the two "sentences", "Tom has 2 apples, 3 bananas." and "4 pears."

Each of the operator words shown in Figure 2 must be used as an operator in the context as shown or a misinterpretation will result. For example, the phrase "the number of times I went to the movies" which should be inter-

preted as a variable string will be interpreted incorrectly as the product of the two variables "number of" and "I went to the movies", because "times" is always considered to be an operator. Similarly, in the current implementation of STUDENT, "of" is considered to be an operator if it is preceded by any number. However, the phrase "2 of the boys who passed" will be misinterpreted as the product of "2" and "the boys who passed".

These examples obviously do not constitute a complete list of misinterpretations and errors STUDENT will make, but it should give the reader an idea of limitations on the STUDENT subset of English. In principle, all of these restrictions could be removed. However, removing some of them would require only minor changes to the program, while others would require techniques not used in the current system.

For example, to correct the error in interpreting "2 of the boys who passed", one can simply check to see if the number before the "of" is less than 1, and if so, only then interpret "of" as an operator "times". However, a much more sophisticated grammar and parsing program would be necessary to distinguish different occurrences of "and" and correctly extract simpler sentences from complex coordinate and subordinate sentences.

Because of limitations of the sort described above, and the fact that the STUDENT system currently occupies almost all of the computer memory, STUDENT serves principally as a demonstration of the power of the techniques utilized in its construction. However, I believe that on a larger computer one could use these techniques to construct a system of practical value which would communicate well with people in English over the limited range of material understood by the program.

#### IV. CONCLUSION

##### A. Results

The purpose of the research reported here was to develop techniques which facilitate natural language communication with a computer. A semantic theory of coherent discourse was proposed as a basis for the design and understanding of such man-machine systems. This

theory was only outlined, and much additional work remains to be done. However, in its present rough form, the theory served as a guide for construction of the STUDENT system, which can communicate in a limited subset of English.

The language analysis in STUDENT is an implementation of the analytic portion of this theory. The STUDENT system has a very narrow semantic base. From the theory it is clear that by utilizing this knowledge of the limited range of meaning of the input discourse, the parsing problem becomes greatly simplified, since the number of linguistic forms that must be recognized is very small. If a parsing system were based on any small semantic base, this same simplification would occur. This suggests that in a general language processor, some time might be spent putting the input into a semantic context before going ahead with the syntactic analysis.

The semantic base of the STUDENT language analysis is delimited by the characteristics of the problem solving system embedded in it. STUDENT is a question-answering system which answers questions posed in the context of "algebra story problems." We shall use four general criteria for evaluating this question-answering system.

1) *Extent of Understanding.* Other question-answering systems analyze input sentence by sentence. Although a representation of the meaning of all input sentences may be placed in some common store, no syntactic connection is ever made between sentences.

In the STUDENT system, an acceptable input is a sequence of sentences, such that these sentences cannot be understood by just finding the meanings of the individual sentences, ignoring their local context. Inter-sentence dependencies must be determined, and inter-sentence syntactic relationships must be used in this case for solution of the problem given. This extension of the syntactic dimension of understanding is important because such inter-sentence dependencies (e.g., the use of pronouns) are very commonly used in natural language communication.

The semantic model in the STUDENT system is based on one relationship (equality) and

five basic arithmetic functions. Composition of these functions yield other functions which are also expressed as individual linguistic forms in the input language. The input language is richer in expressing functions than Lindsay's or Raphael's system. Some logic-based question-answering systems may have more relationships (predicates) allowable in the input, but do not allow any composition of these predicates. The logical combinations of predicates used are only those expressed in the input as logical combinations (using and, or, etc.)

The deductive system in STUDENT, as in Lindsay's and Raphael's programs, is designed for the type of questions to be asked. It can only deduce answers of a certain type from the input information, that is arithmetic values satisfying a set of equations. In performing its deductions it is reasonably sophisticated in avoiding irrelevant information, as are the other two mentioned. It lacks the general power of a logical system, but is much more efficient in obtaining its particular class of deductions than would be a general deductive system utilizing the axioms of arithmetic.

2) *Facility for Extending Abilities.* Extending the syntactic abilities of most other question-answering systems would require reprogramming. In the STUDENT system new definitional transformations can be introduced at run time without any reprogramming. The information concerning these transformations can be input in English, or in a combination of English and METEOR, if that is more appropriate. New syntactic transformations must be added by extending the program.

The semantic base of the STUDENT system can be extended only by adding new program, as is true of other question-answering systems. However STUDENT is organized to facilitate such extensions, by minimizing the interactions of different parts of the program. The necessary information need only be added to the program equivalent of the table of operators in Figure 2.

Similarly, the deductive portion of STUDENT, which solves the derived set of equations, is an independent package. Therefore, a new extended solver can be added to the system by just replacing the package, and maintaining

the input-output characteristics of this subroutine.

3) *Knowledge of Internal Structure Needed by User.* Very little if any internal knowledge of the workings of the STUDENT system need be known by the user. He must have a firm grasp of the type of problem that STUDENT can solve, and a knowledge of the input grammar. For example, he must be aware that the same phrase must always be used to represent the same variable in a problem, within the limits of similarity defined earlier. He must realize that even within these limits STUDENT will not recognize more than one variation on a phrase. But if the user does forget any of these facts, he can still use the system, for the interaction discussed in the next section allows him to make amends for almost any mistake.

4) *Interaction with the User.* The STUDENT system is embedded in a time-sharing environment and this greatly facilitates interaction with the user. STUDENT differentiates between its failure to solve a problem because of its mathematical limitations and failure from lack of sufficient information. In case of failure it asks the user for additional information, and suggests the nature of the needed information (relationships among variables of the problem). It can go back to the user repeatedly for information until it has enough to solve the problem, or until the user gives up.

STUDENT also reports when it does not recognize the format of an input sentence. Using this information as a guide, the user is in a teaching-machine type situation, and can quickly learn to speak STUDENT's brand of input English. By monitoring the assumptions that STUDENT makes about the input, and the global information it uses, the user can stop the system and reword a problem to avoid an unwanted ambiguity, or add new general information to the global information store.

#### B. *Extensions*

The present STUDENT system has reached the maximum size allowable in the LISP system on a thirty-two thousand word IBM 7094. Therefore, very little can be added directly to the present system. All the programming extensions mentioned here are predicated on the



existence of a computer with a much larger memory.

Without inventing any new techniques, I think that the STUDENT system could be made to understand most of the algebra story problems that appear in first year high school text books. If new operators, new combinations of arithmetic operations occur, they can easily be added to the subroutine which maps the kernel English sentences into equations. The number of formats recognizable in the system can be increased without reprogramming through the machinery available for storing global information. The problems it would not handle are those having excessive verbiage or implied information about the world not expressible in a single sentence.

As mentioned earlier, the system can now make use of any given schema only once in solving a problem. This is because the schema equation is added to the set of equations to be solved, and the variables in the schema only identified with one another set of variables appearing in the problem. For example, if "distance equals speed times time" were the schema, the "distance", as a variable in the schema might be set equal to "distance traveled by train" or "distance traveled by plane", but not both in the same problem. This problem could be resolved by not adding the schema equation directly to the set of equations to be solved, but by looking for consistent sets of variables to identify with the schema variables. Then STUDENT could add an instance of the schema equations, with the appropriate substitutions, for each consistent set of variables found which are "similar" to the schema variables.

At the moment the solving subroutine of STUDENT can only perform linear operations on literal equations, and substitutions of numbers in polynomials and exponentials. It would be relatively easy to add the facility for solving quadratic or even higher order solvable equations. One could even add, quite easily, sufficient mechanisms to allow the solver to perform the differentiation needed to do related rate problems in the differential calculus.

The semantic base of the STUDENT system could be expanded. In order to add the relations recognized by the SIR system of Raphael, for

example, one would have to add on the lowest level of the STUDENT program the set of kernel sentences understood in SIR, their mapping to the SIR model, and the question-answering routine to retrieve facts. Then the apparatus of the STUDENT system would process much more complicated input statements for the SIR model. One serious problem which arises when the semantic base is extended is based on the fact that one kernel may have an interpretation in terms of two different semantic bases. For example, "Tom has 3 fish." can be interpreted in both SIR and the present STUDENT system. To resolve this semantic ambiguity, the program can check the context of the ambiguous statement to see if there has been one consistent model into which all the other statements have been processed. If the latter condition does not determine a single preferred interpretation for the statement, then both interpretations can be stored.

One use for our model for generation and analysis of discourse would be as a hypothesis about the linguistic behavior of people. STUDENT may be a good predictive model for the behaviour of people when confronted with an algebra problem to solve. This can be tested, and such a study may lead to a better understanding of human behaviour, and/or a better reformulation of this theory of language processing.

I think we are far from writing a program which can understand all, or even a very large segment of English. However, within its narrow field of competence, STUDENT has demonstrated that "understanding" machines<sup>12</sup> can be built. Indeed, I believe that using the techniques developed in this research, one could construct a system of practical value which would communicate well with people in English over the range of material understood by the program.

#### REFERENCES

1. BOBROW, D. G., "METEOR: A LISP Interpreter for String Transformations"; in (13)
2. MCCARTHY, J., et. al., *LISP 1.5 Programmers Reference Manual*; MIT Press, Cambridge, Mass.; 1963

3. CORBATO, F. J., et. al., *The Compatible Time-Sharing System*; MIT Press, Cambridge, Mass.; 1963
4. GREEN, B. F., et. al., "Baseball: An Automatic Question Answerer"; *Proc. WJCC*; May 1961
5. LINDSAY, R. K., "Inferential Memory as the Basis of Machines Which Understand Natural Language," in (14)
6. RAPHAEL, B., "A Computer Program Which 'Understands'"; *Proc. FJCC*; Spartan Press, Baltimore, Maryland; 1964
7. SIMMONS, R. F., "Answering English Questions by Computer—A Survey"; SDC Report SP-1556, Santa Monica, California; April 1964
8. BOBROW, D. G., "Natural Language Input for a Computer Problem Solving System", Ph.D. Thesis, Mathematics Department, M.I.T., Cambridge, Mass.; 1964
9. QUINE, W. V., *Word and Object*; MIT Press, Cambridge, Mass.; 1960
10. CHOMSKY, A. N., *Syntactic Structures*; Mouton and Co.; S-Grauenhage; 1957
11. YNGVE, V., "A Model and an Hypothesis for Language Structure," *Proceedings of the American Philosophical Society*, vol. 104, No. 5; 1960
12. MINSKY, M., "Steps toward Artificial Intelligence" in (14)
13. BERKLEY, E. C. and BOBROW, D. G. (editors) *The Programming Language LISP: Its Operation and Applications*, Information International Inc., Cambridge, Mass.; 1964
14. FEIGENBAUM, E. and FELDMAN, J. (editors) *Computers and Thought*, McGraw Hill, New York; 1963

# THE UNIT PREFERENCE STRATEGY IN THEOREM PROVING\*

*Lawrence Wos, Daniel Carson, and George Robinson  
Argonne National Laboratory  
Argonne, Illinois*

## *Unit Preference and Set of Support Strategies*

The theorems, axioms, etc., to which the algorithm and strategies described in this paper are applied are stated in a normal form defined as follows: A *literal* is formed by prefixing a predicate letter to an appropriate number of arguments (constants, variables, or expressions formed with the aid of function symbols) and then perhaps writing a negation sign (–) before the predicate letter. For example:

$P(b,x) \quad -P(b,x) \quad Q(y) \quad R(a,b,x,z,c) \quad S$

are all literals if P, Q, R, and S are two-, one-, five-, and zero-place predicate letters, respectively. The predicate letter is usually thought of as standing for some n-place relation. Then the literal  $P(a,b)$ , for example, is thought of as saying that the ordered pair (a,b) has the property P. The literal  $-P(a,b)$  is thought of as saying that (a,b) does not have the property P.

One may build a *clause* by writing a sequence of literals separated by disjunction (logical “or”) signs. Logical “or” will be symbolized by a small letter “v” (distinguishable from possible uses of “v” as a variable from context). Where it is desired to indicate dependence of a particular argument of a predicate letter upon one or more other variables, function symbols are employed. The clause

$P(x,y,z) \vee Q(x,f(x,y))$

thus says that either the ordered triple (x,y,z) has the property P or the ordered pair (x,f(x,y)) has the property Q (or both). Each of the variables in a clause is then thought of as being universally quantified (that is, if the variable x occurs in a clause, the clause is assumed to be preceded by an implicit quantifier, “for each x.” Functional expressions such as f(x,y) are then treated as existentially quantified variables. Roughly speaking, f(x,y) in the example above stands for an element (depending on x and y) which forms, when used as the second element with x as the first element, an ordered pair which has the property Q. A *unit clause* is a clause composed of a single literal.

Finally, one may consider a sequence of clauses implicitly joined by logical “and” (conjunction). Such a sequence of clauses will be said to be in (*conjunctive*) *normal form*.

*Instantiation*, as applied to this normal form, can be thought of as the forming of a possibly less general (more specific) *instance* of a clause by performing a systematic replacement of variables by constants, new variables, or by expressions formed with the aid of function symbols. Substituting b for x and f(d,u) for y in the clause

$P(x,y) \vee Q(b,y)$

would yield a less general instance of that clause:

$P(b,f(d,u)) \vee Q(b,f(d,u)).$

---

\* Work performed under the auspices of the U. S. Atomic Energy Commission.

The latter is less general in that, while it can be deduced from the former, the former cannot be deduced from the latter.

Early computer-oriented theorem-proving efforts employed search techniques involving instantiation of a given set of logical formulae. In these methods, successively larger sets of constants were generated. As each set was generated, all permissible substitutions of such constants for variables were made in the original formulae, producing successively larger sets of instances. These instances differed from the original formulae in that they included no logical quantifiers: "For each  $x$ ," "For some  $x$ ," etc. For sets of such quantifier-free formulae, straightforward techniques were known for determining validity or inconsistency. P. C. Gilmore<sup>2</sup> described an IBM 704 program using the technique of conversion of sets of quantifier-free formulae from conjunctive normal form to disjunctive normal form. Davis and Putnam introduced a substantial improvement in the method of testing such sets of quantifier-free formulae. A 704 program<sup>4</sup> incorporating this improvement proved to be several orders of magnitude faster than Gilmore's program for the same machine.

These instantiation techniques employed the argument forms *existential instantiation* and *universal instantiation* (see for example Quine<sup>5</sup>) to infer the quantifier-free instances from the original formulae. All permissible substitutions were made in a systematic, exhaustive manner, guaranteeing that if a proof existed of the desired theorem, it would be captured in the steadily expanding sets of instances. The disastrous rate of growth of these sets, due to the inclusion of numerous unprofitable inferences, spelled the doom of exhaustive instantiation. Study of the nature of the instances which could be expected to result from such a program led, however, to the suspicion that the logical completeness of the method could be retained but the combinatoric explosion substantially reduced by considering the generalizations represented by classes of similar instances. This led to the formulation of computer-oriented rules of inference which suppressed *universal instantiation* and retained universally quantified variables in their more general form. These rules were codified and

put on a sound logical basis by J. A. Robinson<sup>3</sup> in his paper on the resolution principle. As a computer algorithm, he proposed that the original set of formulae be completely "resolved" in the sense that all possible applications of the inference rule *resolution* be made in a systematic, exhaustive manner.

Where exhaustive instantiation methods met their downfall was in blindly forming all possible maximally specific instances from a given set of clauses; once formed, such instances were never to be discarded. Only then was consideration given to what interactions might occur between the instances (specifically, to whether the set thus generated was inconsistent). Furthermore, in most interesting cases, many new objects to be substituted were manufactured, resulting in a combinatoric explosion of further instances.

Resolution, as proposed by J. A. Robinson, curtails the number of instances produced, in that it produces a new clause only when it can be determined in advance that two existing clauses will, when each is instantiated, yield a pair of instances that will interact, forming a clause which could not have been inferred from either parent clause taken by itself. Specifically, in resolution a pair of clauses (each of which is called a resolvent) is examined to see if there is a substitution which will transform the clauses into a pair of the form

$$L \vee L_1 \vee L_2 \vee \dots \vee L_m, \quad \neg L \vee K_1 \vee K_2 \vee \dots \vee K_n$$

From such a pair, the clause

$$L_1 \vee L_2 \vee \dots \vee L_m \vee K_1 \vee K_2 \vee \dots \vee K_n$$

can be inferred. This clause (called the resolvent) is then added to the set of clauses which have been accumulated from previous inferences. Furthermore, a substitution is considered only if it is the most general that could be made, thus maintaining the maximum degree of generality in the result. Another closely related method of inference is *factoring*: A substitution is sought such that (a) two or more of the literals of a clause will collapse into a single literal, and (b) no more general substitution would have the same effect. The resulting clause is called a *factor*.

Substitution of these rules of inference for the earlier instantiation inference rules was

shown to produce a reduction of the combinatoric explosion by a factor in excess of  $10^{50}$ , leading to some rather spectacular achievements when compared to the instantiation techniques. Nevertheless, the search algorithm employed a more or less random generation of resolvents inferred by means of the resolution principle. Using only these techniques, previously established bench-mark problems required prohibitive amounts of machine time. It seemed that a change of emphasis would be profitable.

One approach would be to concentrate upon the strategies of search. The current paper considers one such strategy, the *unit preference strategy*. This strategy has been implemented in a theorem-proving program now successfully running on the Control Data 3600. We will describe the search algorithm employed in the program, prove its soundness and completeness, consider some examples, and describe additional search strategies which can be employed to effect further improvement.

The principal search strategy arises from the fact that the object of the resolution principle is the generation through inference of two unit clauses which are manifestly contradictory. Here and elsewhere in this paper, *contradict* is used as in mathematics in the somewhat broader sense of conflict, rather than in the narrower sense frequently used in logic. Two clauses will be called contradictory if they are mutually exclusive (contrary) clauses. We do not require them to bear the relation of a clause to its negation. With this in mind, it seemed worthwhile to orient the program to produce shorter and still shorter clauses in preference to other possible inferences.

#### The Unit Preference Strategy

The data for consideration consists of a set of clauses in the normal form. The clauses correspond to a given set of axioms and the denial of a theorem to be proved from these axioms, for it is to be remembered that the program finds proofs and not theorems. Remember that all variables that occur are treated as universally quantified; all existentially quantified variables have been replaced by constants or functions.

The algorithm is divided into two sections, a unit section and a non-unit section. Where a

$j$ -clause is a clause of length  $j$  (i.e., has  $j$  literals), and the  $j$ -list is the list of  $j$ -clauses, the logic of the unit section is as follows, starting with  $j = 1$ :

1. Search the unit list and the  $j$ -list for a pair of elements  $C$  and  $D$  such that  $C$  is a unit,  $D$  is of length  $j$ , and for some literal  $m$  of  $D$  resolution has not been attempted for  $C$  and  $D$  on  $m$ ; if no such pair exists, execute step 2; if such a pair exists, execute step 3.

(The search in the program proceeds by taking the first unit and the first  $j$ -clause and examining each of its literals; the procedure is repeated with the same  $j$ -clause and the next unit, and continues until the unit list is exhausted. Then the process is applied to the next  $j$ -clause.)

2. If  $j$  is the length of the longest clause present, enter the non-unit section; if not, increase  $j$  by 1 and return to 1.
3. Resolve  $C$  and  $D$  on  $m$ ; if no resolvent is generated, return to 1 and resume the search; if a resolvent of length  $i > 0$  is generated, add the resolvent to the  $i$ -list, set  $j = i$ , and return to 1; if the empty resolvent is generated, execute the proof recovery. (The generation of the empty resolvent is equivalent to finding that the clauses  $C$  and  $D$  are manifestly contradictory.)

In order to avoid the possibility of being caught in an infinite loop, there is a constraint placed on step 3. The constraint is formulated in terms of the concept of the level of a clause. Let  $S_0$  be the original set of clauses; define  $S_i$  for  $i > 0$  to be the set of resolvents of  $S_{i-1}$  together with  $S_{i-1}$ . The level  $k$  of a clause  $C$  is 0 if  $C$  is input, is that of  $B$  if  $C$  is obtained by factoring  $B$ , and is 1 greater than the maximum of the levels of  $A$  and  $B$  if  $C$  is obtained by resolving  $A$  and  $B$ . The constraint imposed on 3 is: if the resolvent of  $C$  and  $D$  is a non-unit whose level is a specified bound  $k_0$ , or a unit whose level exceeds  $k_0$ , then it is not added to the corresponding list, and the pair is treated as if no resolvent were generated. To illustrate the difficulty thus avoided consider the set consisting of the clauses  $P(a)$ ,  $\neg P(x) \vee P(f(x))$ , which correspond to a subset of the Peano

axioms, and some clause of length 3. Without the level bound, the program would generate  $P(f(a)), P(f(f(a))), P(f(f(f(a))))$ , . . . , *ad infinitum*. We would be caught in an infinite loop which would continually present and execute the task of resolving a new unit with the same 2-clause instead of either passing to the proof recovery or to the non-unit section.

The non-unit section begins by setting  $j = 2$ , then:

1. Search the  $j$ -list for a clause  $D$  with literals  $l$  and  $m$  on which factoring has not been attempted; if one exists, execute 2; if not, execute 3.
2. Factor  $D$  on  $l$  and  $m$ ; if no factor is generated, return to 1; if a factor  $C$  of length  $i$  is generated, add  $C$  to the  $i$ -list, set  $j = i$ , and return to step 1 of the unit section.
3. If  $j = 2$ , increase  $j$  by 1 and return to step 1 of the non-unit section; if  $j \neq 2$ , execute 4.
4. Set  $h = 2$ , replace  $j$  by  $j-1$ , and execute 5.
5. Search the  $h$ -list and the  $j$ -list for a pair  $C$  and  $D$  such that  $C$  is an  $h$ -clause with a literal  $l$ ,  $D$  is a  $j$ -clause with a literal  $m$ , and resolution has not been attempted for  $C$  and  $D$  on  $l$  and  $m$ ; if no such pair exists, execute 6; if such a pair exists, execute 7.
6. If  $h \geq$  the maximum length of all clauses present, the program stops with the conclusion that no proof exists within level  $k_0$ ; if  $h <$  this maximum and  $h + 1 \geq j$ , replace  $j$  by  $h + j$  and return to 1 of the non-unit section; if  $h + 1 < j$ , replace  $h$  by  $h + 1$  and  $j$  by  $j - 1$  and return to 5.
7. Resolve  $C$  and  $D$  on  $l$  and  $m$ ; if no resolvent is generated, return to 5; if a resolvent  $B$  of length  $i$  is generated, then add  $B$  to the  $i$ -list, set  $j = i$ , and return to 1 of the unit section. Again the level constraint imposed on Step 3 of the unit section applies.

#### Soundness and Completeness

The soundness of the procedure follows from the fact that if  $C$  is in  $S_i$  or a factor of some  $D$  in  $S_i$  for any  $i$ ,  $C$  is implied by  $S_0$ , i.e.,  $C$  is a consequence of  $S_0$ . So, if  $C$  and  $D$  are elements

of the unit list which conflict (are manifestly contradictory),  $S_0$  is inconsistent.

The argument for completeness is as follows: J. A. Robinson<sup>3</sup> proved in effect that if  $S_0$  is an inconsistent set, there exists a  $k$  such that  $S_k$  contains the empty resolvent; this implies that  $S_{k-1}$  contains two clauses which are units or have unit factors which conflict. The claim is that when the level bound  $k_0$  of the program is equal to this  $k$ , whether  $k_0$  is immediately set equal to  $k$  or set equal to  $1, 2, \dots, k$ , the desired unit conflict will be obtained. At any given time the set of clauses which occur on any list is contained in  $S_k$ , and is therefore finite. The search through any list for potential factors or through any pair of lists for potential resolvents is a finite process since all lists are finite. The number of searches through any list is limited by the size of the input and the number of clauses which are adjoined and is, therefore, finite also. So any clause which occurs on any list will do so after a finite number of steps. In a similar fashion we can see that all clauses on all lists will be factored and, within the level bound, pairwise resolved in a finite number of steps. Consider the case where  $k_0$  is sufficiently large to have  $S_{k_0}$  contain the empty resolvent. If  $k_0 = 1$ ,  $S_0$  contains the clauses which yield the conflict, and the previous argument shows that the program will examine the pair in question. If  $k_0 = 2$ , the same argument shows that  $S_1$  will be contained in the lists of clauses, and again the conflict will be obtained. Applying the argument  $k_0$  times shows  $S_{k_0-1}$  will be contained in the lists of clauses. So we have proven the following lemma.

**Lemma:** Using the search algorithm with level bound  $k_0$  such that  $S_{k_0}$  contains the empty resolvent, the program finds a proof if and only if  $S_0$  is unsatisfiable.

#### Subsidiary Strategies

The most important subsidiary strategy is based on the concept of a chain. With the appropriate grouping, any clause which occurs on any list is expressible in terms of the elements of  $S_0$  together with the operators of resolution and factoring. Such an expression is called a *chain*, and the number of resolutions which occur therein is its *length*. The elements

of  $S_0$  and the factors of those elements are chains of length 0; those of  $S_1$  have chains of length 0 and 1; those of  $S_2$  have chains of length of 0, 1, 2 and 3.

If  $T$  is a nonempty subset of  $S_0$  and  $C$  is a chain of length 0 whose single element is in  $T$  or is a factor of an element in  $T$ , we say  $C$  is supported by  $T$  or  $C$  has  $T$ -support. Chains  $C$  of length greater than 0 have  $T$ -support if, for every resolution, that occurs in  $C$ , at least one of the resolvents has  $T$ -support. The clause  $B$  is said to be derived from  $T$ , if its chain (its expression or derivation in  $S_0$ ) has  $T$ -support.

The strategy employed here is to choose  $T$  and generate only those elements of  $S_i$  for  $i > 0$  which are derived from  $T$ . The choice of  $T$  obviously has a profound effect on the number of clauses generated during the search for a proof. The question remains as to which available choices of  $T$  preserve completeness of the procedure. By the lemma of the previous section,  $T = S_0$  is admissible under the appropriate conditions. For a given  $S_0$  the clauses might be divided into 3 categories; those which correspond to the basic axioms of the theory under study; those which correspond to the special hypotheses of the theorem under consideration, and those which correspond to the denial of the conclusion of the theorem. For example, consider the theorem: in a group, if the square of every element is the identity, the group is commutative. The first category would consist of a set of axioms which characterize groups; the second would consist of the axiom, for every  $x$  in the group  $x^2 = e$ ; the third would be the denial of commutativity, there exist  $a$  and  $b$  such that  $ab \neq ba$ . (This example will be considered later under various conditions.) It appears that, where  $T$  is the join of the last two categories, this choice of  $T$  for set of support preserves completeness and is most valuable in obtaining proofs in a reasonable amount of time.

Among the other strategies which can be employed are: deletion of the unit clause  $B$  upon generation if the unit list contains a clause  $A$  such that every instance of  $B$  is also one of  $A$ ; deletion upon generation of a clause if it contains two literals which have opposite sign but are otherwise identical; deletion of  $B$

if the unit  $A$  is a resolvent of  $B$  with some  $C$  such that all instances of some literal  $k$  of  $B$  are contained in the set of instances of  $A$ .

EXAMPLE 1: In an associative system with left and right solutions, there is a right identity element.

Basic Axioms:

- |     |                                |   |                  |
|-----|--------------------------------|---|------------------|
| A1. | $-P(x,y,u) \vee -P(y,z,v)$     | } |                  |
|     | $\vee -P(x,v,w) \vee P(u,z,w)$ |   |                  |
| A2. | $-P(x,y,u) \vee -P(y,z,v)$     | } |                  |
|     | $\vee -P(u,z,w) \vee P(x,v,w)$ |   |                  |
|     |                                |   | (associativity)  |
| A3. | $P(g(x,y),x,y)$                |   | (left solution)  |
| A4. | $P(x,h(x,y),y)$                |   | (right solution) |
| A5. | $P(x,y,f(x,y))$                |   | (closure)        |

Negation of Conclusion:

- A6.  $-P(j(x),x,j(x))$  (no right identity)

When A6 was taken as the only member of the set of support, the computer generated the following proof in 35 milliseconds with 11 clauses in memory at the time the proof was detected:

1.  $-P(x,y,u) \vee -P(y,z,v)$   
 $-P(x,v,w) \vee P(u,z,w)$  (A1)
2.  $P(g(x,y),x,y)$  (A3)
3.  $P(x,h(x,y),y)$  (A4)
4.  $-P(j(x),x,j(x))$  (A6)
5.  $-P(x,y,j(z)) \vee -P(y,z,v)$   
 $\vee -P(x,v,j(z))$  (from 4 and 1)
6.  $-P(y,z,v) \vee -P(g(y,j(z)),v,j(z))$   
(from 2 and 5)
7.  $-P(v,z,v)$  (from 2 and 6)

Since 3 and 7 are manifestly contradictory, the proof is complete.

EXAMPLE 2: In an associative system with an identity element, if the square of every element is the identity, the system is commutative.

Basic Axioms:

- |     |                                |   |                  |
|-----|--------------------------------|---|------------------|
| A1. | $P(x,e,x)$                     |   | (right identity) |
| A2. | $P(e,x,x)$                     |   | (left identity)  |
| A3. | $-P(x,y,u) \vee -P(y,z,v)$     | } |                  |
|     | $\vee -P(u,z,w) \vee P(x,v,w)$ |   |                  |
| A4. | $-P(x,y,u) \vee -P(y,z,v)$     | } |                  |
|     | $\vee -P(x,v,w) \vee P(u,z,w)$ |   |                  |
|     |                                |   | (associativity)  |

*Special Hypothesis:*

- A5.  $P(x,x,e)$   
 (The square of every element is the identity.)

*Negation of Conclusion:*

- A6.  $P(a,b,c)$   
 A7.  $\neg P(b,a,c)$   
 (There are elements  $a$  and  $b$  which do not commute.)

When A1 through A7 were used as the set of support, the machine generated the following proof in 1.124 seconds with 119 clauses in memory at the time the proof was detected.

1.  $P(x,e,x)$  (A1)
2.  $P(e,x,x)$  (A2)
3.  $\neg P(x,y,u) \vee \neg P(y,z,v)$   
 $\vee \neg P(u,z,w) \vee P(x,v,w)$  (A3)
4.  $\neg P(x,y,u) \vee \neg P(y,z,v)$   
 $\vee \neg P(x,v,w) \vee P(u,z,w)$  (A4)
5.  $P(x,x,e)$  (A5)
6.  $P(a,b,c)$  (A6)
7.  $\neg P(b,a,c)$  (A7)
8.  $\neg P(x,y,e) \vee \neg P(y,w,v)$   
 $\vee P(x,v,w)$  (from 2 and 3)
9.  $\neg P(y,w,v) \vee P(y,v,w)$  (from 5 and 8)
10.  $\neg P(w,y,u) \vee \neg P(y,z,e)$   
 $\vee P(u,z,w)$  (from 1 and 4)
11.  $\neg P(w,z,u) \vee P(u,z,w)$  (from 5 and 10)
12.  $P(c,b,a)$  (from 6 and 11)
13.  $P(c,a,b)$  (from 12 and 9)
14.  $\neg P(c,a,b)$  (from 7 and 11)

Since 13 and 14 are manifestly contradictory, the proof is complete.

When, instead, only A5, A6, and A7 were used as the set of support, the machine generated the following proof in the faster time of 538 milliseconds with 72 clauses in memory at the time the proof was detected:

1.  $P(x,e,x)$  (A1)
2.  $P(e,x,x)$  (A2)
3.  $\neg P(x,y,u) \vee \neg P(y,z,v)$   
 $\vee \neg P(u,z,w) \vee P(x,v,w)$  (A3)
4.  $\neg P(x,y,u) \vee \neg P(y,z,v)$   
 $\vee \neg P(x,v,w) \vee P(u,z,w)$  (A4)
5.  $P(x,x,e)$  (A5)
6.  $P(a,b,c)$  (A6)
7.  $\neg P(b,a,c)$  (A7)

8.  $\neg P(y,z,v) \vee \neg P(e,z,w)$   
 $\vee P(y,v,w)$  (from 5 and 3)
9.  $\neg P(y,w,v) \vee P(y,v,w)$  (from 2 and 8)
10.  $\neg P(x,z,u) \vee \neg P(x,e,w)$   
 $\vee P(u,z,w)$  (from 5 and 4)
11.  $\neg P(w,z,u) \vee P(u,z,w)$  (from 1 and 10)
12.  $P(c,b,a)$  (from 6 and 11)
13.  $P(c,a,b)$  (from 12 and 9)
14.  $\neg P(c,a,b)$  (from 7 and 11)

Since 13 and 14 are manifestly contradictory, the proof is complete.

EXAMPLE 3: In a group, if the square of every element is the identity, the group is commutative.

*Basic Axioms:*

- A1.  $P(x,y,f(x,y))$  (closure)
- A2.  $P(e,x,x)$  } (existence of identity)
- A3.  $P(x,e,x)$  }
- A4.  $P(x,g(x),e)$  } (existence of inverse)
- A5.  $P(g(x),x,e)$  }
- A6.  $R(x,x)$  (reflexivity of =)
- A7.  $\neg R(x,y) \vee R(y,x)$  (symmetry of =)
- A8.  $\neg R(x,y) \vee \neg R(y,z) \vee R(x,z)$  (transitivity of =)
- A9.  $\neg P(x,y,u) \vee \neg P(x,y,v) \vee R(u,v)$  (multiplication is well-defined)
- A10.  $\neg P(x,y,u) \vee \neg P(y,z,v) \vee \neg P(u,z,w) \vee P(x,v,w)$  }
- A11.  $\neg P(x,y,u) \vee \neg P(y,z,v) \vee \neg P(x,v,w) \vee P(u,z,w)$  } (associativity)
- A12.  $\neg R(u,v) \vee \neg P(x,y,u) \vee P(x,y,v)$  }
- A13.  $\neg R(u,v) \vee \neg P(x,u,y) \vee P(x,v,y)$  }
- A14.  $\neg R(u,v) \vee \neg P(u,x,y) \vee P(v,x,y)$  }
- A15.  $\neg R(u,v) \vee R(f(x,u),f(x,v))$  }
- A16.  $\neg R(u,v) \vee R(f(u,y),f(v,y))$  }
- A17.  $\neg R(u,v) \vee R(g(u),g(v))$  (substitution for =)

*Special Hypothesis:*

- A18.  $P(x,x,e)$   
 (The square of every element is the identity.)

*Negation of Conclusion:*

- A19.  $P(a,b,c)$   
 A20.  $\neg P(b,a,c)$   
 (There are elements  $a$  and  $b$  which do not commute.)



With A18 through A20 used as the set of support, the computer generated the following proof in 54 seconds with 563 clauses in memory at the time the proof was detected:

1.  $P(x,y,f(x,y))$  (A1)
2.  $P(e,x,x)$  (A2)
3.  $P(x,e,x)$  (A3)
4.  $\neg P(x,y,u) \vee \neg P(y,z,v) \vee$   
 $\neg P(u,z,w) \vee P(x,v,w)$  (A10)
5.  $\neg P(x,y,u) \vee \neg P(y,z,v) \vee$   
 $\neg P(x,v,w) \vee P(u,z,w)$  (A11)
6.  $P(x,x,e)$  (A18)
7.  $P(a,b,c)$  (A19)
8.  $\neg P(b,a,c)$  (A20)
9.  $\neg P(y,z,v) \vee \neg P(e,z,w) \vee$   
 $P(y,v,w)$  (from 6 and 4)
10.  $\neg P(e,z,w) \vee$   
 $P(y,f(y,z),w)$  (from 1 and 9)
11.  $P(y,f(y,w),w)$  (from 2 and 10)
12.  $\neg P(x,y,z) \vee \neg P(y,z,v) \vee$   
 $P(x,v,e)$  (from 6 and 4)
13.  $\neg P(x,y,z) \vee$   
 $P(x,f(y,z),e)$  (from 1 and 12)
14.  $P(a,f(b,c),e)$  (from 7 and 13)
15.  $\neg P(b,y,u) \vee \neg P(y,z,a) \vee$   
 $\neg P(u,z,c)$  (from 8 and 4)
16.  $\neg P(e,z,a) \vee \neg P(b,z,c)$  (from 3 and 15)
17.  $\neg P(e,f(b,c),a)$  (from 11 and 16)
18.  $\neg P(y,z,v) \vee \neg P(y,v,w) \vee$   
 $P(e,z,w)$  (from 6 and 5)
19.  $\neg P(w,z,e) \vee P(e,z,w)$  (from 3 and 18)
20.  $P(e,f(b,c),a)$  (from 14 and 19)

Since 17 and 20 are manifestly contradictory, the proof is complete.

#### Discussion

In order to study the importance of the various strategies discussed above, Example 2 was

run without the aid of the *unit preference strategy* or the *set of support strategy*. At the end of 30 minutes the program had just finished generating  $S_2$ , having also generated  $S_1$ , without obtaining a proof. By comparison, with the aid of these strategies, the proof was obtained in about .5 seconds.

Example 3 serves to illustrate the comparative difficulty encountered when attacking a mathematical theorem without *a priori* knowledge, such as that employed in Example 2, as to which subset of the basic axioms of the theory is relevant. The proof of Example 2 was completed in about .5 seconds, while Example 3 required 54 seconds although the same strategy was applied to both. It should be noted that, without the *set of support strategy*, Example 3 was beyond the range of a 65,000 word machine.

#### BIBLIOGRAPHY

1. DAVIS, M., and PUTNAM, H., "A computing procedure for quantification theory," J. ACM 7, 201-215 (1960).
2. GILMORE, P. C., "A proof method for quantification theory," IBM J. Res. Develop. 4, 28-35 (1960).
3. ROBINSON, J. A., "A machine oriented logic based on the resolution principle." To be published.
4. ROBINSON, J. A., "GAMMA I, a general theorem-proving program for the IBM 704," Argonne National Laboratory Report ANL-6447. November, 1961.
5. QUINE, W. V. O., *Methods of Logic*, (Holt, Rinehart, and Winston, New York, 1959), Revised Ed.



# COMMENTS ON LEARNING AND ADAPTIVE MACHINES FOR PATTERN RECOGNITION\*

*C. Hugh Mays*

## I INTRODUCTION

By learning and adaptive machines I mean special purpose machines with internal components having adjustable values. The use of such machines for the solution of several kinds of problems has been proposed. These include the design of switching functions, the design of classification machines (classification machines is meant to imply recognition, prediction, decision and classification machines), automatic manufacturing of certain devices, self optimization of decision machines with time variable input statistics, improving the reliability of digital processes, and automatic wiring and testing of microcomponents. Most of the work to date has been on using learning machines to design switching functions and classification machines.

Learning machines will be useful only if they can solve significant problems faster, cheaper or with greater programming ease than conventional machines. The recommendations and comments in this paper are predominantly concerned with the question of the potential advantages of learning and adaptive machine approaches.

In a learning machine, samples of data are used as inputs to the machine. Other inputs indicate the desired machine response to the samples. Values of internal components are changed in accordance with some algorithm until the actual response is equal to the desired response.

## II PROBLEM STATEMENT

### A. *Data Classification*

Data classification and processing problems are becoming so complex that new techniques are required if these problems are to be solved economically. I will be mainly concerned with data classification problems in the following material, (i.e., problems requiring that a decision or prediction be made on the basis of given data and performance criteria).

For Example:

- (1) A decision to buy or sell a particular stock must be made.
- (2) The state of the weather in one week must be predicted.
- (3) A blood sample must be classified to determine if a patient has a given disease.
- (4) Directions must be given to the pilots of several airplanes when their locations, velocities and directions of flight are known.

### B. *Availability of Data*

In designing a classification system we may have a complete mathematical and statistical description of the physical process generating the data (e.g., certain control systems) or we may have samples of the data and a relatively incomplete mathematical and statistical description (e.g., a weather system). The kinds of information that may be available when a data

---

\* This work was done at Stanford University, Stanford, California.

classification system is to be designed include the following:

- (1) A complete or partial mathematical and statistical description.
- (2) Many samples of data with a clear indication of the class represented by each sample.
- (3) Few samples of data with a clear indication of the class of each sample.
- (4) Many samples of data without a clear indication of the class of each sample.
- (5) Few samples without a clear indication of class of each sample.
- (6) A problem with time variable characteristics.
- (7) Any reasonable combination of the above items.

In the following material most of the discussion will be concerned with items 1, 2, 3 and any reasonable combination of these items. The discussion will assume that the decision and prediction problems are to be solved by a completely mechanized system.

### C. Mechanization of Classification Processes

There are two aspects to data classification. These are 1) the design of the classification mechanism and 2) the problem of what to measure and how to preprocess the measurements for presentation to a classification system.

The main techniques used for the design of classification mechanisms are logical design<sup>1</sup>, the use of decision theory<sup>2</sup>, the use of learning or adaptive machines<sup>3, 4</sup> and the use of heuristic programs.<sup>24</sup> Logical design can be used when a process is not subject to noise. Decision theory is used when a process is noisy and knowledge about the statistics of the data generated by the process is available. Decision theory is presently limited to cases where the statistics are relatively simple. The most significant results have been obtained for Gaussian statistics.

This paper contains very little material on the use of heuristic programming approaches; Minsky's paper<sup>24</sup> contains a number of references to this subject.

The problem of what to measure and how to preprocess the results of the measurements for use in a classification machine is not very well

understood. Some work has been done on finding measurements that give invariant results for certain transformations (e.g., rotations and translations) of planar figures. I know of no work that has been completed for arbitrary transformations.

Learning machines are special purpose machines used to determine the values of components of a classification system. The learning machine may also be used as an element of a classification system rather than as a computer to determine numeric solutions. Presently most learning machines use networks of adaptive threshold elements.

## III THRESHOLD ELEMENTS AND ADAPTIVE THRESHOLD ELEMENTS

### A. Fixed Elements

In a threshold element a weighted sum of input variables is formed<sup>5, 6</sup>. If this sum is greater than or equal to a threshold value, the output of the element is a logical one; otherwise the output is a logical zero. In an adaptive threshold element the weight values are adjustable by adaptation circuitry.

Geometrically, the threshold element implements a hyperplane through the space of input variables<sup>7, 8</sup>. In the case where the input variables represent data from two classes, the classes are called linearly separable if there exists a set of weights and a threshold such that the response of the threshold element is a logical one for inputs representing one class and a logical zero for inputs representing the other class.

The hyperplane represents an optimum separating surface for certain kinds of statistical problems (e.g., two classes with Gaussian distributions and equal covariance matrices<sup>9</sup>). In these problems it is impossible to do a perfect job of separating the classes. Therefore the hyperplane must be placed so as to minimize the expected cost of misclassifying samples. For statistical problems having  $k$  classes with equal covariance matrices, we may require  $k(k-1)/2$  threshold elements followed by a decoding network. If the analog value of the weighted sum minus the threshold is available for manipulation, we need only  $k-1$  summing elements plus circuits to compare the analog

values. I have no idea which scheme is cheaper. For a few classes, a classification mechanism with only threshold elements is probably cheaper. As more classes are added the use of comparison circuits in conjunction with summing elements is probably cheaper.

Networks of threshold elements can be used to approximate other separating surfaces (e.g., quadratic and higher order surfaces). If we allow the use of analog multipliers (AND gates in the case of binary inputs), threshold elements can be used to exactly realize quadratic and higher order surfaces.

Since threshold elements can realize AND, OR, and NOT functions, networks of these elements can be used to realize any switching function or logical decision rule. There has been a considerable amount of work on the synthesis of switching functions with threshold elements<sup>5, 6, 10</sup>.

### B. Adaptive Elements

Algorithms for changing the weight values of a single adaptive threshold element are very well understood<sup>3, 11-13</sup>. I know of only one case<sup>25</sup> where these algorithms tend to produce an optimum separating hyperplane; and this is for a relatively simple statistical case (two classes, equal covariance matrices, equal a priori probabilities and the probability of misclassification errors is to be minimized). Algorithms for nontrivial networks of adaptive threshold elements are not very well understood. (Nontrivial means that the desired response does not uniquely determine the response of every threshold element.) I know of proposed algorithms<sup>12</sup> for such networks but of no satisfactory proof that the algorithms will cause convergence to a solution in a finite amount of time. However, experimental results indicate that these algorithms cause convergence most of the time.

The most successful adaptive machines have used very simple networks of adaptive threshold elements. In these machines the algorithms for changing weight values can be interpreted in terms of surface searching methods<sup>26</sup>. With these methods the weight values are always changed in a direction that moves the weights closer to a solution. It appears that nontrivial networks of adaptive threshold elements have

local minimum (at least for the networks so far considered). If surface searching methods are to be used, work needs to be done on finding ways to adapt nontrivial networks so as to avoid local minimum.

As the number of inputs to threshold elements is raised, the tolerance requirements on weight values get continually tighter<sup>14, 15</sup>, if the full power of the elements is to be realized. This means that very precise components with small drift values must be used. One way around this problem is to use more threshold elements. This introduces redundancy into any classification mechanism and increases the cost. The cost increase caused by redundancy may be more than offset by the use of less precise components.

Threshold elements appear to be very useful and, at least conceptually, they are simple units. Realization of threshold elements with hardware has included the use of Kirchoff adders, (at Stanford), magnetic cores<sup>16, 17</sup>, summing amplifiers (at Stanford) and resistor bridge networks<sup>18</sup>. Fixed threshold logic can be implemented easily but adaptive threshold elements place restrictions on the phenomena that can be used. For example, it would be difficult to make an adaptive threshold element that used the number of turns on a magnetic core to control weight values. The two most successful adaptive components have been a variable resistance device<sup>18</sup> and a magnetic device that uses remanent flux states to control weight values<sup>17</sup>.

## IV USE OF LEARNING MACHINES FOR DESIGN PURPOSES

### A. The Experimental Approach

In a classification machine, a sample of data is used as an input to the machine and some response to this input is obtained. For example the input might be a photograph of a cancer smear and the response a diagnosis of the smear. In a learning machine a sample of data is used as an input together with an input indicating the desired machine response. The internal structure of the machine is then modified according to some algorithm so that the actual response is equal to the desired response. One sample or several samples may be used as inputs at any one time. Several machines have

been built (Adaline and Madaline at Stanford, Perceptron at Cornell) that use only one sample at a time as an input. As far as I know, no one has built a learning machine to take more than one sample as an input at one time. There is no reason why such a machine could not be built and there may be advantages to do so, particularly if the samples are correlated.

Most experiments with learning machines have been performed as follows. Samples of data together with the classifications of the samples are stored in a conventional memory. The samples and their classifications are then picked one at a time to be used as inputs to a learning machine. This process continues until the response of the machine to each sample agrees with the desired response; the machine is said to have converged at this point. It may be necessary to use each sample as an input more than one time before convergence is obtained. After convergence is obtained the machine is tested on new samples. If the machine makes very few misclassifications of new samples, the experiment is considered to be successful and the machine is said to have generalized.

The experimental approach outlined above has been used for determining machines for weather prediction<sup>19</sup>, speech recognition<sup>20</sup>, classification of radar returns (no published reports) and classification of electrocardiograms<sup>21</sup>. These experiments were performed using digital computer simulations of the learning machine. These experiments appear to be very successful but a fair amount of work went into determining what measurements to take and how to preprocess them. In evaluating results it is difficult to separate the effects of preprocessing from the learning machine approach.

The experimental approach described above is used as a mechanism for understanding learning machines. Each experiment adds a little to our understanding. These experiments may show that our approach has been too naive and that we must understand a great deal more about the physical process we are experimenting with. On the other hand, the learning machine approach may provide a technique for exploring physical processes. For example the weather experiment described above showed that large weight values in the learning ma-

chine corresponded to geographic locations that the weather man found to be useful in predicting the weather.

### B. *Generality, Generalizing and Optimum Machines*

In order to handle many different classification problems, learning machines must be flexible; in other words, the machine must be able to realize many different mappings between inputs and outputs. I call this property generality. If the machine has great generality, training on a set of samples may imply very little about the response of the machine to new samples; in other words, the machine may not generalize very well. It appears that if generalizing is to be obtained, either the generality of the machine must be limited or the algorithms for changing the internal structure must be tailored to the problem being solved. The relationship between machine generality and algorithms on the one hand and generalizing on the other is not very well understood.

The question of optimum classifying mechanisms has been mentioned above in connection with certain problems handled by decision theory. With the one exception mentioned above, I know of no case where a learning machine gives an optimum classifying system. If optimization criteria can be defined for particular classification problems and a way can be found to implement these criteria in the learning machine, the problem of generalizing may be fairly well solved. There has been work on using analog computers<sup>22</sup> for solving certain linear programming problems. This work is similar to recent work on learning machines. Results using analog computers were never very successful because of accuracy limitations; however, the approach may indicate ways to enter optimization criteria into learning machines.

If a learning machine is trained on a limited number of samples the machine may converge to a very non-optimum solution because the amount of data is not sufficient to define an optimum solution. Sometimes a limited amount of data may be accompanied by a partial mathematical and statistical description of the process generating the data. I know of no general

method for using the mathematical and statistical information as an input to the learning machine. If a method can be found to use all available information as inputs to learning machines, a limited number of samples may not be such a great handicap.

The above discussion indicates that a learning machine may require a rather large instruction set. This instruction set may have peculiar characteristics when compared to the instructions for a conventional digital computer. For example, an instruction used to enter statistical characteristics of data may require that the machine rewire its components. Of course this could be done by use of logical gates.

### C. *Preprocessing*

If a learning machine has limited generality, its performance is very dependent on what measurements are taken and how these measurements are preprocessed for presentation to the machine. In the weather prediction experiments mentioned above, the input information was barometric pressure and change of pressure over a 24 hour period at several geographic locations. When the information presented was changed to the actual pressures 24 hours apart, the performance of the solution found by the learning machine was much worse than that found using the same information presented in the form of pressure and change of pressure. The preprocessing of the data had a considerable effect on performance even though the information content was not changed.

In other cases, the preprocessing of data may determine if a learning machine of limited generality will converge to a solution at all. The designer of a classification system may not be able to use a learning machine without a considerable knowledge of the process generating the data and of the limitations of the machine. The designer's work may be to determine how to preprocess the data so that the learning machine will converge to a satisfactory solution as determined by cost criteria.

### D. *Ease of Use and Speed*

One of the possible advantages of using a learning machine for design of classification systems is ease of programming. The idea of using samples of data and desired responses as

inputs and then letting the machine grind away until convergence is obtained is relatively simple. It should be possible to design learning machines that can be easily programmed.

If a long time is required to converge with a learning machine, it might be cheaper to use a general purpose digital computer. On the other hand if the learning machine can be made cheaply, a longer time to solve a problem may not be too much of a disadvantage. The question of speed is essentially a hardware question and it is too early to make any comments on the relative speed of digital computers and learning machines (which may be special purpose digital computers).

## V OTHER USES FOR LEARNING AND ADAPTIVE MACHINES

The above discussion was mainly concerned with using learning machines for the solution of conventional problems. Problems exist for which machines similar to learning machines might be used as permanent parts of operating systems.

Possible other uses for adaptive machines include the following:

- (1) Manufacturing classification machines
- (2) Compensation for drift and failure of machine components
- (3) Optimization of machines having inputs with unknown time variations.

The idea of using the concepts of learning machines in manufacturing processes is appealing. A single kind of adaptive unit produced in large quantities could be trained to perform many different functions by presenting different inputs and desired responses to different units. A practical question is, Does the decreased cost of producing identical units more than offset the increased cost of adaptive units?

It may turn out that some sort of adaptive capability will be helpful in manufacturing devices using microcomponents. Two of the major problems with microcomponents appear to be the problem of interconnections and the problem of not obtaining 100% yields. Perhaps some sort of adaptive scheme can be formulated to provide automatic interconnection of good components and rejection of bad components.

The use of redundancy and adaptation to increase the reliability and lifetime of digital systems has been proposed<sup>23</sup>. I know of no practical system that uses adaptation for compensation of changes in component values, but the idea appears to be workable. Redundancy implies that all working component are not being used to their full capabilities. In some instances it may be desirable to use adaptive techniques to assure that all components are being used to their full capabilities.

One of the more exciting possibilities for adaptive techniques is their use for self-optimization of a classification machine as the statistics of the inputs change. For example, in a speech recognition system, a machine that performs well on one speaker can be found relatively easily; however, a machine that will perform well on many different speakers is more difficult to find. With an adaptive speech recognizer, the system could be optimized for individual speakers. Other examples include a communication channel with time variable noise characteristics and a piece of equipment that generates data with an average that changes as the equipment ages.

In Section IV A above a procedure for use of a learning machine was described that required a conventional memory for the storage of data. In some applications it may be necessary to start classifying inputs before all data is available; as more data becomes available it should be possible to design a better classification machine.

It may be possible to design adaptive machines that can have their performance updated as more data becomes available without having to store all past data in a conventional memory.

A concept evolving in the field of learning machines with time variable input statistics has been called learning without a teacher. In the learning machines described above, a desired response was supplied with each training sample. In learning without a teacher, no desired response is supplied; the values of internal components are changed by rules that depend on the response of the machine rather than on desired responses. There has been relatively little work on this idea and I know of no published work.

## VI RECOMMENDATIONS

The basic problem in the learning machine field is to determine if learning and adaptive machine approaches offer cost, speed or ease of programming advantages over more conventional approaches. Before the solution to this problem is obtained, several theoretical and technological problems need to be explored. Specific recommendations are given in the following paragraphs:

- (1) A search to see if there are significant problems that can be solved by learning machine approaches should be made.
- (2) The properties and limitations of threshold elements should be thoroughly understood. A side benefit from this item may be efficient methods of synthesis of switching functions with threshold elements.
- (3) Algorithms for adjusting the weights and thresholds of networks of threshold elements should be found. This item will require research into the structure of such networks as well as into algorithms for adjusting weights.
- (4) The use of something other than threshold elements in learning machines should be investigated. I have no specific technique to suggest.
- (5) Some way of entering optimization criteria into a learning machine should be found. Work done on using analog computers to solve linear programming problems may give a clue on how to do this.
- (6) Some way of entering partial mathematical descriptions into a learning machine should be found. The problem with this and the previous item is finding a way to enter many different kinds of information into a single machine.
- (7) The instruction set for the learning machine and the set's relationship to machine structure and algorithms should be investigated. Some instructions may require manual or automatic rewiring of the machine structure.
- (8) The relationship of generality to machine structure should be investigated.



- (9) The relationships of generality, machine structure, and algorithms to generalizing should be investigated.
- (10) Problems of measurements and preprocessing as related to learning machines should be investigated. It may not be possible to come up with broad conclusions on this item. Measurements and preprocessing may depend so intimately on the process generating data that only conclusions for specific processes can be formulated. Probably this item can be best approached by studying specific processes and trying to generalize the results.
- (11) The question of what to do with a learning machine when only a small sample size is available should be investigated.
- (12) The question of tolerance requirements for the components of a learning machine should be investigated.
- (13) Questions concerning speed and cost are predominantly technological questions. To answer these questions the cost and speed of components to implement the findings of the investigation outlined above should be determined. The need for special purpose devices should be investigated.
- (14) The concept of learning without a teacher should be investigated.
- (15) The use of adaptive techniques in manufacturing processes should be investigated.
- (16) Techniques for updating the performance of learning and adaptive machines as more data and information become available should be investigated. The problem here is to find updating procedures that do not require that all past input data be stored in a conventional memory.

C. Hugh Mays  
IBM, Poughkeepsie, New York

#### REFERENCES

1. S. H. CALDWELL, "Switching Circuits and Logical Design," John Wiley & Sons, Inc., New York, N. Y.; 1959.
2. D. BLACKWELL and M. A. GIRSHICK, "Theory of Games and Statistical Decisions," John Wiley & Son, Inc., New York, N. Y.; 1954.
3. F. ROSENBLATT, "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms," Spartan Books, Washington, D. C.; 1962.
4. B. WIDROW and J. B. ANGELL, "Reliable, Trainable Networks for Computing and Control," Aerospace Engineering, Vol. 21, No. 9, September 1962, pp. 78-123.
5. R. O. WINDER, "Single Stage Threshold Logic," Proc. First Annual Symposium on Switching Circuit Theory and Logical Design; October 1960.
6. S. MUROGA, I. IODA, and S. TAKASU, "Theory of Majority Decision Elements," Journal of the Franklin Institute, Vol. 271, pp. 376-418; May 1961.
7. R. L. MATTSON, Master's Thesis, "The Design and Analysis of an Adaptive System for Statistical Classification," M. I. T., Cambridge, Mass.; May 22, 1959.
8. R. L. MATTSON, "A Self-organizing Binary System," Proc. EJCC, pp. 212-217; 1959.
9. G. S. SEBESTYEN, "Decision-Making Processes in Pattern Recognition," MacMillan Co., New York, N. Y.; 1962.
10. P. M. LEWIS, II and C. L. COATES, "Realization of Logical Functions by a Network of Threshold Components with Specified Sensitivity," IEEE Transactions on Electronic Computers, Vol. EC-12, No. 5, pp. 443-454; October 1963.
11. B. WIDROW, "Adaptive Sampled-Data Systems," Stanford Electronics Laboratories, TR No. 2104-1, Stanford University, Stanford, California; July 15, 1960.
12. W. C. RIDGEWAY, III, "An Adaptive Logic System with Generalizing Properties," Stanford Electronics Laboratories, TR No. 1556-1, Stanford University, Stanford, California; April, 1962.
13. C. H. MAYS, "Adaptive Threshold Logic," Stanford Electronics Laboratories, TR No. 1557-1, Stanford University, Stanford, California; April, 1963.

14. E. GOTO, "Threshold, Majority, and Bilateral Switching Devices," in *Switching Theory in Space Technology*, edited by H. A. Ken and J. W. F. Main, Stanford University Press, Stanford, California; pp. 47-67; 1963.
15. C. H. MAYS, "Adaptive Threshold Logic" in *Papers on Adaptive Systems*, compiled by B. Widrow and C. F. Franklin, Stanford Electronics Laboratories, TR No. 2104-2, Stanford University, Stanford, California, pp. 33-47; May, 1962.
16. M. KARNAUGH, "Pulse-Switching Circuits Using Magnetic Cores," *Proc. IRE*; May, 1955.
17. H. S. CRAFTS, "Components That Learn and How to Use Them," *Electronics*, Vol. 36, pp. 49-53; March, 1963.
18. M. E. HOFF, JR., "Learning Phenomena in Networks of Adaptive Switching Circuits," Stanford Electronic Laboratories, TR No. 1554-1, Stanford University, Stanford, California; July, 1962.
19. M. J. HU, "A Trainable Weather-Forecasting System," Stanford Electronics Laboratories, TR No. 6759-1, Stanford University, Stanford, California; June, 1963.
20. L. R. TALBERT, et al, "A Real-Time Adaptive Speech-Recognition System," Stanford Electronics Laboratories, TR No. 6760-1, Stanford University, Stanford, California; May, 1963.
21. D. F. SPECHT, "Vectocardiographic Diagnosis Utilizing Adaptive Pattern Recognition Techniques," Engineer's Thesis, Stanford U, to be published.
22. G. A. and T. M. KORN, "Electronic Analog Computers," 2nd edition, McGraw-Hill Book Company, Inc., New York, pp. 147-151; 1956.
23. W. H. PIERCE, "Adaptive Vote Takers Improve the Use of Redundancy," in *Redundancy Techniques for Computer Systems*, edited by R. H. Wilcox and W. C. Mann, Spartan Books, 1962.
24. MARVIN MINSKY, "Steps Toward Artificial Intelligence," *Proceedings of the IRE*, Vol. 49, No. 1, pp 8-30, January 1964.
25. G. F. GRONER, "Statistical Analysis of Adaptive Linear Classifier," Doctoral Dissertation, Stanford University, Stanford, California; April, 1964.
26. J. S. KOFORD, "Adaptive Pattern Dichotomization," Stanford Electronics Laboratories, TR No. 6759-1, Stanford University, Stanford, California; April, 1964.

# FLODAC—A PURE FLUID DIGITAL COMPUTER

*R. S. Gluskin, M. Jacoby, and T. D. Reader*  
*UNIVAC*

*A Division of the Sperry Rand Corporation*  
*Blue Bell, Pennsylvania*

## INTRODUCTION

The use of fluids (both liquids and gases) for the transmission and amplification of power has been common for over a century. This power has been controlled by valves, pistons, and other mechanical parts. Within the past decade considerable attention has been given, both in this country and in Russia,<sup>1</sup> to the use of fluids for control and logic functions, and until recently these systems also employed mechanical moving parts. In 1960 the Diamond Ordnance Fuze Laboratory (now the Harry Diamond Laboratories) of the U. S. Army announced a fluid amplifier with no moving parts<sup>2</sup>—a discovery which seems likely to revolutionize the whole field of fluid logic and control.

There are two fundamental mechanisms involved in pure fluid amplification: One is momentum transfer, and the other is wall effect.

A jet of fluid having a density  $\rho$ , a velocity  $V$ , and a cross sectional area  $A$  has a momentum  $M_1 = \rho V^2 A$ . Consider now a smaller jet, of momentum  $M_2$ , which we shall call the control jet, impinging at right angles on the larger jet, of momentum  $M_1$ , which we shall call the power jet, as is shown in Figure 1. From conservation of momentum principles it can be seen that the power jet will be deflected through an angle  $\theta = \tan^{-1} \frac{M_2}{M_1}$ . If a receiver or catcher is placed as shown in Figure 1 all the energy of the power jet can, in principle at least, be recaptured if there is zero control jet flow. But as

the control jet flow is increased the power jet will be deflected, and less and less energy will enter the receiver.

The wall effect is shown in Figure 2. If a wall is placed close to a jet it is found that the jet appears to be attracted to the wall and will often attach itself to the wall quite strongly. The reason for this is that as the jet moves it entrains fluid from the surrounding medium. This entrained fluid must be made up by fluid from afar. If a wall is placed close to one side of a jet, the flow of replacement fluid is impeded, resulting in a slightly lower pressure on the side of the jet close to the wall than on the other side where there is no impediment. Consequently, the jet will bend toward the wall,

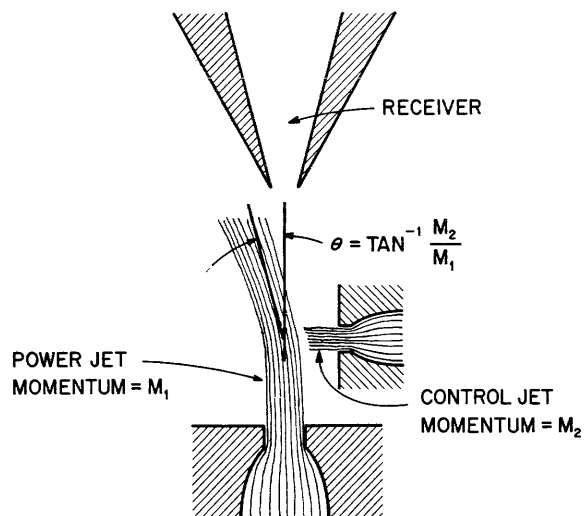


Figure 1. Momentum Exchange.

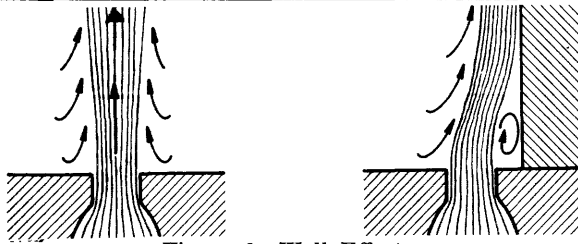


Figure 2. Wall Effect.

making it even harder for replacement fluid to flow into the low-pressure region, and eventually lock onto the wall altogether, forming what is called a low-pressure bubble upstream from the point of attachment. This wall effect is sometimes called the Coanda effect.<sup>3</sup> Both wall effect and momentum transfer are usually involved in most fluid amplifiers.

The Harry Diamond Laboratories' amplifier, shown in Figure 3, consists of a power jet, A, an interaction region, B, two control jets, C and D, and two output passages, E and F.

The amplifier may be made to operate either proportionally or in a digital fashion, depending on slight differences in geometry. In the proportional device, fluid issuing from jet A will divide almost equally between output passages E and F in the absence of any control signals at C or D; but if a small amount of fluid is blown into the device through control port C, the main jet will be deflected to the right, and more of it will exit from passage F than from

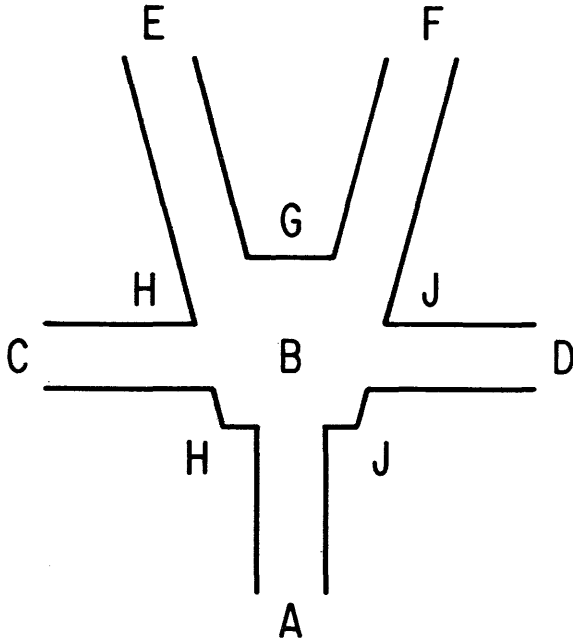


Figure 3. Representation of Amplifier Developed at Harry Diamond Laboratories.

passage E. For small signals, the output variation can be made fairly linear with respect to the input, and furthermore, the output variation will be larger than the input variation, thus producing gain or amplification.

If certain changes are made in the geometry of the proportional amplifier, namely, if the divider tip G is moved further downstream and if the walls HH and JJ are moved closer to the centerline of symmetry, the amplifier becomes bistable or digital. That is, the power jet bends by itself and attaches to one or the other of the sidewalls so that even in the absence of any control signal, substantially all the flow is from a single output passage, for example, passage E.

Assuming an output from passage E, if the pressure at control port C is now slowly increased, no change occurs until a certain threshold value is reached, at which point the power jet suddenly switches to the other side and exits from output passage F. If the control signal is returned to zero, the power jet remains locked to wall JJ and continues to issue from output passage F. Thus, the device has memory capability and is the logical equivalent of the electronic flip-flop.

All the other fundamental switching and logic functions which are now performed electronically can also be implemented by pure fluid devices. Figure 4 shows a pure fluid inverter. This device is purposely made very asymmetrical so that in the absence of a control signal the fluid flows essentially in a straight line and exits from the left-hand leg. Only when a control signal of sufficient strength is present will the jet be blown over to the right-hand leg. The wall lock-on effect on the right-hand leg is minimized, and when the control signal disappears the output will immediately return to the left-hand leg. Thus, the output from the left-hand leg is the inverse of the control signal; the output from the right-hand leg is the control signal amplified.

An OR gate may be readily constructed by converting the pressure energy in two or more signals to velocity energy and directing this into a common receiver, as is shown in Figure 5. Because of the vector quality of the two jets, there will be very little leakage of signal from

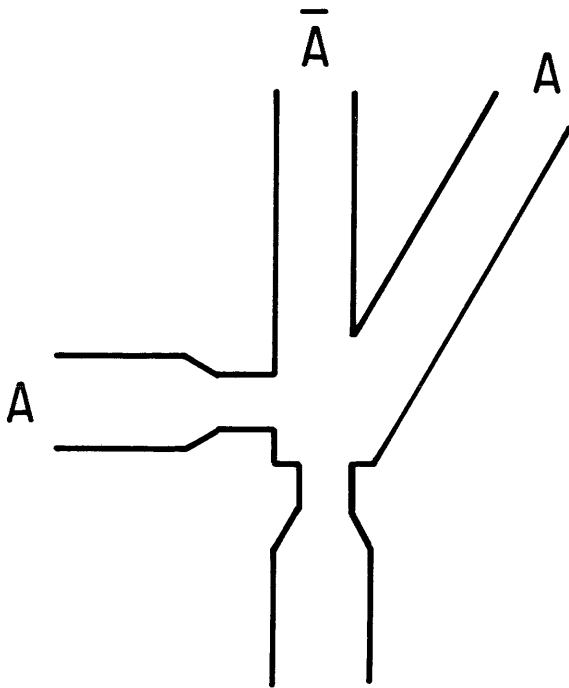


Figure 4. Inverter.

one input backward through the other unless the output is substantially blocked, in which case the fluid would have no place else to go. The purpose of the two side passages shown in Figure 5 is to provide for fluid escape in just this eventuality. If a high impedance is presented to the output the fluid will escape through the side bleeds rather than reversing and flowing into the other signal input.

The vector properties of high-velocity jets are also made use of in the AND gate (Figure 6). If the signal A is present alone it will pass straight through the AND gate and vent to atmosphere, and similarly for B alone. If signals A and B are present simultaneously, however, and have approximately equal amplitudes, then the resulting jet will make an angle of 45° to the original jets and will be caught in the receiver placed at this position. With proper design of the receiver exact balance of the two signals is not necessary. Ratios of 2 : 1 are easily accommodated.

The combination of the OR gate and the inverter as shown in Figure 7 gives an extremely powerful element. The right-hand output is the amplified OR function of the inputs while simultaneously the left-hand output gives the

amplified NOR function, that is,  $\bar{A} \cdot \bar{B}$ . This principle has been extended to achieve a NOR gate with a fan-in and fan-out of four; that is, the device has been provided with four input terminals. The presence of a signal on any one of these terminals will switch the device off, and the output has been divided and channeled to four output terminals so that four identical elements can be driven from one element. In this NOR gate the OR output is not provided.

As is well known, all logic and switching functions can be implemented with NOR gates alone. A flip-flop, for example, can be made by interconnecting two NOR elements, as is shown in Figure 8.

In this configuration, one element is in the 1 state, and its output is used to switch the other element into the 0 state. The elements remain stable in their respective states until an outside signal changes the state of the flip-flop. With reference to Figure 8, assume that element A is in the 1 state, that is, its output signal is in leg d. This signal is sent to element B through input j which switches the element into the 0 state, that is, the jet is diverted to

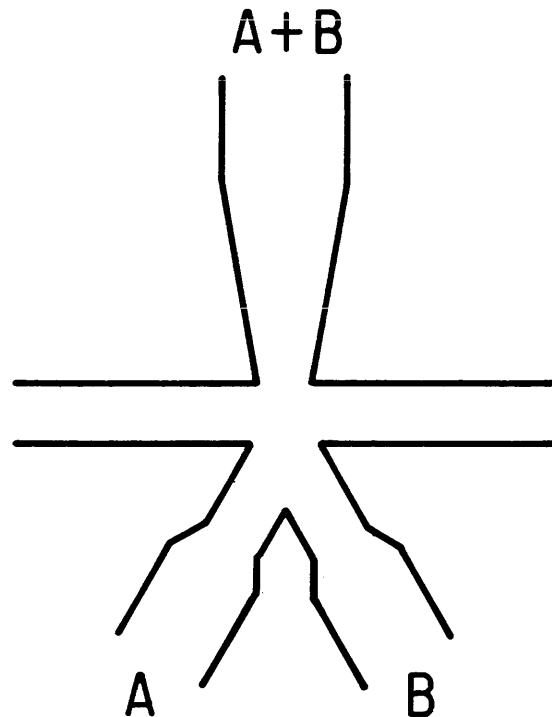


Figure 5. OR Gate.

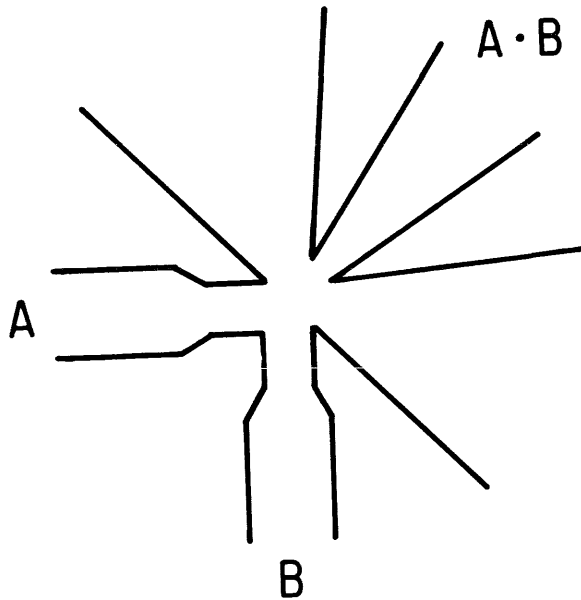


Figure 6. AND Gate.

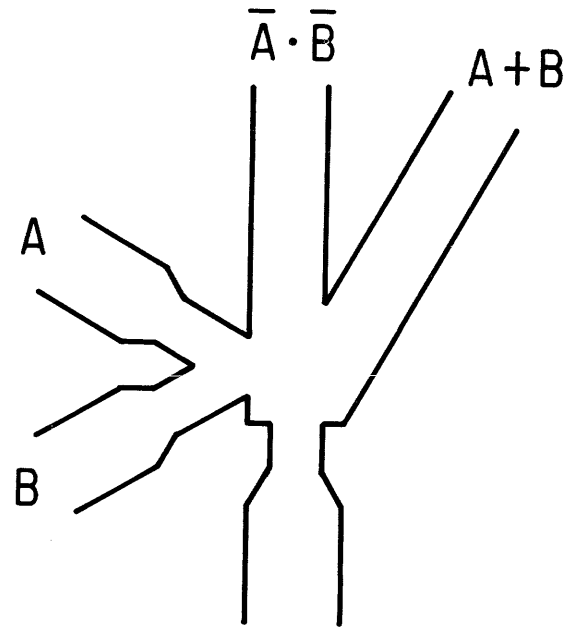


Figure 7. NOR Gate.

output f. There is no signal in leg e of element B; therefore, element A is not affected by element B. The flip-flop will remain stable in this state until an outside signal is applied to element A by way of input h. When the signal appears, A will be switched to the 0 state, and the jet will be diverted from leg d to leg c. The signal to input j of B will go off, and the element will switch from the 0 state to the 1 state, that is, the jet will switch from leg f to leg e. The signal from B will then be applied to A through input g. When the outside signal through h is removed, the signal through g will keep A in the 0 state. The flip-flop has been switched, and it will remain stable in this state until another outside signal is applied to input k of element B.

FLODAC is built entirely of NOR gates and requires about 250 of these elements to do its job. FLODAC, incidentally, stands for Fluid Operated Digital Automatic Computer.

#### Why Fluid Amplifiers?

Granting the feasibility of constructing complex digital systems from fluid amplifiers, what is the motivation for doing so? What advantages, if any, do fluid amplifiers have over their well-established electronic counterparts? The use of fluid amplifiers rather than their electronic counterparts may be justified on the

basis of four significant advantages: reliability, environmental immunity, low cost, and absence of r-f radiation. Each of these advantages is briefly discussed below:

1. *Reliability:* Pure fluid amplifiers have no moving parts except the fluid itself. There is nothing to wear out, nothing to age, nothing to burn out. With the proper selection of structural material and fluid, there are no potential chemical or solid-

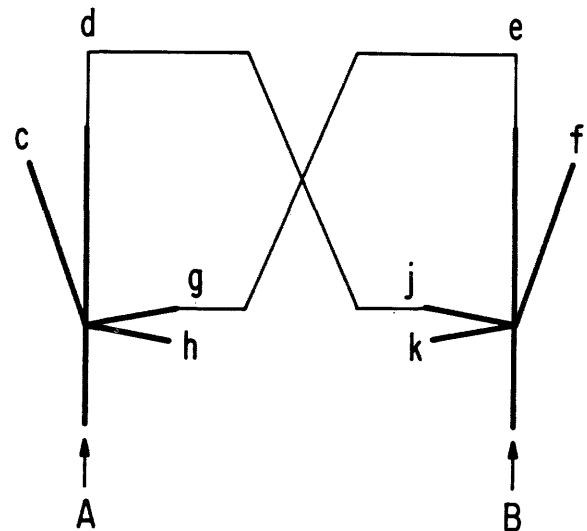


Figure 8. FLODAC Flip-Flop.

state reactions. There need be no delicate structures. In short, the life of a fluid amplifier should be practically infinite, whether in use or quiescent. About the only conceivable cause of deterioration would be dirt in the fluid, and this can be controlled easily by filtration and the use of closed-cycle systems.

The fluid amplifier art is still too young for masses of statistical data on reliability to have been compiled. However, the nature and operation of these devices are such that extremely favorable comparisons with electronic and other types of devices can be expected.<sup>4</sup>

2. *Environmental Immunity:* Fluid amplifiers can be made of almost any solid material, for example, plastics, metals, glass, or ceramics. If the right materials are selected, operation is possible under environmental conditions which preclude the use of electronic devices. For example, ceramic fluid amplifiers could operate at white heat. Metal fluid amplifiers should be operable in intense radiation fields. With the proper materials and assembly procedures, shocks, or accelerations, of thousands of G's should present no problem.
3. *Low Cost:* Fluid amplifiers consist basically of rectangular channels in a suitable material. They can be fabricated by any one of a number of extremely low-cost methods, such as casting, injection molding, stamping, or etching. Entire circuits of fluid amplifiers, including the interconnecting passageways, can be formed by such methods in one low-cost operation. Planes could be stacked one on top of another with holes in the planes at the proper locations for the necessary interconnections. With techniques such as these, which are already being developed, it is estimated that the cost of fluid amplifier circuits may be as much as 100 times less than the cost of comparable electronic circuits.
4. *R-f Radiation:* No electromagnetic energy is radiated by fluid circuits; consequently, a very common and often serious problem associated with electronic logic, namely,

r-f radiation, is eliminated. Often extensive and costly measures must be taken to shield electronic equipment to prevent interference with other equipment or communications or to prevent detection of radiated intelligence by hostile agencies. However, such measures are never 100 percent effective. Also, external radiation can seriously affect electronic equipment by causing errors or malfunctions. Both of these problems are completely eliminated by fluid devices, which neither emit nor are affected by radiation.

#### *Operational Speed*

Fluid amplifiers have one significant disadvantage: their operational speed is relatively slow. Switching times are of the order of a millisecond, and signal propagation time is of the order of a millisecond per foot. Fluid amplifiers, at present, are only approaching kilocycle rates of operation as opposed to the megacycle and higher rates common in electronic systems. Speeds can be expected to improve, of course, but nanosecond switching times are not foreseeable today. Because of this speed limitation, there may be applications where fluid amplifiers are not suitable.

It should be noted that the inherent speed limitation can be offset appreciably by taking advantage of the low cost and high reliability of fluid amplifiers, which make it economical to compensate for much of the speed deficit by making extensive use of parallel and polymorphic operation.

In any event, there are numerous applications where the speed of fluid circuitry is adequate. Today's system designer must realize both the merits and shortcomings of these new elements; by appraising his problem requirements objectively, he can use fluid amplifiers with excellent results wherever their advantages enable them to do the job better and/or more economically.

#### SPECIFICATIONS AND LOGICAL DESIGN

Every general-purpose digital computer must have means for accomplishing four basic functions: Memory, Arithmetic, Control, and Input/Output. Consequently, it was necessary to provide these functions if we were to fully meet the goal of demonstrating a generalized fluid com-

puter, even though on a very small scale. All four functions are fully and formally developed in FLODAC.

The problems of memory size, word size, and instruction set are all interrelated. The objective was to build a very small air-powered general-purpose digital computer which could be programmed to do a few elementary problems. At least three instructions seemed necessary to prove generality: an Arithmetic instruction, a Data Transfer instruction, and a Conditional Jump instruction. Since two bits are needed to specify three instructions, a fourth instruction becomes possible without any increase in word size. It was decided to make this a Halt instruction. The four instructions used then are:

Instruction Code Explanation

Transfer	T m	10	(A) → m
Add	A m	11	(m) + (A) → A
Jump	J n	01	Go to instruction in memory location n if (A) ≠ 0; otherwise continue with the next instruction in the memory.
Halt	H	00	Stop the computer.

After a few trials it was found that a reasonable program could be written using all four instructions with a memory of only *four words*. This program, which is the basic test program for FLODAC, will be discussed later.

A four-word memory implies two bits for addressing, and this, with the two bit operation code, fixes the word length at four bits. The first two bits of a word are the operation code and the last two the address. Alternately, a word may consist of numerical data only.

To recapitulate, FLODAC has four instructions and four words of memory; each word is four bits long. To compensate as much as possible for the speed disadvantage of fluid elements, operation is bit parallel.

Figure 9 shows the overall block diagram of FLODAC. Each instruction is processed in four steps by the step counter which is driven from the master clock. The control counter contains the address of the next instruction to

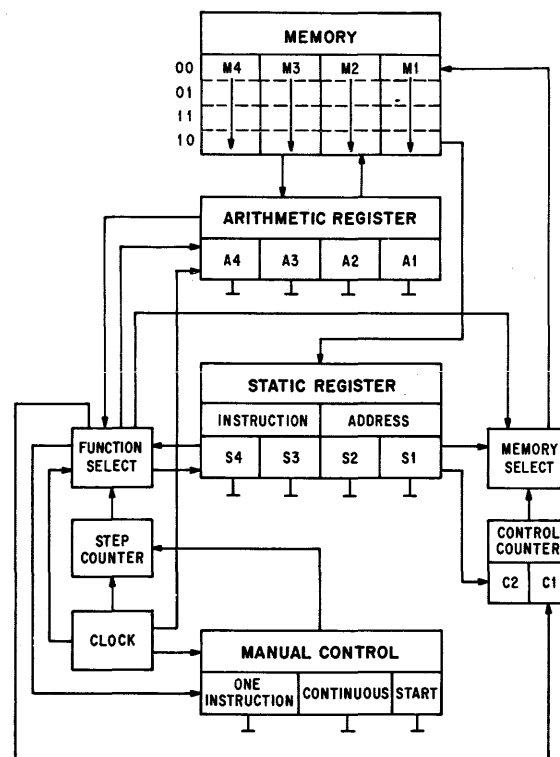


Figure 9. FLODAC, Block Diagram.

be executed and, except during the fulfillment of the Conditional Jump instruction, is augmented by one at the end of each instruction cycle. During step 1 the control counter operates the memory select circuits, and by the end of step 1 the specified memory word, which contains the next instruction, is read into the static register. The two left-hand bits are decoded as to operation, and this information is sent to the function select circuits, where, in conjunction with step counter and clock signals, the necessary gating pulses for all instructions are generated. The two right-hand bits, specifying the operand address, are sent to the memory select circuits, permitting the required data word to be read out. All this takes place during step 1. The actual instruction execution is carried out during some or all of the last three steps.

The Add instruction is carried out in two stages. The first stage is completed during step 3 and consists of adding the word in the memory to the word in the arithmetic register without regard to carry. This portion of the Add instruction changes a bit in the arithmetic register from a 1 to 0 or vice versa, whenever there



is a 1 in the corresponding position of the word to be added. The second stage of the addition process starts at the beginning of step 4 and causes a carry pulse to be sent to the next more significant bit position wherever the sum bit is presently a zero and the addend bit was a one. Means are provided to rapidly transmit the carries to subsequent stages if the original carry pulse would in turn produce another carry at the next higher bit position. This carry generation and propagation proceeds asynchronously and could continue during steps 4 and 1, if necessary. It is actually completed by the end of step 4.

A brief description of the operation of the memory may be in order here. The memory is

a two-dimensional matrix array using flip-flops as the storage means.

Figure 10 is a schematic of the FLODAC memory. Each block represents a NOR element with four inputs and four outputs. Lines into the left side and bottom of the blocks indicate inputs to the elements, and lines from the right side and top represent outputs from the elements. The memory contains four words, each consisting of four bits of information. The words are located in vertical columns, the right column being the first or 00 word address. The horizontal rows contain the same information bit for all four words, the top row being the least significant bit.

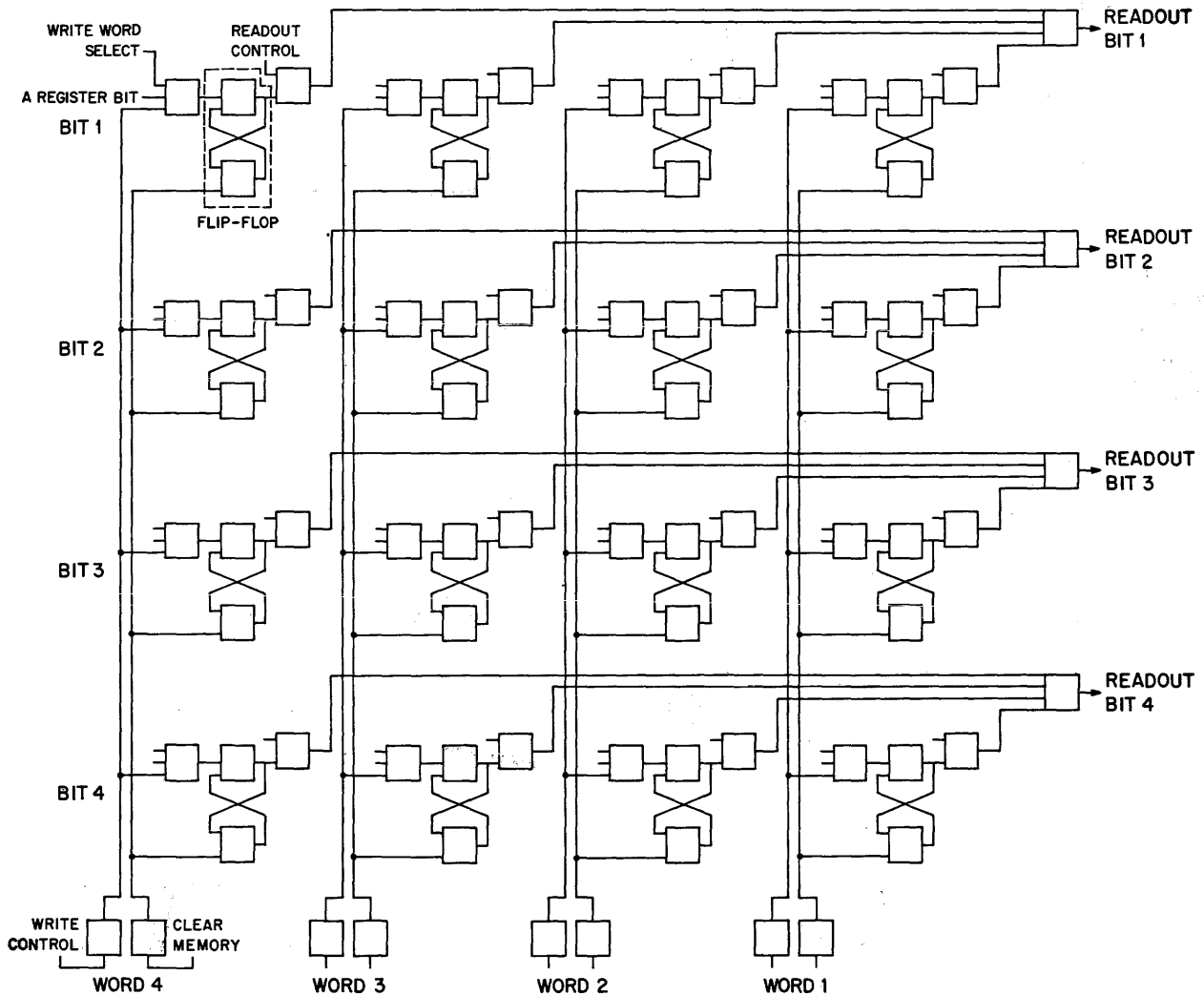


Figure 10. FLODAC Memory, Schematic Diagram.

Writing into memory is a three-step operation. First, the memory storage location is selected by the static register and the memory select circuits. Second, a clear memory signal from the function select erases any information already in that word location by resetting all flip-flops to zero. Third, the word in the accumulator is gated into the memory location by another function select signal. The write operation utilizes three counts of the step counter cycle.

A simple nondestructive readout is employed. Each of the 16 memory flip-flops feeds into an intermediate NOR element which in turn sends a signal to one of the four memory readout elements (far right in schematic). These intermediate elements are controlled by the memory select circuits. When a particular address is chosen for readout, the memory select turns off the intermediate elements of the other three words, thus prohibiting any output signals from these words. Parallel operation is used. Therefore, all bits of a word are written into and read out simultaneously.

Referring back to the overall block diagram, Figure 9, it is seen that a manual control unit is provided which makes possible two modes of operation: *continuous* and *one instruction*. To enter information into the memory of FLODAC the computer is placed in the *one instruction* mode. A word is then set up manually in the A register, and a Transfer instruction specifying the desired memory location of the word is set up in the static register. When the START switch, also located in the manual control unit, is activated, the word in the A register will be transferred into the memory. This process is repeated for each word to be loaded. It should be mentioned perhaps that the act of setting a switch consists merely in putting one's finger lightly over a bleed hole on the control console (Figure 11). The resulting back pressure will then set a flip-flop or generate a fluid pulse, as

the case may be. It is difficult to imagine a simpler form of keyboard. The contents of the A register and static register are displayed by using bipositional visual indicators. The indicators consist of a colored ball in a glass tube. The balls are lifted into view by a pressure signal when the flip-flops are in the 1 state.

## CONSTRUCTION AND TEST

### *Production of Logical Elements*

The NOR elements used in FLODAC were made by injection-molding a thermoplastic material into a metal negative master.

The physical size of the fluid devices is a function of the width of the power input nozzles. The widths of the nozzles used on the FLODAC elements were either 0.016 inch or 0.020 inch. These widths were chosen because they allowed the use of standard laboratory fabrication techniques and equipment and afforded accessible tolerances. Increased dimensional accuracy was obtained by machining the master from a large template five times normal size and reducing by means of a pantomill.

Reducing the size of the elements with the above fabrication method is limited by the size of the cutting tool used in making the master. A nozzle width of 0.005 inch might be attainable, but present photoetching processes are able to produce still smaller and more accurate models.

### *Testing the Individual Elements*

Because of dimensional variations occurring during fabrication, the characteristics of the elements sometimes differed. It was necessary, therefore, to set up a testing procedure to check out all elements before using them in the FLODAC circuits. Two testing criteria were chosen. The first was that the pressure recovery be at least a set minimum value. Pressure recovery is the ratio, expressed in percentage,

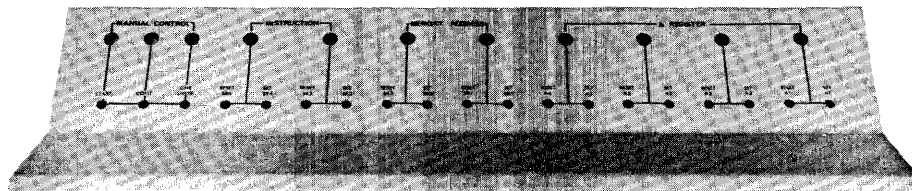


Figure 11. Control Console.

of the output pressure to the input supply pressure. At the present state-of-the-art, the NOR elements have a rather low pressure recovery factor. The FLODAC elements averaged about 30 percent, although other experimental models have reached almost 50 percent. Acceptability for FLODAC required a pressure recovery of at least 28 percent.

The second test for the elements was gain. This is the ratio of the element output to the input signal required to switch the element. A figure of 1.6 was chosen as the criterion here. The supply pressure for the elements in FLODAC was 20 inches of water (0.8 of a pound per square inch). This means that the output pressure would have to be at least five inches of water and that the elements would have to switch with less than three inches of water input signal pressure. Elements not meeting these specifications were rejected. All tests were made with the device loaded with the equivalent load of four other elements.

#### *Method of Assembly and Testing of Circuits*

To simplify construction and testing of FLODAC, the computer was divided into two parts. Each half consists of a power supply manifold and three rows of NOR elements. There is a total of 280 elements in FLODAC. The existing circuitry requires only 250 NOR elements, but the extra elements were added in case replacements had to be made or for possible changes or extensions to the logic. The NOR element power inputs are plugged directly into the manifold. Interconnection of the elements is done by simply connecting one of the four outputs of an element to one of the four inputs of the next logical element in the circuit. These connections were made with plastic tubing.

One side of the computer contains the clock, step counter, instruction portion of the static register, function table, and A register circuits; the other side contains the control counter, address portion of static register, memory select, and memory circuits.

The two halves were wired and tested independently. Simulated pressure signals were used where necessary when testing the circuits on each side. When both were working separately, the entire computer was assembled, all the cross connections between the two sections

were made, and appropriate outputs were connected to the control panel indicators. To facilitate maintenance, FLODAC was constructed so that one side hinges out, exposing all of the internal circuitry (Figure 12).

The entire system was then tested, and after straightening out a few minor problems in the circuitry, FLODAC was working reliably as an independent, coordinated system. Figure 13 is an overall view of the finished FLODAC assembly.

#### *Testing the Complete Computer*

After all the elements had been interconnected, the system was ready for a complete checkout. This was done by carrying out simple programs which required use of the four computer instructions: Add, Transfer, Jump, and Halt. Instructions were stored in the four-word, four-bit memory unit.

The FLODAC clock was capable of being pulsed manually so that a program could be carried out one step at a time and checked for correctness at every intermediate step. Pressure taps connected to indicator manometers were placed at critical points throughout the computer circuitry. This showed the state of the elements and greatly simplified troubleshooting.

When a program was working satisfactorily with manual clock control, the program was tried with automatic computer controls. In-

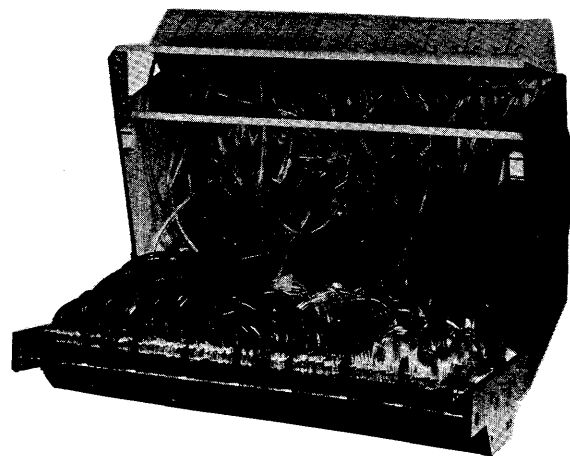


Figure 12. FLODAC Circuitry, Internal View.

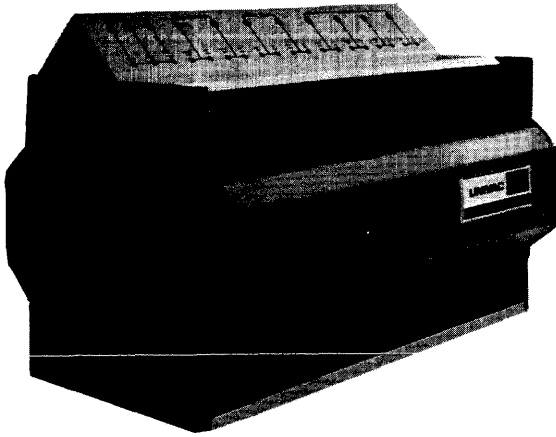


Figure 13. FLODAC.

formation was stored in the memory, the process was started by manually pressing the program START button, and the program was carried out automatically.

The most comprehensive test program makes use of all four instructions and is shown below as a matter of interest.

Memory Location	Instruction	Address	Explanation
1	A	4	$(4) + (A) \rightarrow A$
2	T	4	$(A) \rightarrow 4$
3	J	1	Go to location 1 if $(A) \neq 0000$ ; otherwise go to location 4.
4	xx	xx	Arbitrary number; or Halt instruction if 0000.

This program takes an arbitrary number (stored in memory location 4), adds it to the accumulator, transfers the sum back to memory location 4, tests to see if the number in the accumulator is now zero, and repeats this process automatically until the sum in the accumulator is zero. When this condition is reached, as it must be after at most five cycles, the Conditional Jump back to 1, stored in memory location 3, is not carried out, and the program moves on to memory location 4. The content of memory location 4 is now 0000, which is interpreted as a Halt instruction, and the computer stops. The above program, of course, doubles

the number in the accumulator on each cycle after the first.

Alternately, memory locations 2 and 3 can be interchanged, and in this case the accumulator will be augmented each cycle by the number initially stored in location 4 until it again reaches zero. In this form 16 cycles are possible.

## SUMMARY AND CONCLUSIONS

The individual reliability test given to each component NOR element greatly reduced the probability of encountering any serious problems during the final checkout of the system. A few minor logic and performance problems did show up, but these were easily traced and corrected. FLODAC was operating reliably only a few weeks after construction was begun.

The nominal clock rate of FLODAC is ten cycles per second. This was chosen to avoid wave propagation and reflection problems which are potential dangers at not very much higher frequencies because of the long lead lengths involved and the lack of attention given to exact impedance matching. It should be pointed out that the speed of fluid signal propagation in air is almost one million times slower than the speed of signal propagation in electric wires. Thus, from the point of view of signal wave length, a frequency of ten cycles per second using air as the working medium is analogous to a frequency of ten megacycles in electronics. Simpler circuits built with elements identical with those in FLODAC, but compactly packaged, have operated with clock frequencies as high as 250 cycles per second.

FLODAC has amply demonstrated that a pure fluid general-purpose digital computer is indeed feasible. The question remains: Is such a computer desirable? There are many areas in which fluid logic cannot hope to compete with electronic logic simply because of the speed limitations inherent in fluid systems even if the utmost advantage is taken of parallel and polymorphic operation.

Conversely, however, there are areas involving extreme environments, such as very high radiation levels or very high temperatures, where present-day electronics cannot hope to compete with fluids, and here fluid logic may

supply the only means of solving many pressing military and space science problems. Between these two extremes there is a vast area where fluid logic does appear to be competitive with electronics and where the advantages and disadvantages of both approaches will have to be carefully studied. This area includes such devices as adding machines, desk calculators, tabulating machines, process control computers, and such peripherals as keyboards and punched card and paper tape readers. Here, timing rates are frequently below a kilocycle, and the speed disadvantage of fluid amplifiers disappears. Here too, the tremendous cost advantage plus the postulated reliability advantage makes fluid logic look very attractive indeed. It may further develop that, given sufficient cost and reliability advantages, the marketplace may well learn to live with slower computing speeds for small general-purpose computers.

The authors believe that there is a vast role to be played by fluid technology in the computer field, and this view is shared by their company.

UNIVAC's FLODAC is the precursor of what we hope will be long series of useful pure fluid systems of increasing complexity and decreasing cost.

#### REFERENCES

1. AUGER, RAYMOND N., "Pneumatic Computer Research in the USSR," *AUTOMATIC CONTROL*, Vol. 13, No. 6 (December 1960), pp. 43-48.
2. Refer to parts A through F of the Diamond Ordnance Fuze Laboratories, "News Release" for March 2, 1960.
3. COANDA, H., "Precede et dispositif pour faire devier, une veine fluide penetrant autre fluids," Patent No. 788,140 (France), 217, 1934.
4. Fox, H. L., "A Comparison of the Reliability of Electronic Components and Pure Fluid Amplifiers," *Proceedings of the Fluid Amplification Symposium*, October 1962, Diamond Ordnance Fuze Laboratories, Washington 25, D. C.



# DESIGN AUTOMATION UTILIZING A MODIFIED POLISH NOTATION

*William K. Orr and James M. Spitze  
Friden, Incorporated  
San Leandro, California*

## INTRODUCTION

Our first objective in developing the Design Automation (DA) System described herein was to produce and maintain, using a digital computer, the manufacturing and field service documents for a small electronic calculator. However, as a long range objective we wanted a system capable of handling the documentation for virtually any digital computing device, and aiding in certain design functions. On the surface this appears to be the very task performed by existing DA systems but as it turns out the construction techniques used on the calculator give rise to problems not normally handled by these systems.

Because of the wide variance between existing and anticipated construction techniques, it was decided to use Boolean equations as the basic input. The equations are written in a modified form of Polish notation which enables one to very effectively relate the logic to the hardware for trouble shooting purposes. Along with the equations a description of each gate, flip-flop, etc. is input to the system. The form of these descriptions readily accommodates most types of hardware implementation.

This paper is based upon our experience in documenting the design of a small electronic calculator. We shall indicate, as we go, how the system is generalized to handle other digital computing devices.

## THE EC/130

To give the reader some idea as to the size of the device we set out to document, we mention here a few of the features of the Friden electronic calculator model EC/130. In addition to those features normally associated with desk calculators the EC/130 (cf. Figure 1) has a four high push down stack as well as auxiliary storage. All stack information is stored on a magnetostrictive sonic delay line and constantly displayed on the cathode ray tube shown in Figure 2.

The EC/130 is a low cost high production item on which great engineering effort has been expended to reduce size and cost to a minimum. As a result of this intense engineering effort the degree of standardization is minimal. No

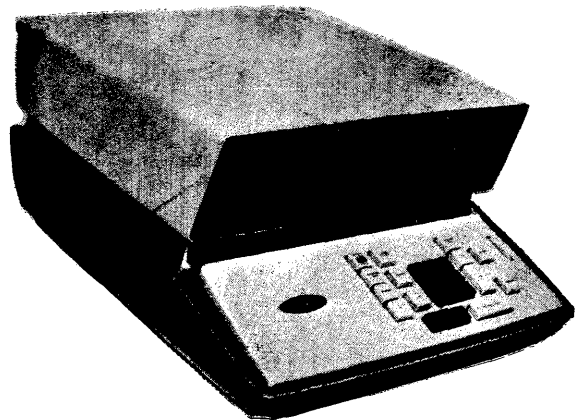


Figure 1. Friden Electronic Calculator Model EC/130.

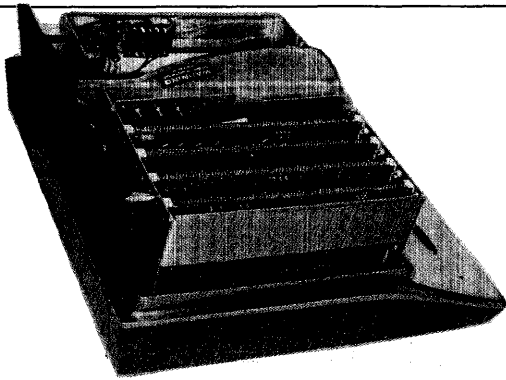


Figure 2. View of Calculator Packaging.

standard building blocks, such as flip-flop modules, are used. For a basic circuit type (i.e. amplifier, flip-flop, etc.) used  $n$  times, there are typically  $n/2$  different circuit designs.

The basic constructional unit in the EC/130 is a printed circuit card such as the one pictured in Figure 3. Each card contains from 400 to 500 components. For purposes of automatic insertion each lead of a component is located on a point of a .050 grid. Thus, each hole on a circuit card has an ordered pair of numbers ( $x,y$ ) associated with it giving its position relative to this fixed coordinate system.

To document the logic of the EC/130 and yet be able to handle other more complex systems, we needed a descriptive language which is hardware independent and yet can be related to any type of hardware implementation. For this reason we chose Boolean equations as our basic input. The particular format in which the equations are written was devised to facilitate the process of relating the logic to the hardware.

#### MODIFIED POLISH NOTATION

Any digital computing device is logically a set of interconnected "elements," where for our

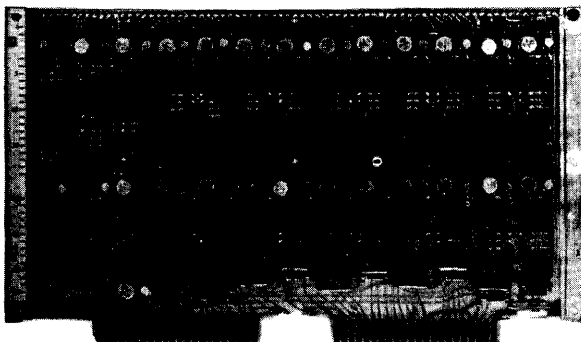


Figure 3. Typical Printed Circuit Card.

purpose we define an element as any combination of components which has a clearly definable function with the exception of AND gates, OR gates, and INVERTERS. For example, flip-flops and drivers are considered elements. When the elements of a computing device are combined, they are combined using AND and OR gates. Thus, all such combinations can be described using Boolean equations.<sup>1</sup> Boolean equations have been used by other manufacturers to describe equipment designs;<sup>2,3</sup> however, we chose not to use the standard Boolean notation for an equation written in this notation cannot be conveniently related to the corresponding hardware. There has also been some work done, on an experimental basis, using a combination of block diagram and Polish notation features as the descriptive language.<sup>4</sup> We avoided the use of block diagram features for their use tends to destroy the continuity of an equation line.

A characteristic of most logic block diagrams, such as the one in Figure 4, is that the inputs (operands) are always shown to the left of the associated gates (operators). A notation using this form of operand-operator ordering is the notation of Lukasiewicz,<sup>5</sup> the so-called Polish notation. In this notation the input to the three legged OR gate of Figure 4 would be written  $AB + C +$ .

In the usual Polish notation each operator is a binary or unary operator in the sense that it operates on not more than two operands. Let us consider the operators  $+$  (or) and  $\cdot$  (and) as not binary but  $n$ -ary. Then the expression  $AB + C +$  could be written  $ABC +$ . If we allow the scope of an operator to extend over  $n$  operands we must in some way delimit this scope. To delimit operator scope and also to enhance readability we use parentheses. With the scope of the operators extended and parentheses added to the notation we now write the equation equivalent to Figure 4.

$$I1 = (ABC +) (DE +) \cdot \quad (i)$$

Figure 5 illustrates how equation (i) is related to the hardware. The location of a given gate input (i.e. the point at which this input may be tested) is printed directly under the name or gate where the input signal originates. For ex-



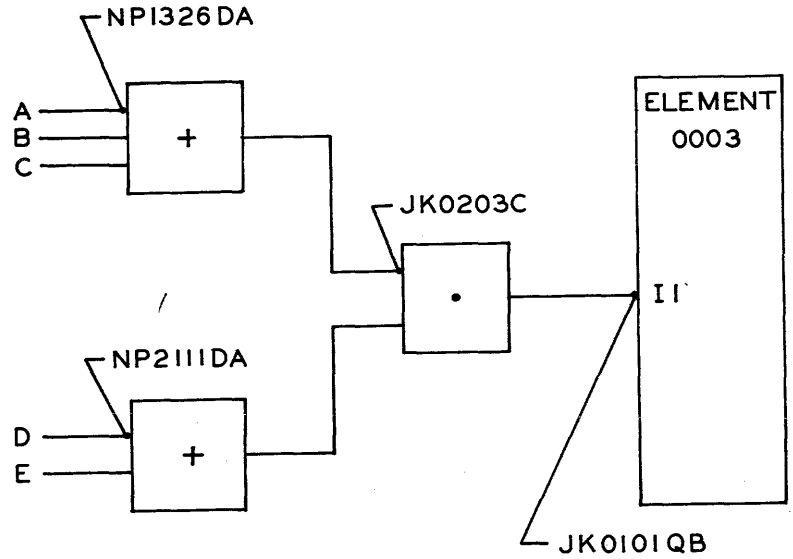


Figure 4. Logic Block Diagram.

ample, the quantity printed under the name A (A being the name of a particular element output), indicates where the signal, A, is connected to the OR gate (the reader should refer to Figure 4 where some of the location information has been indicated). The AND gate output is the input to an element hence the location of the element input is given in line 0003.002.00 rather than under the AND gate itself. The indication here is that the particular element input can be found on circuit card JK at (01,01), the component at JK0101 is a transistor (Q) and the lead of interest is the base (B) lead.

It is important to note that the continuity of equation (i) is completely retained in Figure 5, and the clear separation between the logic and hardware information enhances readability. Further, the format in which the location information is printed reflects the structure of the schematic of Figure 4.

A gate, G, is said to be at "logical level" j if there are j-1 gates interposed between G and

the named input signals. For example, the AND gate in Figure 4 is at logical level 2. The full importance of the logical level indication (cf. Figure 5) is clearly evident when an equation is longer than one line. Figure 6 serves to illustrate how equations of arbitrary length are handled and also supplies those details missing from Figure 5.

Figure 6 is typical of the form that an element description takes. Line 0132.00.00 is the heading line and contains the element name, the element function code and the latest element revision letter. Line 0132.01.00, contains the name and location of the element output. The gate reference line, 0132.02.12 contains the construction file reference (discussed in a later section) for each gate in the input equation. In the modified Polish notation there is a one-to-one correspondence between operator symbols and hardware gates. The correspondence between the symbols and the associated references is thus determined by the order in which the gates appear in the equation. The remarks

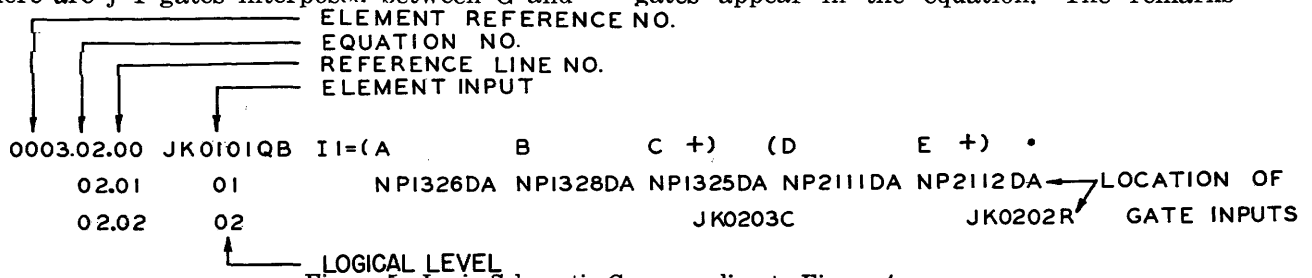


Figure 5. Logic Schematic Corresponding to Figure 4.

```

0132.00.00 EPCSTEP6 INV AA
01.00 JK0627QC OI= EPCSTEP6
02.00 JK0528R II= ((DPCTRO / DPSTORFN+) / COMDIGFN+) / DIVFF EPCTR4FN +
02.01 01 NP0611DA
02.02 02 NP0812DA NP1761DA
02.03 03 NP0809R
02.04 04 NP0412DA NP0411DA
02.05 05 NP0409R
02.06 06 NP0212DA NP0211DA NP0329DA
02.07 07 JKI710DA

(EPCTR2FN HOME * (NBIST NSHIL *) DPSFN +)4
02.08 01 JKI425R JK1009C JK2106R JK2207C
02.09 02 JK0336DA JK0529DA JK0632DA
02.10 03 JK0932R
02.11 07 JKI711DA

02.12 0050 0061 0070 0098 0099 0100 0101 0131 0120 0125 0126
02.13 OUTPUT FAILURE RESULTS IN STACK I SHIFT LEFT ERRORS

```

Figure 6. Computer Generated Logic Schematic.

line, 0132.02.13, is used to convey trouble shooting information to the service man.

In Figure 6 there are two lines with a logical level indication of 07. This implies that the inputs to the gate at level 07 are from elements or gates implied by two lines of the equation. In general, the logical level indicates the relationship between the location information associated with one line of an equation and that of another.

The information which the DA system uses to produce the logic schematic (cf. Figure 6) consists of the construction file and the input schematic pictured in Figure 7.

It may seem that we have gone out of our way to eliminate one of the most important features of the Polish notation, namely, that it is parenthesis free. However, keep in mind that at the outset we were after a notation that is convenient to use in design documentation. We have traded those features of the Polish notation which facilitate mechanical translation for features which add to the power of the notation for our use in documentation.

## RELATING THE LOGIC TO THE HARDWARE

It is an inherent requirement of the DA system that its output fulfill the documentation needs of both the production and field service

departments. At the outset of the system design it was assumed that these needs would be relatively compatible, i.e., that information generated in a format acceptable to one department could be effectively utilized by the alternate department with only a minor amount of reformatting. That this is *not* the case becomes evident when one considers the differing needs of the two departments. Whereas a repairman is interested in finding a circuit point as quickly as possible, the production line requires the precise location and size of each lead associated with a part for automatic drilling and part insertion.

Referring to the circuit card picture (Figure 3), a repairman can more easily locate a circuit point if he is given the location in terms of visual columns and rows (note the repairman's removable plastic scale on the left margin of the card) than if he is given the location in terms of the .050" co-ordinates used in the card production. To provide visual co-ordinates to the field service department and absolute .050" co-ordinates to the production department, an updatable visual-to-absolute transformation file is maintained as a portion of the part file. The part file also contains an updatable dimensional description of each part used in the EC/130.

Using the transformation file, the system performs automatic visual-to-absolute co-ordinate transformation as the need arises. Given the transformation file and dimensional informa-

```

013200 EPCSTEP6 INV
013201 OI= EPCSTEP6
013202 II= ((DPCTRO / DPSTORFN+) / COMDIGFN+) / DIVFF EPCTR4FN +
013202 (EPCTR2FN HOME * (NBIST NSHIL *) DPSFN +)4
013202 0050 0061 0070 0098 0099 0100 0101 0131 0120 0125 0126
013202 OUTPUT FAILURE RESULTS IN STACK I SHIFT LEFT ERRORS

```

Figure 7. Input Schematic From Which Figure 6 Is Generated.

tion for a particular part, only the visual location of one lead of the part and a one digit orientation code need be mentioned for the system to automatically determine the absolute location of all of the leads of the part, thus minimizing the input description of the EC/130 circuits as they appear in the construction file.

As an example, consider a transistor (Part No. 223297) on card NP with its base lead located with a relationship to visual column 17 and visual row 14 defined by an orientation code of 1. To completely locate this part, the system input is NP 17 14 1 223297 and the output is as sketched below:

NUMBER	NAME	LOCATION	LEAD	HOLE SIZE
223297	2N1499	NP 159-052	B	.026
		161-054	C	.026
		161-050	E	.026

The base lead (i.e. "key lead") is precisely located with reference to its nearest co-ordinate intersection by the orientation number which, through the transformation process, specifies the exact location of the transistor. For all parts other than transistors the orientation code indicates the relationship (up, down, right, left) of a part's second lead to its key lead.

Experience has shown the value of the two co-ordinate system described above. At an early stage in DA system development, only the .050" grid co-ordinates were used with all leads of a part being specified. Although the production departments could get their documents with ease, the field service repairmen and the writers of the input documents were tied to the same production oriented requirements with a negative effect on DA system acceptance. Recognizing the visual division of printed circuit card parts into rows and columns and the redundant nature of specifying more than a part's first lead when the part was not a varia-

ble size, the system was modified to handle the visual co-ordinate and key lead ideas. The immediate effect was a great improvement in DA system acceptance.

It should be stressed that the key lead concept can be easily extended by a simple change to the transformation programs to handle multi-lead modules of the type commonly found in large digital devices. The fact that the EC/130 is a very small electronic device should not obscure the more significant fact that the basic DA concept described herein is designed to be applicable to much larger systems.

#### THE CONSTRUCTION FILE

Recognizing the prior entry of a part file and co-ordinate transformation file, the first major system input is the construction file, an example of which is shown in Figure 8. NOTE: (1) Figure 8 is an example of the *printed output* of the construction file, (2) the revision code ("AF")

FRIDEN INC										PAGE 020.00AF	
EC/130 CONSTRUCTION FILE											
REV	REF-LINE	LOCATION	T	L	O	A	P	PART NUMBER	IN	PART	REMARKS
LTR	INSERT								OUT	DESCRIPTION	
	010.0	JK004 032	D	A	2			0223292	15	1N662	
	012.0	JK005 030	R				I	0223392	01	12K 5PC 1/2W	
AF	0132.000.0	JK005 028	R				I	0223403	11	6.8K 5PC 1/2W	HIGH SPEED INVERTER
	002.0	JK005 029	C				I	0804083		680PF 10PC 200V	(WATCH OUT FOR
	004.0	JK006 027	R		2		I	0804044		22K 5PC 1/2W	VALUE CHANGES IN
	006.0	JK006 027	Q	B			I	0223297		2N1305	680PF CAP.)
	007.0	JK006 027	Q	C					01		
	009.0	JK005 026	R				I	0223257		10K 5PC 1/2W	
AF	0133.000.0	JK006 023	D	A	2		I	0223292	11	1N662	FOUR INPUT OR GATE
	001.0		C				I				INPUT 2 IS HEAVILY
	002.0	JK006 024	D	A	2		I	0223292	12	1N662	LOADED

Figure 8. Construction File.

in the example for both page and circuit) is determined by the DA system, and (3) the "Part Description" information is obtained from the part file at print time and is not part of the construction file input.

The construction file is a sequential undatable listing of the various circuits used in a digital device, and requires as input: (1) the number and location of each part, (2) a code for each circuit input/output, (3) a file reference number (of the form: Circuit, Line, Insert digit) used for file searching, (4) any remarks pertinent to the construction of the circuit, and (5) several coded entries explained below. The combination of circuit reference number and I/O code is the search key used by the DA system to obtain the hardware location of the inputs and outputs of Logic Schematic elements, gates, and inverters. It should be noted that the input codes for a particular gate are determined by the order in which these inputs appear in the equation. Typically, only one construction file entry line is needed for complete specification of a part.

Any one of several non-zero entries in column A (absolute) permits entry of mixed (visual and/or .050") grid co-ordinate information for those parts whose shape does not permit them to be located in the normal relationship to the visual grid. Entries to columns T (part type) and L (lead code) give additional information about the circuit component and are used: (1) as sorting flags for the production oriented documents and (2) to assist the servicemen in accurately finding a given part. The Column O (orientation) entry describes the orientation of the part with reference to its key lead. Column P is used in EC/130 construction to indicate a plated hole in a circuit card. A normal part entry requires no entries to Columns O, A or P.

The construction file occupies a central position in this DA system. It is the source file for the input/output location information appearing in the Logic Schematic printout, and for generation of the various production oriented documents: from the Construction File, control information is produced for automatic insertion machines and automatic drilling machines; and by-product documents, such as bills-of-material, are produced as requested by various company departments.

#### ADDITIONAL FIELD SERVICE DOCUMENTS

Besides the Logic Schematic there are two other field service documents: a glossary of terms and a logic block diagram. The Logic Block Diagram is manually drawn and shows how all the elements of the Logic Schematic are interconnected to form the EC/130 system. Groups of gates (represented as Boolean equations in the Logic Schematic) are shown as single blocks referenced to the appropriate Logic Schematic equation. The Logic Block Diagram is used as a field service reference and training aid.

The Logic Schematic File is the major data source for the Glossary of Terms. A list of (1) the names used in the Logic Schematic Boolean equations and (2) the names of each Logic Schematic element output is generated. This list (containing the names and associated element numbers) and a card deck giving prose definitions of the various names are used to construct the Glossary of Terms.

The four columns of the Glossary of Terms (Figure 9) are defined as follows: Column #1, Boolean variable name; Column #2, element-equation having this output name (if the name is used in an equation but does not appear as an element output name, the words "EXT. INPUT," i.e., "External Input," appear in this

FRIDEN INC EC/130 GLOSSARY OF TERMS				PAGE 07.00AF
TERM	DEFINED AT	USED IN EQUATIONS		DEFINITION
ENTERKEY	EXT. INPUT	0182.02	0183.02	ENTER KEY
EPCSTEP6	0132.01	0087.05	0091.06 0161.04	ENTRY PHASE COUNTER STEP 6 OUTPUT INVERTER
		0185.02	0254.02 0306.06	FAILURE WILL RESULT IN STACK I SHIFT LEFT
		0307.05	0308.05 0309.05	ERRORS ENTRY NG. ADD OK. SUB OK. MULT NG. DIV
		0310.05		NG
EPCSTPIN	0142.01	0183.02	0185.02 0232.02	ENTRY PHASE COUNTER FLIP-FLOP I RESET
		0236.02	0249.02 0254.02	FAILURE WILL RESULT IN STACK SHIFT UP ERRORS
		0284.04	0284.06	ENTRY NG (MAYBE). SUB NG. MULT NG. DIV NG

Figure 9. Glossary of Terms.

column); Column #3, element-equation numbers where this variable is used; Column #4, prose variable definition.

The Glossary of Terms is the repairman's entry point to the field service documents. In many cases, a brief examination of a digital computing device will indicate its major malfunction. For the EC/130, once this is done, the Glossary of Terms indicates those elements affected by the particular problem and, from this point, reference is made to the Logic Schematic and, if necessary, to the Construction File. Note that the variable names are carefully chosen to be as closely related to their function and as easily understood as possible.

### ENGINEERING CHANGES

An important reason for the existence of a DA system is to decrease the time lag between the decision to make a product change and the carrying out of this decision. The DA system described herein permits rapid additions, changes or deletions to all the files: part file, card co-ordinate transformation file, construction file, logic schematic file, glossary, etc. Any change to one file is automatically carried through the system and reflected as necessary in all subsequently maintained files. As an example of the checking involved in the carrying through of changes, before the Logic Schematic is printed, an automatic check is made to determine if the hardware implied by the Logic Schematic has been properly specified in the other system files.

In the case of the many paged major output documents (i.e. Construction File, Logic Schematic, Glossary of Terms) only the changed pages are printed. These files have associated with them an internally updatable page control file giving page limits and revision codes (for each page). These files are built during the first generation of their related printout, with the page limits being set to waste as little paper as possible while trying to prevent their parent file's basic unit of information (i.e. construction file, circuit; logic schematic, equation; glossary, name) from spreading from one page to the next.

A problem of intrinsic interest is that of adding to (deleting from) a page other than

the last one of a file. At first glance this would push down (up) all subsequent pages causing a potentially excessive printout of mostly unchanged information. To avoid this problem, each page is looked upon as an extendable unit. Additions to a page cause its being printed on more than one sheet of paper with page indications of the form: Page 011.00AB, Page 011.01AB, etc. The page revision letter is updated whenever a page is changed thus precluding the existence of dissimilar pages with the same page designation.

### SYSTEM SUMMARY

The DA system described herein is comprised of seventeen computer programs. Figure 10 gives an overall view of the system. The card inputs, on the left of the figure, are the changes (additions, deletions, corrections) to the various files. The outputs, on the right of the figure, will occur only when required by a change. Immediately preceding the printing of the Part List, Construction File, or Logic Schematic there can occur an error listing indicating all detectable improper file changes.

### CONCLUDING REMARKS

The system we have discussed is most properly a system for Mechanized Design Documentation. However, in view of the fact that logical equations are our basic input we can provide various forms of design assistance, such as, logic simulation, load analysis and some of the more exotic things such as module placement and cost approximation.

One of the unmentioned merits of the system discussed is that it was very simple to implement. The system was designed and the necessary computer programs written for a Honeywell 400 computer in six months. This made it feasible to do the documentation for a machine as small as the EC/130.

The initial reaction of new personnel to the DA system and especially to the logical equation notation was one of mild horror. However, as they began to use the system, this attitude disappeared. In fact, it is now felt that the Logic Schematic is a more useful aid to the trouble shooter than the conventional circuit schematic normally supplied.

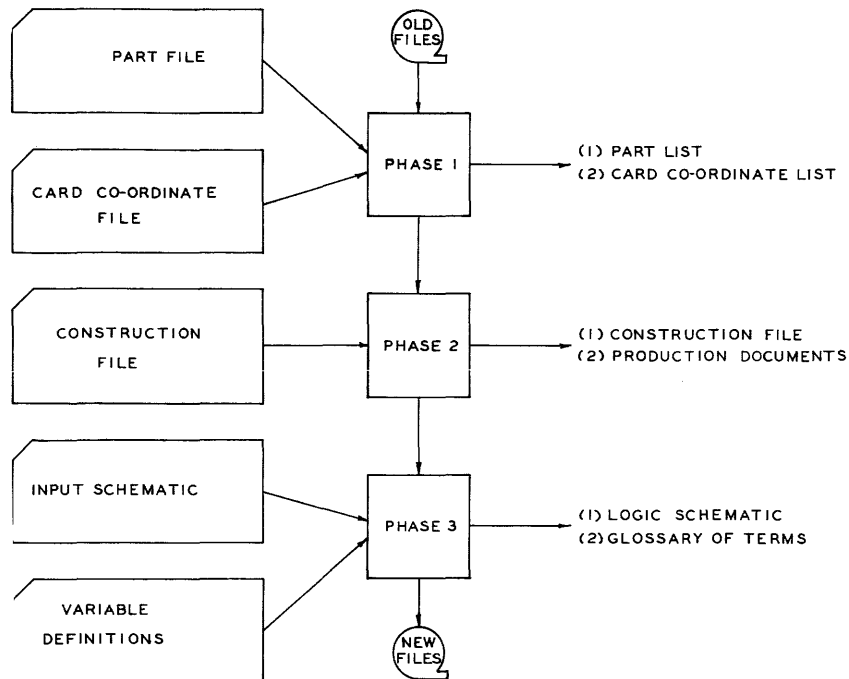


Figure 10. DA System Diagram.

#### ACKNOWLEDGMENTS

We are indebted to Messrs. L. P. Robinson and G. H. Hare for their help and encouragement throughout the development and implementation of this system. We would like also to express our appreciation to Messrs. A. M. Gould and R. A. Ragen who were responsible for the design and documentation of the EC/130.

#### REFERENCES

1. PHISTER, MONTGOMERY, JR.: Logical Design of Digital Computers (1959). (This is a valuable reference for the reader who is not familiar with the use of Boolean equations in logical design).
2. GORDON, W. L.: Data Processing Techniques in Design Automation. Proc. Eastern Joint Computer Conference (1960), 205.
3. HANNIG, W. A., and T. L. MAYES: Impact of Automation on Digital Computer Design. Proc. Eastern Joint Computer Conference (1960), 211.
4. PREISS, R. J.: An Experimental System for Logic Design Data Accumulation and Retrieval. Proc. IFIP Congress (1962).
5. LUKASIEWICZ, J.: "Aristotle's Syllogistic from the Standpoint of Modern Formal Logic." Oxford, 1951.

# SYSTEMATIC DESIGN OF CRYOGENIC LOGIC CIRCUITS\*

*C. C. Yang and J. T. Lou  
Computer Sciences Laboratory  
The Technological Institute  
Northwestern University  
Evanston, Illinois*

## INTRODUCTION

In recent years considerable interest has been manifested in the fields of information storage and retrieval and pattern recognition. An important element in the design of information-retrieval systems and pattern recognition devices is associative memory 5, 10-13, 15. An associative memory is particularly suitable for search of a record from a large file, the sorting of data into an ordered list 9, 10, 12 and the identification of patterns. Cryotron circuits are important components in the construction of an associative memory system. Despite this importance and though cryotron technology has been actively developed for some years, it is still in its infancy.

An associative memory is designed to perform simultaneous comparisons of all stored information with the interrogating bits. Thus, in an associative memory system the memory cells should be able to read and write and the switching circuits should be able to perform comparisons.

Cryotrons have been recognized as the elements most suited to the construction of such a system. They are ranked among the most challenging components in the digital computers to be designed.

In an associative memory system switching circuits play as important a role as the memory

cells. This paper is concerned with the systematic design of cryotron logic circuits for such storage systems. No constraints are imposed upon the logic functions to be synthesized. However, emphasis is placed upon three-terminal cryotron gate networks to realize complementary logic functions.

## USE OF COMPLEMENTARY LOGIC FUNCTIONS

The design of cryotron logic circuits discussed here involves the realization of complementary logic functions. In a cryotron circuit each single gate has a resistance of the order of 0.01 ohm in the resistive state, and zero resistance in the superconducting state. If the resistive state is identified as "0" of a binary variable and the superconducting state is designated as "1," the two states are rather difficult to distinguish from each other because of the small difference in resistance which corresponds to a small difference in current or voltage of the circuit. To improve the reliability of cryotron-circuit operation, use of the complementary logic functions are proposed.

An elementary cryotron circuit consists of two separate branches representing a state and its complement. If one branch is superconducting, the other is resistive. The superconducting path allows the entire current to pass through;

\* The work reported here was supported in part by the National Science Foundation and the Office of Naval Research.

the resistive branch blocks the flow of the current. Thus the superconducting branch, or non-zero current condition, designates the "1" bit; and the resistive branch, or zero current condition, designates the "0" bit. This logic circuit can be designed systematically by using a serial-parallel branch which realizes a given logic function and another branch which realizes the complement of that function.

An alternate method of designing a cryotron logic circuit is to synthesize the given logic function by using a controlled by-pass branch. Whenever the main branch becomes superconducting, the by-pass branch is changed to resistive or is disconnected from the circuit by a selection and control circuit. Conversely, when the main branch becomes resistive, no matter how small the resistance may be, the by-pass branch is made superconducting to allow the entire current to flow. This method may require a smaller number of cryotrons but the design is complicated by the selection and control circuits.

This paper is primarily concerned with the systematic synthesis of cryotron circuits to realize the complementary logic functions. Each cryotron circuit is split into two parts: the control circuit and the gate network. Each gate in association with a relevant control is an integral circuit element which forms a cryotron.

### CRYOTRON SWITCHING CIRCUITS VS. MULTITERMINAL RELAY NETWORKS

In view of the analogies between cryotrons and relays as well as their respective switching circuits, the techniques originally developed for the synthesis of transfer-contact relay networks may be applied to synthesize a three-terminal cryotron gate network with complementary outputs, or to synthesize a two-terminal cryotron gate network with a by-pass branch and the necessary selection and control circuit. Resembling the multiterminal relay network illustrated in Fig. 1, a cryotron switching circuit may be characterized by a block diagram as shown in Fig. 2. Their similarities and differences are discussed below.

Each relay contact has two stable states: "open" or "closed." The "closed" state may be identified as a "1" and the "open" state may

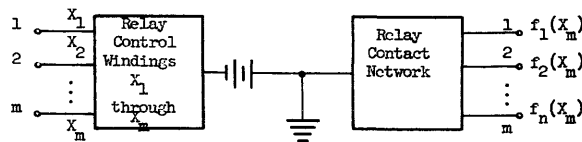


Figure 1. Block Diagram of a Multiterminal Relay Switching Circuit.

be designated as a "0". Each cryotron gate is in one of its two stable states: the superconducting state or the resistive state, which may be designated as "1" and "0," respectively. The state of the relay contact is changed by application of a sufficient current to its relevant control winding. Similarly, the superconductive-to-resistive (or vice versa) state-transition of the gate is accomplished by passing an appropriate current through its associated cryotron control. In both cases the state transition is entirely independent of the direction of the control current, but dependent on its magnitude.

A single relay winding may control several pairs of relay contacts, each pair of which is known as a transfer-contact. One contact is normally open and the other normally closed. Each cryogenic switching device, as developed in this paper, may control a number of gate-pairs, each pair having the same variable assignment. The maximum allowable number of cryotron controls in a single switching device is limited by the resultant inductance of the serial cryotron controls in each load circuit of this device. On a single relay a number of transfer-contacts may be used, so long as the maximum allowable number of springs is not exceeded.

For convenience of analysis a multiterminal relay switching circuit may be split into two parts: the control winding arrangement and the contact network as shown in Fig. 1. A cryo-

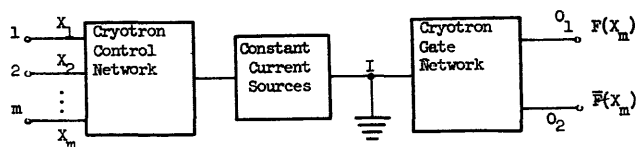


Figure 2. Block Diagram of a Cryotron Switching Circuit with Complementary Output Functions.



tron switching circuit may likewise be split into two separate networks as shown in Fig. 2.

There are, however, several differences between them. While each transfer-contact with its complementary variable assignment  $X_i$  and  $\bar{X}_i$  is controlled by a single relay winding, each pair of gates assigned by  $X_i$  and  $\bar{X}_i$  is generally controlled by its respective cryotron controls; but the current which passes through one of the two controls at a time is determined by a single cryogenic switching device. Thus the complementary gate-pairs may be considered equivalent to the transfer-contacts and the switching devices equivalent to the relay windings.

While the variables  $X_i$  and  $\bar{X}_i$  are assigned, respectively, to the normally open and normally closed contacts of a transfer-contact relay, their truth values are:  $X_i = 0$  and  $\bar{X}_i = 1$  when the relay is not operated. These values are changed to

$$X_i = 1 \quad \text{and} \quad \bar{X}_i = 0$$

when the relay winding is excited by sufficient current, at which time the relay is operated. The variables  $X_i$  and  $\bar{X}_i$  are assigned to two separate cryotrons. One cryotron gate which is represented by  $X_i$  is considered to be normally superconducting without critical or over-excitation, i.e., the current passing through its control is below the critical value for gate state-transition, or even zero. The other, normally resistive gate, is assigned by the variable  $\bar{X}_i$ . The resistive state is maintained by the application of a sufficiently large current in the control. The change of the variable truth value is accomplished by switching a current from one control circuit to the other, or vice versa. For example, the normally resistive gate which represents the variable  $X_i = 0$  is changed to  $X_i = 1$  when its control current is transferred to the other cryotron control to make the latter cryotron gate resistive, i.e.,  $\bar{X}_i = 0$ . In other words, two cryotrons each of which includes a "control" and a "gate" are needed for the selection of the variables  $X_i$  and  $\bar{X}_i$ ; but in the relay circuit a single transfer contact relay winding can obtain this result.

Although the cryotron gate network, like the relay contact network, may have  $n$  outputs each of which fulfills a different logic function among  $f_1(X_m), f_2(X_m), \dots, f_n(X_m)$ , where

$m$  indicates the total number of distinct complementary variable-pairs, the series-parallel connection of only two outputs which represent the complementary functions  $F(X_m)$  and  $\bar{F}(X_m)$  is generally adopted, as previously described. By taking advantage of these existing similarities several well-developed synthesizing methods—particularly the techniques of cascading three-terminal subnetworks with complementary outputs, the tree configuration, and the symmetric network<sup>4</sup>—may be employed for the realization of logic functions by means of cryotron circuits.

### CRYOTRON CONTROL NETWORK

A cryotron control network for use in controlling the gate states follows. This network consists of  $m$  elements if the gate network is composed of  $m$  distinct complementary gate-pairs. Each element here is called a control current switching device, shown in Fig. 3. Its structural skeleton was originally proposed by Buckingham<sup>3</sup> and later studied by Vail and his co-workers.<sup>16</sup> However, its mode of operation as herein proposed is entirely different.

As here conceived, each of these devices is used for controlling a pair of complementary cryotrons which are labeled  $X_i$  and  $\bar{X}_i$  and shown in Fig. 3. Each device, however, may control several gate-pairs if all of them are assigned by the same Boolean variables. The terminals  $P_1$  and  $P_2$  of the primary loop are connected to a current source of pulses or even sine waves. The terminals  $S_1$  and  $S_2$  of the secondary loop are connected to a constant direct current source  $I_d$ . Suppose that the primary current  $I_p$  is a positive pulse applied in the direction indicated by the arrowhead in Fig. 3. The secondary current  $I_s$  is induced counterclockwise during the leading edge of  $I_p$ . If the current  $I_d$  takes the direction indicated in the diagram, the current  $I_s$  and a part of  $I_d$

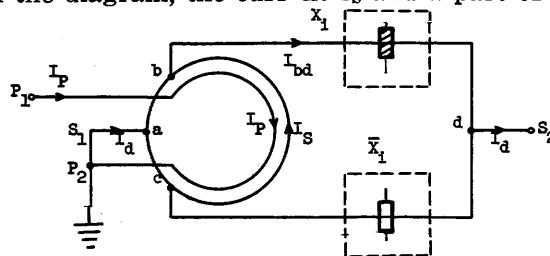


Figure 3a. Cryogenic Control Current Switching Device for the  $i^{\text{th}}$  Cryotron-Pair.

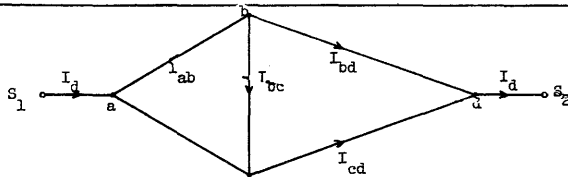


Figure 3b. Equilibrium Current Distribution in the Equivalent Graph.

become additive while passing through the arc ac. The resultant current is sufficient to make the arc ac temporarily resistive and thus the current taking the shorter superconducting path abd which includes the control circuit of the cryotron  $X_i$  is large enough to make the gate  $X_i$  resistive; but the gate  $X_i$  remains superconducting because the current taking the longer path abcd which includes the control circuit of the cryotron  $X_i$  is less in value. This means that a bit "1" is transmitted through the  $X_i$  gate circuit.

It should be noted that no current flows from  $I_d$  through the arc ac once it becomes resistive until a second induced current  $I'_s$  is made to flow clockwise around the secondary loop during the trailing edge of  $I_p$ . At that time the arc ab is made temporarily resistive. The cryotron gate  $X_i$  is changed into resistive because of the larger current which passes through the path acd.

The switching action takes place only when the primary current  $I_p$  is changing either at the leading edge or at the trailing edge of the applied pulse  $I_p$ . This implies that a sine wave current source may be used to take advantage of easy frequency control as well as phase adjustment.

When there are  $m$  distinct gate-pairs in the gate network, a total number of  $m$  control current switching devices are needed. The secondary circuits of these  $m$  devices may be connected in parallel and supplied by a common direct current source with a magnitude  $mI_d$ , or each device may be supplied by a separate source  $I_d$ . In the secondary circuit of each device, the distribution of  $I_d$  between the two parallel superconducting paths bd and bcd (if once the arc ac becomes resistive) or cd and cbd (if once the arc ab becomes resistive) is inversely proportional to the inductances of those paths which differ only in the small amount of inductance of the arc bc. The in-

ductance differences of the connecting paths bd and cd with respect to the arc bc inductance, must be kept as small as possible and the  $m$  devices therefore should not be connected in series and use a common source  $I_d$  unless the arc bc inductance is made large enough to keep these differences small.

Referring to Fig. 3 and assuming that the switching device is symmetric in construction, the arcs ab and ac have the same inductance values ( $L_{ab} = L_{ac}$ ); this is equally true of the load paths bd and cd whose values are ( $L_{bd} = L_{cd}$ ). The distribution of current components are shown in Fig. 3(b). Their magnitudes are found to be:

$$I_{ab} = I_d \quad (1a)$$

$$I_{bd} = (L_{bc} + L_{bd}) I_d / (L_{bc} + 2 L_{bd}) \quad (1b)$$

$$I_{bc} = I_{cd} = L_{bd} I_d / (L_{bc} + 2 L_{bd}) \quad (1c)$$

Each of these currents flows from the node indicated by its first subscript to the node represented by the second subscript.

The current component  $I_{bd}$  must be large enough to maintain the gate  $X_i$  in its resistive state; but the current  $I_{cd}$  passing through the control  $X_i$  must be insufficient to transit the state of the gate  $X_i$ .

If the current source  $I_d$  is removed, persistent currents  $I_{P1}$  and  $I_{P2}$  will circulate in the loops abca and abcd, respectively, and are given by

$$I_{P1} = L_{bd} I_d / (L_{bc} + 2 L_{bd}) \quad (2a)$$

$$I_{P2} = L_{bd} I_d / 2(L_{bc} + 2 L_{bd}) \quad (2b)$$

The removal of  $I_d$  from the device is equivalent to the superposition of another  $I_d$  with reversed polarity on the original  $I_d$ . Since this device is assumed to be a superconducting circuit of a balance bridge type, the reversed  $I_d$  alone is distributed in each of the paths ba, db, dc, and ca with a magnitude  $I_d / 2$  and a reversed polarity with respect to Eq. (1). When these currents and those shown in Eq. (1) are superimposed in each path, the resultant branch currents in the equivalent graph are:

$$I'_{ab} = I_d / 2 \quad (3a)$$

$$I'_{bd} = L_{bc} I_d / 2 (L_{bc} + 2 L_{bd}) \quad (3b)$$

$$I'_{bc} = L_{bd} I_d / (L_{bc} + 2 L_{bd}) \quad (3c)$$

$$I'_{dc} = L_{bc} I_d / 2(L_{bc} + 2 L_{bd}) \quad (3d)$$

$$I'_{ca} = I_d / 2 \quad (3e)$$

We can convert these branch currents into two loop currents which circulate in the loops abca and abdca and thus form the persistent currents, as described in Eq. (2) and shown in Fig. 4. However, when the  $I_d$  source is reapplied, the original current distribution shown in Fig. 3(b) and given by Eq. (1) is recovered. Therefore these persistent currents will not cause an unsatisfactory result.

CRYOTRON GATE NETWORK

A cryotron gate network which is considered equivalent to a transfer-contact relay circuit may be synthesized by one of the following methods:

- (a) Cascading three-terminal sub-networks with complementary outputs;
- (b) Tree networks; and
- (c) Symmetric networks if the logic functions to be realized are identified as symmetric functions.

If any of these techniques is to be used, the following aspects should be taken into consideration:

1. The synthesized circuit should contain a minimum number of cryotrons for economical realization.
2. All gates which constitute the circuit should appear in complementary pairs to meet the requirement of the cryotron control network. Each gate-pair is represented by the complementary Boolean variables  $X_i$  and  $X_{i1}$ .

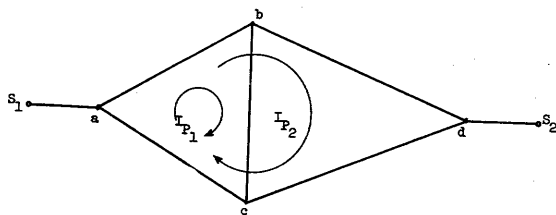


Figure 4. Persistent Currents in the Equivalent Graph After  $I_d$  Source is Removed.

3. Different variable-pairs should be distributed as uniformly as possible throughout the synthesized network to equalize the inductive loads among the control current switching devices.

Three methods of synthesis procedure follow:

A. *Synthesis by Cascading Three-Terminal Sub-Networks*

Use of this method entails the following:

1. Check whether the function  $F(X_m)$  is given in the form of the minimum product as

$$F(X_m) = \prod_{k=1}^n P_k \quad (4)$$

If it is given in this form, its complement  $F'(X_m)$  is then derived as a minimum sum of the following form:

$$F'(X_m) = \sum_{k=1}^n P_k \quad (5)$$

by the application of De Morgan's theorem.<sup>4</sup> If  $F(X_m)$  is not given as Eq. (4), one of the minimization methods<sup>4</sup> should be used to reduce these functions to their optimum forms as given in Eqs. (4) and (5) in order to secure an economical realization.

The function  $F(X_m)$  is expressed as an "And" function made up of the subfunctions  $P_1$  through  $P_n$ . Its complement  $F'(X_m)$  is represented by an "Or" function composed of the complementary subfunctions  $P_1$  through  $P_n$ .

2. By use of the following relationship of the logic function

$$P_k + P_k P_{k+1} = P_k + P_{k+1}, k = 1, 2, \dots, n - 1 \quad (6)$$

the sub-networks realized by those subfunctions can be connected, as shown in the resulting block diagram of Fig. 5.

The sub-network  $P_1$  starts from input terminal I, i.e., node 1, and stops at node 2,  $P_2$  from 2 to 3, and so forth, until  $P_n$  from n to output terminal  $O_1$ . The function  $F(X_m)$  is thus realized between terminals I and  $O_1$  and expressed by Eq. (4). All sub-networks realized by the complementary subfunctions  $P_1$  through  $P_n$

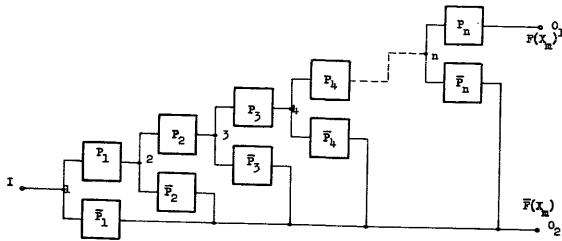


Figure 5. Block Diagram of Cryotron Gate-Network Synthesized by Cascading n Three-Terminal Sub-Networks.

stop at the same output terminal  $O_2$ , but each starts from the node  $k$  which is identical with the subscript  $k$  of its relevant subfunction  $P_k$ . The function  $F(X_m)$  given in Eq. (5) then appears between the terminals  $I$  and  $O_2$ .

Whenever any subfunction  $P_k$  is still a composite function instead of a single variable, the above steps should be repeated for synthesizing the small three-terminal sub-network with node  $k$  as its input terminal and node  $k+1$  and terminal  $O_2$  as its output terminals.

*Illustrative Example 1*

Design a three-terminal cryotron gate network which realizes the logic function

$$F(X_4) = X_1 X_2 X_3 X_4 + X_1 X_2 X_3 \bar{X}_4 + X_1 \bar{X}_2 X_3 X_4 \quad (7)$$

and its complement, using complementary cryotron gate-pairs only.

In this particular problem the function  $F(X_4)$  of Eq. (7) is not given in the same form as Eq. (4). Since it contains only three terms in its first canonical form, its complement  $\bar{F}(X_4)$  should have thirteen terms in the canonical form. By the use of a Karnaugh map,  $F(X_4)$  in the form of Eq. (5) can easily be found:

$$F(X_4) = X_1(X_2 + X_3) + (X_2 + X_3)X_4 + X_1(X_2 + X_3 X_4) \quad (8)$$

When  $F(X_4)$  is complemented by De Morgan's theorem  $\bar{F}(X_4)$  is found to be

$$F(X_4) = (X_1 + X_2 X_3) (X_2 X_3 + X_4) (X_1 + X_2 [X_3 + X_4]) \quad (9)$$

By comparison of Eqs. (8) and (9) with Eqs. (5) and (4), respectively, the subfunctions are:

$$\begin{aligned} P_1 &= X_1 + X_2 X_3, & \bar{P}_1 &= X_1(X_2 + X_3) \\ P_2 &= X_2 X_3 + X_4, & \bar{P}_2 &= (X_2 + X_3)X_4 \\ P_3 &= X_1 + X_2(X_3 + X_4), & \bar{P}_3 &= X_1(X_2 + X_3 X_4) \end{aligned} \quad (10)$$

Each of these three pairs of subfunctions in Eq. (10) represents a smaller three-terminal sub-network which can be further decomposed by repeating the synthesizing procedure because each pair  $P_i$  and  $\bar{P}_i$  ( $i = 1, 2, \text{ or } 3$ ) is also complementary.

The final network is shown in Fig. 6 in which the total number of complementary gate-pairs is ten and the distribution of variable-pairs is 2, 3, 3, 2 for the order  $X_1, X_2, X_3, X_4$ . In this case the required control network should have four cryogenic switching devices. The device for cryotrons assigned by the variable-pair  $X_1$  and  $X_1$  or  $X_4$  and  $X_4$  has two serially connected cryotron controls in each load circuit. Each of the other two devices, however, has three series controls in a load circuit.

*B. Synthesis by the Tree Method*

A complete tree of  $m$  distinct variable-pairs  $X_i$  and  $\bar{X}_i$  possesses  $2^m$  disjunctive outputs, each of which represents a single term of the  $2^m$  possible distinct terms. The outputs are said to be "disjunctive" with respect to each other if there is no superconducting path between any pair of the output terminals 0 through  $2^m - 1$  which also indicate the decimal equivalents of the  $2^m$  distinct terms. Suppose that the variable  $X_i$  is designated by  $2^{i-1}$  as its decimal equivalent.

A complete tree gate network with three variables  $X_3, X_2$  and  $X_1$  distributed in the ratio of 1:2:4, illustrated in Fig. 7, can be converted into a three-terminal network with complementary output functions by connecting those outputs of the tree to form the output terminal

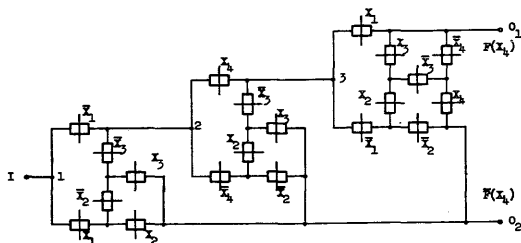


Figure 6. Cryotron Gate-Network Synthesized by Cascading Three-Terminal Sub-Networks.

$O_1$  which represents all terms in the given function  $F(X_m)$ . All the remaining outputs when connected as a common terminal  $O_2$  contribute the complementary function  $F(X_m)$ . The synthesis can be carried out in two major steps:

1. Since each output terminal of a complete tree represents a term of logic function, the given function  $F(X_m)$  and its complement  $F(X_m)$  should be expressed in first canonical forms.
2. Assign variable-pairs to cryotrons such that the output terminals of a complete tree which constitute the terms of the function  $F(X_m)$  can be located as close as possible to the outputs. When these terminals are connected to form  $F(X_m)$  and all

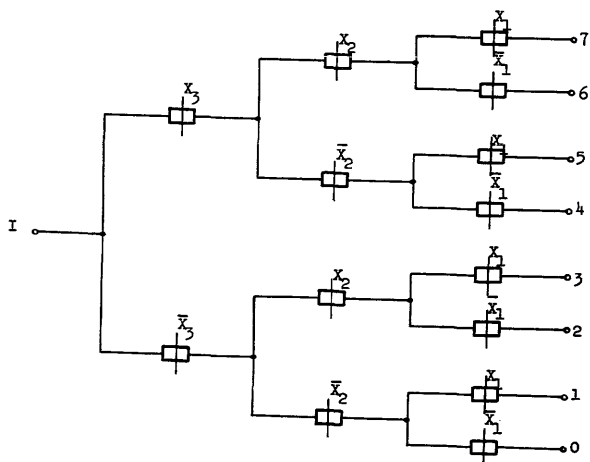


Figure 7. Cryotron Gate-Network in Complete Tree Form.

the remaining output terminals are tied together to form  $F(X_m)$ , some complementary cryotron-pairs are connected in parallel at output terminals. Each pair of these parallel paths is always superconducting. They are therefore redundant and can be removed. An incomplete tree which forms the synthesized gate network thus results.

One possible way of obtaining this result is to find the minimum sum form of the given function by using the minimization method. The relative frequencies of the appearances of various variables in this minimum form are examined. The most frequently appearing variable should be assigned nearest to the input side but not the least common one.

An economical realization and almost uniform distribution of the inductive loads among various cryogenic control current switching devices can also be attained by use of the general principles of contact redistribution.

These two considerations should be combined whenever there are more than one distinct variables which appear with equal and lowest frequency. These variables should be assigned nearest to the output side and also distributed in accordance with the Shannon rule.<sup>4</sup> Since the distribution of variables in a tree is not unique, there are various ways in which they may be assigned.

*Illustrative Example 2.*

Suppose that the given function to be realized by the cryotron gate network is expressed in the minimum sum form

$$F(X_3) = X_2 X_3 + X_1 X_2 X_3 \tag{11}$$

which may be expanded into first canonical form as

$$F(X_3) = X_1 X_2 X_3 + X_1 X_2 X_3 + X_1 X_2 X_3 \tag{12}$$

Its complementary function in first canonical form is found to be

$$F(X_3) = X_1 X_2 X_3 + X_1 X_2 X_3 + X_1 X_2 X_3 + X_1 X_2 X_3 + X_1 X_2 X_3 \tag{13}$$

Since  $F(X_3)$  is given in minimum sum form, shown in Eq. (11), examination of its variables shows that the variable  $X_1$  appears only once,

but  $X_2$  and  $X_3$  appear twice. The variable  $X_2$  or  $X_3$  but not  $X_1$  should be distributed nearest to the input side, otherwise economical realization cannot be obtained.

If the initial variable distribution in a complete tree is 1, 2, 4, the variables can be assigned to cryotrons in the order  $X_3, X_2, X_1$ , or  $X_2, X_3, X_1$ . The variable  $X_1$  is assigned nearest to the output side. The final synthesized network in both cases contains eight cryotrons with three distinct variable-pairs assigned in a distribution of 1, 2, 1. If the decimal equivalent of  $X_1$  is designated by  $2^{i-1}$  one of the three gate-pairs assigned by  $X_1$  and  $X_1$  is tied together at the output terminals 0 and 1; the other two pairs are connected at terminals 2 through 5. Since each pair of these gates is connected in parallel, it is always superconducting and thus must be removed. The gate network synthesized by this method is shown in Fig. 8 in which the complementary variable-pairs  $X_2$  and  $X_3$  can be interchanged. The output terminals  $0_1$  and  $0_2$  are derived from the complete tree by connecting the output terminals 0, 1, 6 and all remainders, respectively.

If the initial distribution of 1, 3, 3 is used in the complete tree in accordance with Shannon's rule, the variable  $X_2$  or  $X_3$  must be located nearest the input terminal. The remaining two variables are assigned to cryotrons in diagonal positions. When  $X_1$  and  $X_3$  are diagonally positioned, the three gate-pairs near terminals 0 and 1, 3 and 7, and 4 and 5 are redundant. If  $X_1$  and  $X_2$  are diagonally assigned, the redundancy occurs at terminals 0 and 1, 2 and 3, and 5 and 7.

For any other cases of assignment, the synthesized network must use twelve cryotrons. This is not economical.

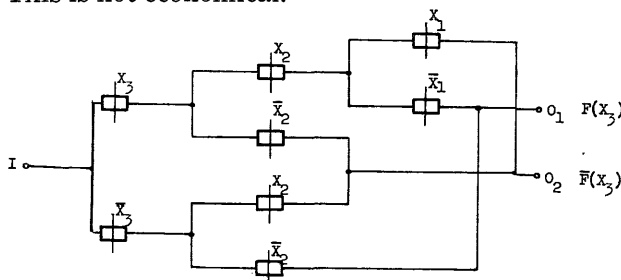


Figure 8. Cryotron Gate-Network Synthesized by the Tree Method.

### Illustrative Example 3

Referring to Example 1, the given function  $F(X_4)$  of Eq. (9) contains the terms whose decimal equivalents according to previous designation are 2, 9 and 14. If the variables are assigned in the order of  $X_1, X_2, X_3, X_4$  from the input terminal I to the output side in a complete tree of 1, 2, 4, 8 distribution, the output terminals 2, 9 and 14 which correspond to the terms 2, 9 and 14 in  $F(X_4)$  are located within two sub-trees of the variables  $X_3$  and  $X_4$ . When the remaining 13 output terminals  $i$  ( $i = 0, 1, \dots, 15$  but exclusive of 2, 9 and 14) are connected to form  $F(X_4)$ , two pairs of complementary gates  $X_3$  and  $X_3$  and five pairs of gates  $X_4$  and  $X_4$  are connected in parallel at the common output terminal. They can be discarded. Therefore sixteen cryotrons are used in the synthesized network. The distribution of complementary cryotron-pairs in the order of  $X_1, X_2, X_3$  and  $X_4$  is 1, 2, 2, 3. In this case,  $X_1$  and  $X_2$  (and also  $X_3$  and  $X_4$ ) can be interchanged because of the same frequency of appearance.

If this property is not used to assign variables, two more cryotrons must be added. As compared with the first method explained in Example 1, four cryotrons are saved by this technique.

### C. Synthesis via Symmetric Network

In a cryotron switching circuit, a logic function realized by a gate network with two terminals built on cryotrons assigned by  $m$  distinct variable-pairs is said to be symmetric when a superconducting path is provided between these two terminals by any  $n$  truth value changes of these  $m$  variable-pairs. The symmetric function according to the definition given here is usually expressed symbolically by

$$S_n(X_1, \dots, X_1, \dots, X_j, \dots, X_m)$$

The variables in the parenthesis represent the variables of symmetry, i.e., the parameters of symmetric function. The subscript  $n$  of  $S$  indicates how many distinctly assigned gate-pairs must have their states transmitted in order to make the path superconducting. This quantity is called the characteristic number.

Some of the variables of symmetry may be represented by the symbols without a bar over each as, for instance,  $X_1, X_m$ , etc., while others

are written  $X_i, X_j$ , etc. These parameters are all considered normally resistive gates, but their complements  $\bar{X}_1, \bar{X}_i, \bar{X}_j, \bar{X}_m$ , etc., are treated as normally superconducting gates. The designations of the parameters  $X_i, X_j$ , etc., and their complements,  $\bar{X}_i, \bar{X}_j$ , etc., are exactly opposite here to those which occur when  $X_i, X_j$ , etc., are not parameters of symmetric function. When a symmetric function is involved, attention must be paid to its parameters.

It should be noted that the sum, as well as its complement, of several symmetric functions, are both symmetric. However, the symmetric property of these composite functions cannot be derived from the previous definition, especially when the parameters are mixed with  $X_i$  and  $\bar{X}_j$ . A more inclusive definition of the symmetric function is introduced by Shannon<sup>4</sup>: "A function of the  $m$  variables  $X_1, \dots, X_i, \dots, X_j, \dots, X_m$  is said to be symmetric in these variables if any interchange of the variables leaves the function identically the same."

The given function  $F(X_m)$  which is identified as symmetric in accordance with either of the described definitions, may be realized in a cryotron circuit by the following procedure:

1. Ascertain whether or not the given function is symmetric. If it is symmetric according to either definition, then determine its parameters and its characteristic numbers. Suppose the parameters are mixed with  $X_1, \dots, X_i, \dots, X_j, \dots, X_m$  and the characteristic numbers are equal to  $p, q$ .

2. Express  $F(X_m)$  in symbolic form by means of the known parameters and characteristic numbers.

$$\begin{aligned}
 F(X_m) &= S_{p,q}(X_1, \dots, X_i, \dots, X_j, \dots, X_m) \\
 &= S_p(X_1, \dots, X_i, \dots, X_j, \dots, X_m) \\
 &\quad + S_q(X_1, \dots, X_i, \dots, X_j, \dots, X_m) \tag{15}
 \end{aligned}$$

3. Find the complement of Eq. (15) and express it in symbolic form.

$$\begin{aligned}
 F(X_m) &= S_{p,q}(X_1, \dots, X_i, \dots, X_j, \dots, X_m) \\
 &= \sum_{\substack{k=p,q \\ k=0}}^m S_k(X_1, \dots, X_i, \dots, X_j, \dots, X_m)
 \end{aligned}$$

The complement is found by replacing the characteristic numbers  $p$  and  $q$  with a set of numerical values  $k$ , where  $k = 0, 1, 2, \dots, m$  but exclusive of  $p$  and  $q$ . It should be noted that all parameters in the parentheses do not take complements.

4. Assign the parameters to all horizontally positioned gates and their complements to all vertically positioned gates.

As an illustration, suppose the parameters for some function are determined as  $X_1, X_2$  and  $X_3$ . The complete symmetric gate network with its proper variable assignment is shown in Fig. 9.

5. Connect the output terminals marked by the component symmetric functions

$$\begin{aligned}
 &S_p(X_1, \dots, X_i, \dots, X_j, \dots, X_m) \text{ and} \\
 &S_q(X_1, \dots, X_i, \dots, X_j, \dots, X_m)
 \end{aligned}$$

to form the composite symmetric function  $F(X_m)$ . All the remaining output terminals marked by

$$S_k(X_1, \dots, X_i, \dots, X_j, \dots, X_m)$$

where  $k = 0, 1, \dots, m$  but exclusive of  $p$  and  $q$ , when connected represent the function  $F(X_m)$ .

6. Discard any redundant gates, one of which usually appears as a complementary gate-pair connected in parallel and equivalent to a short-circuited path. Some of them may be removed by the method of folding.<sup>4</sup>

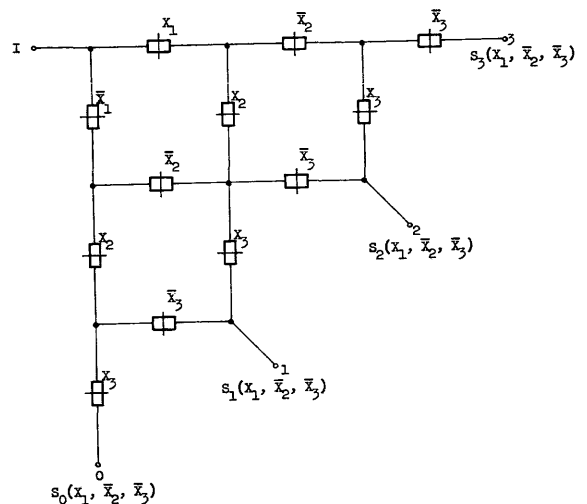


Figure 9. Cryotron Gate-Network in Complete Symmetric Form.

*Illustrative Example 4*

Realize the logic function

$$F(X_3) = X_1 X_2 X_3 + X_1 X_2 X_3 + X_1 X_2 X_3 \quad (17)$$

by cryotron circuit.

The given function  $F(X_3)$  is not symmetric by the first definition if we consider the parameters of symmetry to be  $X_1, X_2, X_3$  or  $X_1, X_2, X_3$ . When  $X_1$  and  $X_2$  or  $X_1$  and  $X_3$  are interchanged, Eq. (17) is left identically the same. Thus it is symmetric in the variables  $X_1, X_2, X_3$  by the second definition. Eq. (17) is equal to "0" (which designates a resistive path), because each term contains two resistive gates. Whenever any two of the resistive gates  $X_1, X_2$  and  $X_3$  become superconducting, Eq. (17) is changed to "1" (which represents a superconducting path). The characteristic number is therefore equal to 2 in accordance with the first definition. In addition, the characteristic number should satisfy the following relation:

$${}_m C_n = m! / n! (m - n)! \quad (18)$$

which shows all possible ways of producing the characteristic number  $n$ . The given function  $F(X_3)$  expressed in symmetric symbolic notation is

$$F(X_3) = S_2(X_1, X_2, X_3) \quad (19)$$

Its complement is found to be:

$$F(X_3) = S_{0,1,3}(X_1, X_2, X_3) \quad (20)$$

The output terminal 2 of Fig. 9 represents Eq. (19). All other terminals 0, 1, and 3 which are connected represent Eq. (20). The gate-pair  $X_3$  and  $X_3$  located at the left bottom part of Fig. 9 is redundant and therefore must be removed.

If Eqs. (19) and (20) are synthesized by the tree method 2 more cryotrons are needed.

## DISCUSSION

Complementary logic functions realized by cryotron circuits are easily accomplished through the design of a control network and the synthesis of a gate network. If the total number of distinct cryotron-pairs used in the circuit is  $m$  then the minimum number of cryogenic switching devices is equal to  $m$ . When

the Boolean variables assigned to the cryotron-pairs cannot be distributed uniformly, some switching devices may have heavy inductive loads. In such a case, additional switching devices should be added to equalize the inductive loads in their secondary circuits. However, those devices which control cryotron-pairs with the same variable assignment should be operated synchronously, so that their primary circuits are connected in series and supplied by a common current source. The synthesized gate network should use minimal elements to reduce the cost; and it should have uniformly distributed Boolean variables to equalize inductive loads in the switching devices.

If the switching device is considered equivalent to the relay control winding and the complementary gate-pair equivalent to the transfer relay contacts, the same synthesis technique which has been used in the design of the relay switching circuit to synthesize a gate network can be applied. The various synthesis procedures are systematic and easy to follow especially for anyone familiar with relay switching circuits.

In the transfer-contact relay circuit, a minimal contact network without any redundancy may contain a static hazard in the tie or cut sets<sup>4</sup> and cause momentary malfunction or even a complete failure of the circuit. Although the cryotron gate may have an intermediate-state during the state-transition, its superconductive-to-resistive state change (or vice versa) is so rapid that such trouble seems non-existent in a cryotron circuit. This simplifies the synthesis procedure.

When the realized circuitry is made more complex, the overall circuit inductance is proportionately increased. This in turn increases the time constant of the circuit. The switching speed is accordingly lowered. Low gain film cryotrons which have less inductances can be used as the circuit elements to overcome this limitation.

In the method of cascading three-terminal sub-networks, the distribution of the various gate-pairs can be examined by counting the frequency of the appearances of the variables in the given logic function while it is expressed



in its minimum product or sum form. Although the method is simple and straightforward, it seems difficult to synthesize the cryotron network with a minimum number of gates.

In a tree-network of  $m$  distinct gate-pairs, aside from the single pair nearest to the input terminal  $I$ , the distribution among the remaining  $m - 1$  pairs can be made almost uniform by Shannon's rule, though an absolute uniform distribution is impossible. But this almost uniform distribution cannot assure that the synthesized network is the most economical realization which has been illustrated, as Example 3. We should assign variables to cryotrons in such a way that those output terminals which represent the terms of logic function to be synthesized are located as close as possible. We should try to group these terminals within the same sub-tree or two adjacent sub-trees by consideration of the relative frequencies of variable appearance and/or the Shannon rule. This will increase the number of redundant gate-pairs which can be removed.

The gate-pairs in a complete symmetric network with  $i$  parameters cannot be uniformly distributed because a single pair is always nearest to the input and the  $i$ -pair is located nearest to the output terminals. When  $i$  is made larger, the degree of non-uniformity increases, proportionately. When a function is symmetric special attention must be paid to its parameters because the designation of the variable  $X_j$  is resistive. It is exactly opposite if  $X_j$  is not a parameter of the symmetric function. The designation of resistive  $X_j$  is, therefore, restricted in the symmetric network. In this case, the cryotron control assigned by  $X_j$  must be kept by a sufficient current flowing through it to maintain the gate  $X_j$  in its resistive state.

#### BIBLIOGRAPHY

1. BREMER, J. W., "Superconductive Devices," McGraw-Hill Book Company, Inc., New York. 1962.
2. BUCK, D. A., "The Cryotron—A Superconductive Computer Component," Proc. IRE, p. 482. April 1956.
3. BUCKINGHAM, M. J., "The Persistatron: A Superconducting Memory and Switching Element for Computers," Fifth International Conference on Low Temperature Physics and Chemistry, edited by Joseph R. Dillinger, p. 229. August 26–31, 1957.
4. CALDWELL, S. H., "Switching Circuits and Logical Design," John Wiley and Sons, Inc., New York. 1958.
5. DAVIES, P. M., "A Superconductive Associative Memory," Proc. SJCC, p. 79. May 1962.
6. HAYNES, M. K., "Cryotron Storage, Arithmetic and Logical Circuits," Solid-State Electronics, Vol. 1, p. 399. September 1960.
7. ITTNER, W. B., III, and KRAUS, C. J., "Superconducting Computers," Scientific American, Vol. 205, No. 1, p. 125. July 1961.
8. LEARN, A. J., "Superconducting Computers," Electronics, Vol. 34, p. 45. November 24, 1961.
9. LEWIN, M. M., "Retrieval of Ordered Lists from a Content-Addressed Memory," RCA Review, Vol. 23, p. 215. June 1962.
10. NEWHOUSE, V. L., and FRUIN, R. E., "A Cryogenic Data Addressed Memory," Proc. SJCC, p. 89. May 1962.
11. SEEBER, R. R., Jr., "Cryogenic Associative Memory," National Conference, ACM, Milwaukee, Wisconsin. August 1960.
12. SEEBER, R. R., Jr., and LINDQUIST, A. B., "Associative Memory with Ordered Retrieval," IBM J., Vol. 6, No. 1, p. 126. January 1962.
14. SLADE, A. E., and MCMAHON, H. O., "A Cryotron Catalog Memory System," Proc. EJCC, p. 115. December 10–12, 1956.
14. SLADE, A. E., and MCMAHON, H. O.: "A Review of Superconductive Switching Circuits," Proc. National Electronics Conference, Vol. 13, p. 574, Chicago, Illinois. October 7–8, 1957.
15. SLADE, A. E., and SMALLMAN, C. R., "Thin Film Cryotron Catalog Memory," Auto-

- matic Control, Vol. 13, No. 2, p. 48. August 1960. *Also*: Solid-State Electronics, Vol. 1, p. 357. September 1960.
16. VAIL, C. R., LUCAS, S. P., OWEN, H. A., and STEWART, W. C., "An Approach to the Experimental Study of Persistent-Current Devices," Solid-State Electronics, Vol. 1, p. 279. September 1960.
17. YOUNG, D. R., "Superconducting Circuits," Progress in Cryogenics, Vol. 1, edited by K. Mendelssohn. London, England. Heywood and Co., Ltd. 1959.

# BINARY-COMPATIBLE SIGNED-DIGIT ARITHMETIC

*Algirdas Avizienis*  
*University of California, Los Angeles, California*  
*and*  
*Jet Propulsion Laboratory, Pasadena, California*

## 1. Signed-Digit Number Representations

Signed-digit representations are positional number representations with a constant integer radix  $r \geq 3$ , in which the allowed values of the individual digits  $z_i$  are a sequence of  $q$  ( $r+2 \leq q \leq 2r-1$ ) integers:  $(-a, \dots, -1, \dots, a)$ , where  $a$  is the maximum digit magnitude. The value of  $a$  is chosen from the following range:

$$\begin{aligned} \frac{1}{2}(r_o + 1) \leq a \leq r_o - 1, \\ \text{for odd radices } r_o \geq 3; \\ \frac{1}{2}r_e + 1 \leq a \leq r_e - 1, \\ \text{for even radices } r_e \geq 4. \end{aligned}$$

Both positive and negative digit values are allowed. The individual digits contain all sign information, therefore a separate sign digit is not required for the entire number. For example, only one set of digit values exists for radix  $r = 3$  (values  $-2, -1, 0, 1, 2$ ) and for  $r=4$  (values  $-3, -2, -1, 0, 1, 2, 3$ ); for radix  $r=10$  there are four sets, from 13 values ( $-6$  to 6) to 19 values ( $-9$  to 9).

Signed-digit representations are redundant, that is, each radix  $r$  digit  $z_i$  assumes more than  $r$  different values. In a *conventional* (nonredundant) representation only  $r$  values of a digit ( $0, 1, \dots, r-1$ ) are allowed. Signed-digit numbers have *minimal redundancy* ( $r_o + 2$  or  $r_e + 3$  digit values) when the value of  $a$  is chosen as follows:

$$\begin{aligned} a &= \frac{1}{2}(r_o + 1) = a_{\min} \\ a &= \frac{1}{2}r_e + 1 = a_{\min} \end{aligned}$$

They have *maximal redundancy* ( $2r-1$  digit values) for both odd and even radices when  $a$  has the value:

$$a = r - 1 = a_{\max}$$

The characteristic properties of signed-digit representations are listed below.

1. The algebraic value  $Z$  of the number  $z$  composed of  $n + m + 1$  digits ( $z_{-n} \dots z_{-1} z_0 z_1 \dots z_m$ ) is given by the conventional expression:

$$Z = \sum_{i=-n}^m z_i r^i$$

2. Algebraic value  $Z = 0$  if, and only if all  $z_i = 0$ .

3. The *sign* of the algebraic value  $Z$  is given by the sign of the most significant (left most) nonzero digit.

4. To form the representation of the additive inverse  $-Z$ , the sign of every nonzero digit  $z_i$  is changed individually.

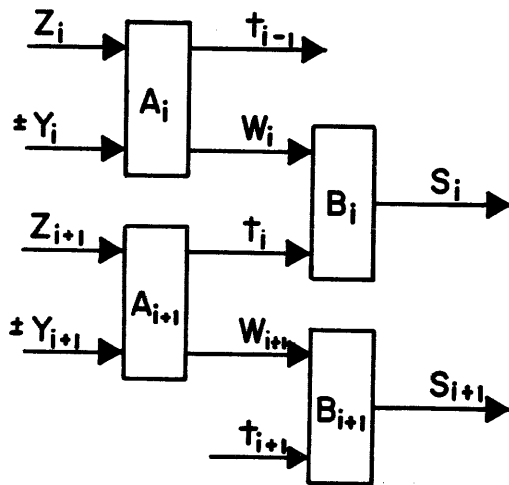
5. The addition and subtraction of two signed-digit operands  $z$  and  $y$  satisfies  $s_i = f(z_i, y_i, z_{i+1}, y_{i+1})$  for all positions  $i$ , where  $s_i$  are digits in the representation of the sum or difference  $s = z \pm y$ .

There are no carry-propagation chains in signed-digit addition (or subtraction), that is, any digit of the sum is a function of only two adjacent digits of the operands. Subtraction is performed as a change of sign followed by an addition. The logical time of one addition is

independent of the length of the operands and is equal to the time required by one digital position. The rules of operation and block diagram of a signed-digit adder for one pair of digits are shown in Figure 1. Here  $t_i$  is called the *transfer digit* and may assume the values 1, 0, and  $-1$ ;  $w_i$  is called the *interim sum digit* and may assume one from the sequence of values:  $(-w_{\max}, \dots, -1, 0, 1, \dots, w_{\max})$ . In the case of minimal redundancy,  $w_{\max} = a_{\min} - 1$  is chosen; in all other cases ( $a > a_{\min}$ ),  $w_{\max}$  is chosen from the range

$$a_{\min} - 1 \leq w_{\max} \leq a - 1$$

Previous publications present a detailed description of signed-digit number systems<sup>1</sup>, and the general rules for variable-precision and



#### ADDITION (TWO STEPS)

$$(A) \quad Z_i + Y_i = t_{i-1} + w_i$$

$$t_{i-1} = 0 \text{ if } |Z_i + Y_i| \leq w_{\max}$$

$$t_{i-1} = 1 \text{ if } Z_i + Y_i > w_{\max}$$

$$t_{i-1} = -1 \text{ if } Z_i + Y_i < -w_{\max}$$

$$(B) \quad S_i = w_i + t_i$$

**SUBTRACTION:** change sign of  $Y_i$  if  $Y_i \neq 0$  and add

significant digit operations<sup>2</sup>, on which this paper is based.

#### 2. Application of Signed-Digit Arithmetic

The elimination of carry propagation removes a fundamental constraint of digital arithmetic units and necessitates a reconsideration of all arithmetic algorithms. The most important new aspects of signed-digit (to be abbreviated "s-d" from now on) arithmetic are:

1. the addition time of a parallel adder consisting of any number of cascaded identical digit-adder packages is (logically) constant;

2. the most significant digits of the product (as well as of the quotient) are generated first and may be processed further before the less significant digits become available;

3. the addition (and subtraction) algorithms apply to operands of an arbitrary multiple precision (arbitrary length with respect to the length of the adder): the most significant sections are added first and may be immediately processed further;

4. the multiplication and division algorithms are identical both for single- and multiple-precision operands;

5. in floating-point arithmetic the application of a special digit value  $\emptyset$  (the space-zero) to designate non-significant positions allows the implementation of normalized significant digit arithmetic<sup>3</sup>;

6. the non-significant digit value (space-zero)  $\emptyset$  may be employed to determine the completion of a multiple-precision significant digit algorithm; in this case the lengths of the operands may be unknown at the beginning of the algorithm.

A rather novel arithmetic processor may be constructed if these properties of s-d arithmetic are utilized. The properties (2) and (3) permit the elimination of temporary storage of intermediate results in a complex algorithm; right shifts are not employed, and the flow of operand and result digits is only in one direction (to the left), resembling signal flow through gate networks. The properties (3), (4) and (6) cancel the distinction between the implementation of single and multiple-precision algorithms in an arithmetic processor and allow the completion of an algorithm to be de-

Figure 1. Rules of Signed-Digit Addition

tected by an inspection of the operands. Property (5) permits the inclusion of significant digit arithmetic while retaining all advantages of the number system and without changing the algorithms. Finally, property (1) permits the assembly of fixed-time adders of any length from identical building blocks (without any carry-lookahead or similar logic structures); this feature promises convenient assembly and restructuring of arithmetic processors for hardware implementation of complex algorithms in a variable structure computer<sup>4</sup>. The cost of the various innovations, when compared to a parallel binary arithmetic unit, is found in the greater complexity of the individual digit adders and in the increased storage requirements (for the same precision of operands) due to the redundancy of the number representation.

A further important consideration in the definition of a practical signed-digit arithmetic processor is its compatibility with the widely employed conventional binary number system. The potential application of s—d arithmetic in the UCLA Variable Structure Computer<sup>5</sup> established the need for a binary-compatible signed-digit arithmetic. In such an arithmetic the s—d arithmetic processor accepts binary as well as s—d operands and produces s—d results. Furthermore, a reconversion algorithm is provided which allows the reconversion of s—d numbers to conventional binary forms either in the signed-digit or in a conventional binary arithmetic processor. The following sections describe a set of algorithms for a binary-compatible s—d arithmetic and outline the implementation of an arithmetic processor which employs these algorithms. The algorithms are applicable to any radix  $r=2^k$ , with  $k \geq 2$ ; the specific description will be given in terms of the radix  $r=8$ . An adaptation for a decimal-compatible signed-digit arithmetic is also quite evident.

3. Structure of One Digit-Adder

It is evident that any maximal-redundancy radix  $r$  s—d number system, with the allowed digit values ranging from  $-(r-1)$  to  $r-1$ , includes all allowed digit values (0 to  $r-1$ ) of the conventional number system with the same radix. The prefixing of an individual sign to each digit then will convert a conventional

number to an s—d number of the same algebraic value, in which all digits carry the same sign as the conventional number. The conversion in this case requires no arithmetic and is executed simultaneously for all digits.

A binary number may be interpreted as a number of radix  $r=2^k$  ( $k \geq 2$ ) by grouping the binary digits into groups of  $k$  bits each. The values of individual digits are then in binary-coded form. Consequently, a sign-and-magnitude form of a binary number becomes a radix  $2^k$  maximal—redundancy s—d number (with all digits sharing a common sign) by means of an interpretation of digit grouping. Any register which provides storage for a radix  $2^k$  s—d number will be able to store the binary number as an s—d number. Since both positive and negative digit values occur in one s—d number, it is necessary to choose the representation for negative digit values. Sign-and-magnitude and complement forms both may be employed within each digit; the complement with respect to  $2^{k+1}$  is a generally convenient choice. Table I shows the "16's complement" coding of digit values for radix 8.

The otherwise unused bit pattern 1000 is employed to designate the non-significant space-zero  $\emptyset$ . The choice of representation for negative digit values determined the rule for digit-wise subtraction, which is implemented as the addition of "16's complements" of the subtrahend digits to the digits of the minuend. The addition table for first step of a radix 8 digit-adder is presented in Table II.

The second step of addition generates the sum digit of value  $s_i = w_i + t_i$ . The space-zero  $\emptyset$  (pattern 1000) is detected by a special

TABLE I: CODING OF DIGIT VALUES FOR RADIX 8

Value	Code	Value	Code
0	0000	$\emptyset$	1000
+1	0001	-1	1111
+2	0010	-2	1110
+3	0011	-3	1101
+4	0100	-4	1100
+5	0101	-5	1011
+6	0110	-6	1010
+7	0111	-7	1001

TABLE II: FIRST-STEP ADDITION TABLE FOR RADIX 8 (MAX. REDUNDANCY)

$z_1 + y_1$	$w_1$	$t_{i-1}$	$z_1 + y_1$	$w_1$	$t_{i-1}$
+14	+6	+1	-14	-6	-1
+13	+5	+1	-13	-5	-1
+12	+4	+1	-12	-4	-1
+11	+3	+1	-11	-3	-1
+10	+2	+1	-10	-2	-1
+9	+1	+1	-9	-1	-1
+8	0	+1	-8	0	-1
+7	-1	+1	-7	+1	-1
+6	+6	0	-6	-6	0
+5	+5	0	-5	-5	0
+4	+4	0	-4	-4	0
+3	+3	0	-3	-3	0
+2	+2	0	-2	-2	0
+1	+1	0	-1	-1	0
0	0	0			

logic circuit at the input of the digit-adder. It is always entered as the value zero (0000) into the adder. This conversion allows the use of the same digit-adder for both "conventional addition" and "significance addition" modes; in the latter mode the output of the digit-adder is forced to space-zero  $\emptyset$  (pattern 1000) if either one or both input digits were space-zeros. In "conventional addition" mode the  $\emptyset$  digits may be employed to mark the end of a variable-length number; in this case, the output is set to  $\emptyset$  only when both input digits are space-zeros. Conventional addition without use of the space-zero values is also possible.

An inspection of Table II shows that an ordinary four-bit (modulo 16) binary adder will generate three correct bits of the interim sum  $w_i$ . The correct value of the leftmost (sign) bit and of the outgoing transfer digit  $t_{i-1}$  is computed by a separate logic circuit. To add the incoming transfer digit  $t_i$  to the interim sum  $w_i$  one of two methods may be chosen: either a second pass to add  $t_i$  is made through the same 4-bit adder which was used to compute  $w_i$ , or a separate  $\pm 1$  circuit (carry-propagate and borrow-propagate arrangement over 4 bits) is employed. A detailed study and comparison of these two methods is currently in progress.<sup>11</sup>

#### 4. Addition and Subtraction

A convenient range for the signed-digit processor is the fractional range  $1 > X > -1$  in which the value  $X$  of an  $n$ -digit number  $x$  composed of the digits  $x_1 x_2 \dots x_n$  is:

$$X = \sum_{i=1}^n x_i r^{-i}$$

Overflow occurs when  $|t_0| = 1$  is generated by the adder position  $i=1$ ; therefore the range in which overflow will never be indicated is

$$\sum_{i=1}^n (r-2)r^{-i} \geq X \geq - \sum_{i=1}^n (r-2)r^{-i}$$

In the "significance addition" mode the space-zero digit  $\emptyset$  enters the digit adder as the value zero, but the output of the digit adder is forced to the space-zero by the special circuit which senses  $\emptyset$ , and is enabled by the "significance addition" command. In this case, the addition  $x_i + \emptyset$  will generate  $|t_{i-1}| = 1$  when  $|x_i| = r-1$ , and  $t_{i-1} = 0$  otherwise, while  $s_i = \emptyset$  will always hold. The result will be rounded to the precision of the shorter operand. The roundoff by means of  $\emptyset$  digits will be without bias if every digit value is assumed to occur with equal probability. The maximum magnitude of the discarded part is  $E_{\max} = r^{-i}(r-1) - r^{-n}$  when it consists of the digits  $x_i, x_{i+1}, \dots, x_n$ . This value is reduced (at the cost of additional logic) to  $E_{\max} = r^{-i}(r/2) - r^{-n}$  by executing  $x_i + \emptyset$  as  $x_i + (\pm r/2)$  whenever  $x_i \neq \emptyset$ . The sign of  $\pm r/2$  is chosen to be the same as the sign of  $x_i$ ; either sign is acceptable for  $x_i = 0$ . In either case, roundoff has been implemented as a part of addition and executed concurrently at the level of individual digits. In the "conventional addition" mode,  $\emptyset$  is always entered as value zero, and the digit adder output is forced to  $\emptyset$  only where both input digits have the value  $\emptyset$ . The result retains the length of the longer operand.

In a floating-point system the exponent is represented by a signed-digit integer of desired range which is held at the left end of the fraction. All fractions are kept in normal form and non-significant positions are filled with  $\emptyset$  (space-zero) digits. There is at least one  $\emptyset$  at the right end of every fraction. When leading zeros develop in a fraction, it is normalized by

s—d augend	Z =	.6 0 7 4 7 5 0
s—d addend	Y =	.2 7 1 5 0 0 0
digit sum <sup>s</sup>	$z_i + y_i =$	<u>4 7 6 11 7 5 0</u>
interim sum digits	$w_i =$	4 1 6 1 1 5 0
transfer digits	$t_i =$	0 1 0 1 1 0 0
digit-adder output	$w_i + t_i =$	<u>.5 1 7 0 1 5 0</u>
conventional sum	S =	.5 1 7 0 1 5 0
significance sum	S' =	.5 1 7 0 0 0 0

Note: the value 0 enters the adder as the value 0 in forming the digit sums  $z_i + y_i$ .

Figure 2. Addition Example (Radix 8)

left shifts and a corresponding decrease of the exponent; significant digits are lost. Overflow is corrected by the transposition of the radix point of the result one position to the left and an increase of one in the exponent; the fraction gains one significant digit. Only one exponent is needed with multiple-precision fractions, since the exponent serves as the index of the relative position of the leftmost digit  $x_1$  of the fraction. Further details of floating-point addition have been presented in a previous paper.<sup>2</sup>

5. The Pack-Add Algorithm and Normalization

The "pack-add" algorithm is a variation of the usual "clear-add" algorithm. It is applicable in several aspects of maximal-redundancy s—d arithmetic. The packed form  $x^*$  of the s—d number  $x$  retains the algebraic value of  $x$ , but has only  $r+1$  possible digit values:  $\pm(r-1, \dots, 1, 0, -1)$ . "Pack-add  $x$ " is implemented as the addition followed by an immediate subtraction of  $\pm 1$  to the  $n$ -digit fraction  $x$ . The addend  $\pm 1$  is represented by the  $n$ -digit fraction  $c$ , in which all digits have the value  $c_i = \pm(r-1)$ , plus an incoming transfer digit  $t_n = \pm 1$ . Simultaneously the overflow transfer digit  $t_0 = \pm 1$  is discarded, thus subtracting  $\pm 1$  from  $x + c + t_n$ ; as a result the algebraic values of  $x$  and  $x^*$  remain equal. The signs of  $c_i$  and  $t_n$  must be the same as the sign of the leftmost digit  $x_1$  in order to guarantee  $|t_0| = 1$ ; if  $x_1 = 0$ , both signs of  $c_i$  and  $t_n$  are allowable. The sign of  $c_i$  which has been employed in the algorithm is designated as the *dominant sign*. When  $x$  contains one or more 0 digits at its right end, the leftmost 0 digit ( $x_{n+1} = 0$ ) will generate  $t_n = \pm 1$  as the result of the addition

of  $c_{n+1}$ , which is executed as  $\pm(r-1) + 0$  in the "significance addition" mode.

After the execution of one pack-add algorithm, digits of the opposite (non-dominant) sign may have only the magnitude 1. Furthermore, any pair of these opposite sign digits of unit magnitude will be separated by at least one digit; at least one of the separating digits will be a non-zero digit with the dominant sign. The minimum separation increases by one digit for every successive application of the pack-add algorithm. These properties facilitate the formation of multiples of s—d multiplicands and divisors in the multiplication and division algorithms.

The application of the pack-add algorithm also permits the elimination of *pseudo-normal* s—d operands. There exists a class of maximal-redundancy s—d numbers of the same algebraic value which assume the forms  $x' = .16 \dots 0$ , and  $x = .02 \dots 0$ , with the worst case being  $x' = .177 \dots 7 7 0$ , and  $x = .000 \dots 0 1 0$ , in which the form  $x'$  is *pseudo-normal*. The pseudo-normal form satisfies  $x_1 \neq 0$ , but fails to satisfy a minimum magnitude requirement for its algebraic value; it also presents an incorrect count of significant digits.

An exact rule for the recognition of normal forms is required in maximal-redundancy s—d arithmetic. An application of the pack-add algorithm to a pseudo-normal form will cause the appearance of leading zeros and permit further normalization of the operand. A convenient definition of a normal form is that one

s—d operand	Z =	.6 0 7 4 7 5 0
packing addend	C =	.7 7 7 7 7 7 7
digit sums	$z_i + y_i =$	<u>15 7 0 3 16 14 7</u>
interim sum digits	$w_i =$	5 1 0 3 6 4 1
transfer digits	$t_i =$	1 1 0 0 1 1 1
digit-adder output	$w_i + t_i =$	<u>.6 1 0 4 7 5 1</u>
packed form	Z* =	.6 1 0 4 7 5 0

Notes: significance addition is employed and the overflow transfer digit  $t_0$  is discarded to obtain Z\*.

The second application of the pack-add algorithm will yield the form Z\*\* = .5704750 without negative digit values.

Figure 3. Pack-Add Algorithm Example (Radix 8).

of the following conditions should be satisfied by  $x$ :

$$\begin{aligned} |x_1| &\geq 2; \\ x_1 &= +1, \text{ and } x_2 \geq 0; \\ x_1 &= -1, \text{ and } x_2 \leq 0; \\ x_2 &= \emptyset \text{ with any value of } x_1. \end{aligned}$$

The magnitude range of non-zero normal forms  $x$  is given by

$$1 - r^{-n} \geq |X| \geq r^{-2} (r-1) + r^{-n}$$

where  $x_n$  is the least significant digit of  $x$ . The normal form of the fraction value  $X = 0$  is uniquely represented by  $x_1 = 0$  and  $x_2 = \emptyset$ . Evidently, other definitions of normal forms may be convenient under different circumstances.

Still another property of the pack-add algorithm is the elimination of all opposite-sign digit values, which is a method of reconversion into the conventional binary form. A digit with the opposite signs survives a pack-add operation only if there was a zero digit at its left; the new form will contain an opposite-sign unit value digit in the former position of this zero digit. Consequently, the longest string of zero digits ending with an opposite-sign digit at its right determines the number of pack-add operations needed for a reconversion;  $k+1$  operations are needed when this longest string contains  $k$  zero digits. The worst case will require  $n-1$  pack-add operations for an  $n$ -digit form which has  $n-2$  zeros separating non-zero digits  $x_1$  and  $x_n$  with unlike signs; however, the average number of operations required for complete elimination of opposite-sign digits will be considerably lower.

## 6. Multiplication

An important property of signed-digit multiplication is the availability of the most significant product digit in its final form after the first two steps of the iterative multiplication algorithm. Given the radix  $r$   $s$ - $d$  multiplicand  $X$  and multiplier  $Y$  (with digits  $y_1, y_2, \dots, y_n$ ), the algorithm is:

$p^{(j)} = r [p^{(j-1)} + x y_j]$ , with  $j = 1, 2, \dots, n$ ; where  $P^{(0)}$  is an initial augend, and  $P^{(n)} = P^{(0)} + XY$  is the product<sup>2</sup>. Only left shifts are employed in this algorithm. For maximal redundancy radix 8 multiplication, the multiplier

digits  $y_j$  are recorded sequentially into digits  $y'_j$  such that  $4 \geq y'_j \geq -4$  holds. The recoding generates a digit  $y'_0$  (value 0, +1, or -1); therefore  $P^{(0)} = rXy'_0$  is specified. The value of  $y'_j$  is a function of the values of  $y_j, y_{j+1}$ , and of the sign of  $y_{j+2}$  during the  $j$ -th step of multiplication.

In binary arithmetic, high speeds of multiplication can be attained by cascading carry-save adders<sup>6</sup> to form a multiple-operand adder which sums several multiples ( $\pm 2^i$ ) $X$  of the binary multiplicand  $X$  at once and produces a partial product in stored-carry form. The final product is obtained by entering the stored-carry form of the product into a carry-propagate adder. The multiple  $2^i X$  are obtained by left-shifting the multiplicand  $X$ . Signed-digit adders may be similarly cascaded to add several operands at once. In this case,  $m-1$  adders will be required to form the  $s$ - $d$  sum of  $m$   $s$ - $d$  operands; evidently, a carry-propagate adder is no longer necessary. The last remaining problem is the formation of the multiples ( $\pm 2^i$ ) $X$  of the  $s$ - $d$  multiplicand  $X$ . Two cases must be distinguished here.

In the first case,  $X$  is delivered to the  $s$ - $d$  processor in conventional binary sign-and-magnitude form; the multiples  $2^i X$  of the magnitude are obtained by left shifts of  $X$  before entering it into the  $s$ - $d$  adder. For the radix 8, the multiples  $2X$  and  $4X$  will be obtained by one-bit and two-bit left shifts of  $X$ ; the multiple  $3X$  is computed by adding  $X$  to  $2X$  in the  $s$ - $d$  adder and storing the result (now in  $s$ - $d$  form) in a separate register. With a binary multiplicand  $X$  and the stored multiple  $3X$ , a single addition in a 2-input  $s$ - $d$  adder will account for one radix 8 multiplier digit  $y'_j$ . For multiplication using  $m$  radix 8 multiplier digits at once,  $m$  signed-digit adders must be cascaded; the partial product is generated in  $s$ - $d$  form.

In the second case, the multiplicand  $X$  is in  $s$ - $d$  form a one-bit left shift will not yield the  $s$ - $d$  form of  $2X$  because adjacent radix 8 digits may have different signs. Multiple-forming circuits<sup>2</sup> may be applied to generate  $2X$  and  $4X$ ; however, a more general solution is provided by the use of one more adder to add in the negative digits separately from the positive digits. Since the weight of the leftmost bit in



the radix 8 digit  $x_1$  is  $-8$ , this bit may be considered as having the weight  $+1$  in the digit  $x''_{i-1}$  of a negative sign-and-magnitude octal number  $X''$ , while  $X$  without the  $-8$  weighted digits remains a positive sign-and-magnitude octal number  $X'$ , where  $X=X' + X''$  holds. Now  $X'$  and  $X''$  may be shifted bitwise and  $4X' + 4X''$  or  $2X' + 2X''$  added to  $p^{(j-1)}$  by the use of two cascaded  $s-d$  adders, or by using the same  $s-d$  adder twice.

We observe that with the above discussed approach, an arrangement of two cascaded  $s-d$  adders will handle two multiplier digits at once for a binary multiplicand  $X$ , and one digit at once for an  $s-d$  multiplicand  $X$ . When more than two  $s-d$  adders are cascaded, it is possible to use a cascade of  $m > 2$  adders for  $m$  multiplier digits at once in case of a binary  $X$ , and for  $m-1$  multiplier digits at once for a signed-digit  $X$ , if the  $s-d$  multiplicand  $X$  is converted to its packed form  $X^*$  prior to the multiplication. The application of the pack-add algorithm to  $\pm X$  leaves  $\pm 1$  as the only digit value of the opposite sign in  $X^*$ ; the number of successive applications is equal to the minimum separation (in digits) between any two opposite sign ( $\pm 1$ ) digits. A single  $s-d$  adder is sufficient to add all  $m-1$  multiples of the opposite sign unit digits of  $X^*$ , when the  $s-d$  multiplicand  $X$  is packed  $m-2$  times before placed into the multiplicand register as  $X^*$ .

In conclusion, it is observed that the cascading of  $s-d$  adders differs from the cascading of carry-save adders for binary multiplication in several significant aspects. First, the carry-save structure serves as a special-purpose adder for multiplication only and requires a carry-propagate adder to generate the final result. The  $s-d$  cascade is composed of fully complete  $s-d$  adders which may be used separately for independent operations when fast multiplication is not required. Furthermore, the radix 8 division algorithm uses the same  $s-d$  adder arrangement as the one-multiplier-digit algorithm for multiplication.

The roundoff of a product is implemented by concluding the algorithm when the required number of product digits has been generated. Significant digit multiplication<sup>3,8</sup> employs the "conventional addition" mode for each step of the algorithm; the number of significant digits

in the product is determined by a sequential scanning of both operands<sup>2</sup>. Exponents in a floating-point multiplication are handled in the usual manner.

7. Division

The most convenient method of division in signed-digit arithmetic is the algorithm described by Robertson<sup>7</sup>, in which the representation of the quotient is redundant and the value of the next quotient digit is selected by comparing approximate magnitudes of the divisor and the partial remainder. One quotient digit  $q_j$  is generated during each step of the division algorithm, which consists of an iterative sequence of left shifts and additions or subtractions:

$R^{(j)} = r [R^{(j-1)} - X q_j]$ , with  $j=1, 2, \dots, n$  where  $X$  is the divisor,  $R^{(0)}$  is the dividend (satisfying  $|R^{(0)}| < k |X|$ , with  $k$  to be specified),  $R^{(n)}$  is the remainder, the  $R^{(j)}$  for  $n > j > 0$  are partial remainders, and  $n$  specifies the required precision of the quotient<sup>2</sup>.

For radix 8 signed-digit division, the allowed values of the quotient digits  $q_j$  may be chosen to be in the range  $4 \geq p_j \geq -4$ . In this case the multiplication algorithm and the division algorithm are interchangeable when  $R^{(j)}$  is substituted for  $P^{(j)}$ , and  $-q_j$  replaces the recoded multiplier digit  $y'_j$ . The entire preceding discussion of implementing multiplication applies to division as well and will not be repeated here. The important difference is that the multiplier digits  $y'_j$  were available, while the quotient digits  $q_j$  must be determined before the next partial remainder  $R^{(j)}$  can be computed.

The quotient digits  $q_j$  is determined by comparing the magnitude of the number  $R'$ , consisting of the first three digits and the temporary overflow position  $i=0$  of the partial remainder  $R^{(j-1)}$ , to the magnitudes of the numbers  $X'_0, X'_1, X'_2, X'_3$  consisting of the same digits ( $i=0, 1, 2, 3$ ) of the multiples  $X/2, 3X/2, 5X/2$  and  $7X/2$  of the normalized divisor  $X$  respectively. The magnitude of the quotient digit will be  $|q_j| = g$ , where  $g$  is the subscript of the least test number  $X'_g$  which satisfies the test condition:

$$|X'_g| > |R'|;$$

if all four test numbers fail to satisfy the above condition,  $|q_1| = 4$  is to be chosen. The sign of each nonzero  $q_i$  is chosen to be such that the sign of the term  $X q_i$  is the same as the sign of the partial remainder  $R^{(j-1)}$ .

The divisor  $X$  is required to be packed and normalized; in this case the analysis of the required precision of comparison<sup>1</sup> shows that the positions  $i = 4, 5, \dots, n$  may be disregarded in the choice of quotient digits. The initial condition which is to be satisfied by  $R^{(0)}$  in order to apply the division algorithm without quotient overflow is that before the first left shift

$$|R'| < |X'_0|$$

must hold, where  $R'$  consists of the digits  $i=0, 1, 2, 3$  of the number  $R^{(0)} - Xq_1$ . When  $|q_1| \leq 3$  is chosen, the above condition is known to be satisfied; the selection of  $|q_1| = 4$  indicates a possible quotient overflow. A convenient solution to the overflow problem in floating-point division (employing no right shifts) is to repeat the selection of  $|q_1|$  once more (before the first left shift) when  $|q_1| = 4$  is indicated. If  $|q_1| \leq 3$  is indicated, the correct magnitude of  $q_1$  is the sum of both indicated values, and the left shift is executed. If the second selection is again  $|q_1| = 4$ , then  $|q_0| = 1$  is recorded ( $q_0$  is the quotient overflow digit) and the procedure is continued until an indication of  $|q_1| \leq 3$  allows the first left shift. After this initial range adjustment, one quotient digit will be generated during each step of the division algorithm.

In the implementation of division, the multiplication hardware can be used for the required test subtractions which determine the quotient digits. Three digits of  $X/2$  (designated as  $X'_0$ ) are obtained by displacing all bits of the first three digits of the packed and normalized divisor  $X$  one binary position to the right. Since  $2X$  and  $3X$  are available, the other tests (if necessary) are performed by first obtaining  $X'_0 \pm R'$  (add if signs of  $X$  and  $R^{(j-1)}$  differ, subtract otherwise) and then separately adding to this result the required digits (positions  $i=0, 1, 2, 3$ ) of  $X, 2X$ , and  $3X$ . This method will yield one radix 8 quotient digit for every addition, employing the one-digit multiplication arrangement supplemented by the comparison circuitry.

Significant digit division<sup>3,8</sup> follows the same rules as multiplication, which was discussed in the preceding section.

### 8. Reconversion to Binary Forms

Several feasible methods exist for the reconversion of  $s-d$  numbers into the conventional form; the specific choice depends on the system relationships of the  $s-d$  arithmetic processor.

One method is the previously discussed repeated application of the pack-add algorithm. This method requires a variable number of steps, and is essentially a serial radix 8 borrow-propagation method with sensing of the completion of all borrow chains.

Another method of reconversion employs a borrow-propagation circuit which accepts the three positively weighted (+4, +2 and +1) bits of each radix 8 digit as bits of a binary operand and the negatively weighted (-8) bit as a binary borrow into the (+1) position of the octal  $s-d$  digit which is immediately to the left. Negative numbers will appear in "two's complement" form. An end-around borrow connection will give the result in binary "one's complement" form, which is readily converted to the sign-and-magnitude form. The "-1" circuits which add the negative transfer digit to the interim sum in every digit adder may be interconnected for this purpose, or a separate borrow circuit may be employed. Any desired amount of borrow-lookahead may be incorporated in this arrangement; borrow-completion sensing may also be used.

Finally, the  $s-d$  number may be entered as a pair of binary operands with opposite signs into a conventional binary adder when it is available in the computing system. The positive operand is composed of the positively weighted bits of all  $s-d$  digits, while the negative operand consists of the negatively weighted bits.

### 9. Conclusion

Signed-digit arithmetic is characterized by an affinity for variable-length operations with floating-point and significance-arithmetic options. Special carry-acceleration circuits are completely eliminated, and the space-zero value

implements some control functions at the level of individual digit adders instead of at a central control location. For instance, explicit information on the length of operands and results is not required. The logical complexity of individual digit adders is consequently increased. For example, a preliminary design of a radix 8 digit-adder with the separate transfer addition circuit indicates that it requires between 2 and  $2\frac{1}{2}$  times as many logic circuits as a ripple-carry radix 8 (three-bit) conventional adder. A special control circuit is also required to detect the presence of the space-zero digit values ( $\emptyset$ ) at adder inputs. The modular nature of digit-adders is expected to be especially suitable for microelectronic systems. One radix  $2^k$ , or radix 10 digit-adder offers a standard building block of considerable complexity which is suitable for micro-electronic implementation. An arithmetic processor with specified performance characteristics then will be constructed as an array of the standard building blocks. Each digit adder is an arithmetic unit of limited capability in the specified processor. The extent of central control functions is reduced and "one-of-a-kind" logic circuits are eliminated.

A second novel and potentially useful property of s-d arithmetic is the order in which the digits of results are produced for the elementary set of algorithms: addition, subtraction, multiplication and division. The most significant digits of the results always appear first and may be processed further without waiting for the less significant digits to be computed. Furthermore, only left shifts are employed in these algorithms, and the digits of the operands and results "flow" in one direction-to the left, while the rate at which they are produced depends on the number of digit adders which are available. Complex algorithms may now be implemented by an array of digit adders without the need for intermediate storage of results and in an asynchronous manner, with the space-zero values serving to indicate completion of the various elementary algorithms. Both aspects of s-d arithmetic which were mentioned above and methods of failure detection are now being investigated for potential application in the restructurable computer system.<sup>5</sup>

## 10. Acknowledgements

The Variable Structure Computer at UCLA<sup>5</sup> has provided the stimulus for the development of this paper. The author wishes to acknowledge many informative discussions with Professors G. Estrin and B. Bussell. Concurrent investigations of other types of signed-digit arithmetic by J. L. Drayer<sup>9</sup> and by R. W. Baker<sup>10</sup> have contributed supporting information on the feasibility of signed-digit arithmetic processors. A detailed study of the logic design of the binary-compatible s-d adder has been conducted by D. M. Kimble.<sup>11</sup>

1. AVIZIENIS, A., "Signed-Digit Number Representations for Fast Parallel Arithmetic," IRE Transactions on Electronic Computers *EC-10* (1961), 389-400.
2. AVIZIENIS, A., "On a Flexible Implementation of Digital Computer Arithmetic," *Information Processing 1962*, C. M. Popplewell, editor, (North-Holland Publishing Co., Amsterdam, 1963), 664-670.
3. METROPOLIS, N., and ASHENHURST, R. L., "Significant Digit Computer Arithmetic," IRE Transactions on Electronic Computers *EC-7* (1958), 265-267.
4. ESTRIN, G., "Organization of Computer Systems-The Fixed Plus Variable Structure Computer," Proceedings of the Western Joint Computer Conference 17 (1960), 33-40.
5. ESTRIN, G., BUSSELL, B., TURN, R., and BIBB, J., "Parallel Processing in a Restructurable Computer System," IRE Transactions on Electronic Computers *EC-12* (1963), 747-755.
6. MACSORLEY, O. L., "High-Speed Arithmetic in Binary Computers," Proceedings of the IRE, 49, No. 1., (January 1961), 67-91.
7. ROBERTSON, J. E., "A New Class of Digital Division Methods," IRE Transactions on Electronic Computers *EC-7* (1958), 218-222.
8. ASHENHURST, R. L., and METROPOLIS, N., "Unnormalized Floating Point Arith-

- metic," *Journal of the Association for Computing Machinery* 6 (1959), 415-428.
9. DRAYER, J. L., "Implementation of Signed-Digit Arithmetic," M. S. Thesis, University of California, Los Angeles, 1964.
  10. BAKER, R. W., "A Study of the Organization of a Signed-Digit Arithmetic Unit," M. S. Thesis, University of California, Los Angeles, 1964.
  11. KIMBLE, D. M., "Implementation of Binary-Compatible Signed-Digit Arithmetic in a Restructurable Computer System," M. S. Thesis, University of California, Los Angeles, 1964.

# A TRANSFLUXOR ANALOG MEMORY USING FREQUENCY MODULATION\*

*Walter J. Karplus and James A. Howard  
Department of Engineering  
University of California  
Los Angeles*

## I. INTRODUCTION

The accurate storage of a continuous voltage has always proven to be a difficult and challenging problem to the designers of electronic analog systems. Most modern analog computer installations include a number of "sample-hold" devices, utilizing high-quality operational amplifiers in combination with special electronic switching circuitry. In such units the voltage is stored as charge on a capacitor, so that leakage and grid currents must be extremely carefully controlled to permit long-time storage with high accuracy. Recently introduced hybrid computer systems have placed an additional requirement upon the analog memory unit: economy. For example, in the discrete-space-discrete-time hybrid computer system now under development at UCLA, 1,000 sample-hold circuits will be required in order to accommodate 1,000 parallel digital-analog channels. Furthermore, hold times of several minutes are desired. Under these conditions conventional capacitor-type analog memories become economically unfeasible. These considerations have stimulated a search for a rapid, accurate and economic analog memory and have resulted in the development of the FM-transfluxor unit described in this paper. Since their introduction by Rajchman<sup>1,2</sup> in 1955, multiaperture magnetic devices (MAD) have assumed an impor-

tant place as magnetic logic and memory devices in digital computer applications. The extension of this technique to analog systems has been proposed from time to time, but no fully satisfactory transfluxor analog memory has been described to date.

In essence, a transfluxor is a ferrite core with at least two holes—the major and the minor aperture. In memory applications, advantage is taken of the fact that an electrical signal applied to the major aperture effects a change in the magnetic field in the entire core. Provided certain geometrical criteria are satisfied, it is then possible to sense the magnetic condition of the core by means of an electrical signal applied to the minor aperture, without affecting the magnetic field about the major aperture. If it is desired to store analog variables, it is necessary to provide for the uninterrupted read-out of a continuous electrical voltage. This in turn necessitates that the sensing signal be applied continuously and that it be suitably modulated by the setting signal about the major aperture whenever a change in the stored information is desired. Suggested approaches to analog memories can be conveniently classified as amplitude-modulated, phase-modulated, or frequency-modulated, as determined by the manner in which the setting signal is made to affect the sensing signal.

\* This work was supported in part by the National Science Foundation under Grant G-24888.

In an amplitude-modulation system,<sup>2, 3, 4, 5, 6</sup> as illustrated in Figure 1a, a constant amplitude sinusoidal oscillator applies the sensing signal to the minor aperture. The sensing signal is picked up by a second winding about the minor aperture and is converted to DC by an amplitude-to-DC converter, consisting of an amplifier, a rectifier and a low-pass filter. The setting signal controls the coupling between the two windings about the minor aperture. The DC output voltage can therefore be varied continuously from a minimum (decoupled) to a maximum (fully coupled) value by means of suitable setting signals.

In the phase-modulated memory<sup>7</sup> shown in Figure 1b, the winding about the minor aperture constitutes an inductor and forms part of an RLC circuit. An oscillator is placed in series with this parallel circuit, so that the phase of the resulting sinusoidal signal is determined by the magnitude of the inductor. By means of a suitable phase-to-DC converter, the phase shift effected by a variation in the magnitude of the inductor is translated into a DC output voltage. The inductor magnitude is in turn determined by the magnetic field in the transfluxor core and is controlled by the setting signal.

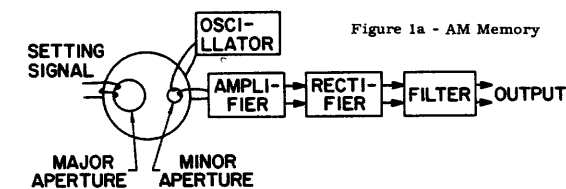


Figure 1a - AM Memory

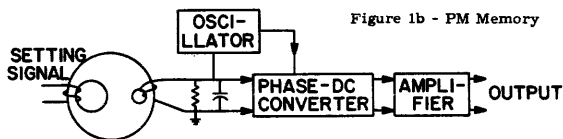


Figure 1b - PM Memory

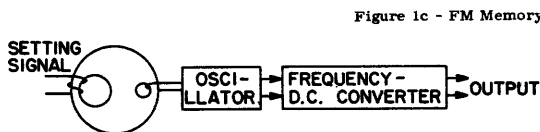


Figure 1c - FM Memory

Figure 1. Transfluxor Analog Memory Systems.

Figure 1a. AM Memory.

Figure 1b. PM Memory.

Figure 1c. FM Memory.

In the frequency-modulation approach<sup>3, 8, 9</sup> described in this paper, a winding about the minor aperture forms part of an oscillator and controls the oscillator frequency. A frequency-to-DC converter is then employed to translate this frequency into a DC output. Since the magnetic field in the transfluxor controls the oscillator frequency, the setting signal can be employed to vary the DC output voltage over a wide range.

All of the memory circuits described above feature an essentially unlimited memory time. Provided environmental extremes are avoided, the drift of the magnetic field within the transfluxor from the value imposed by a setting signal is essentially negligible over a period of weeks and months. Hence, the DC output voltage of the transfluxor memory is likewise constant over a long period of time. An analog memory can then be fashioned simply by causing the analog input voltage to be translated into a suitable setting signal, which in turn produces a proportional DC output voltage. This mode of operation may be termed "open-loop" and has the advantage of simplicity. If a large number of transfluxor memory channels are to be used, however, open-loop operation makes it necessary to calibrate each channel separately to take into account minor variations in transfluxor characteristics. It is then also necessary to assure that the transfluxor temperature at the time of operation is identical to the temperature at the time when the transfluxor was calibrated. These inconvenient features can be overcome by employing a "closed-loop" mode as illustrated in Figure 2. Here the DC output voltage is compared to the DC input voltage and the resulting error signal is made to activate a drive circuit. Setting pulses are then applied to the major aperture until the error voltage falls within prescribed limits. The drive circuit can then be disconnected from the memory unit without affecting the DC output. This facilitates time-shared operation.

In the course of the past two years, all three modes of transfluxor memory operation have been analyzed experimentally and theoretically. On the basis of these studies it was determined that the frequency-modulation approach offers the greatest accuracy, economy and freedom

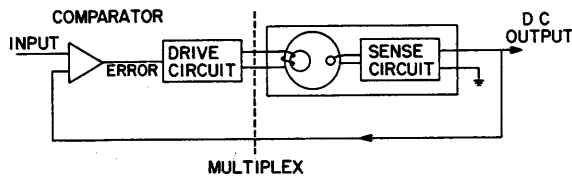


Figure 2. Transfluxor Closed-Loop Memory.

from noise effects. Accordingly, an FM-transfluxor memory, operating in a closed-loop fashion, has been developed and tested. The overall features sought in this unit can be summarized as follows:

- Nominal accuracy: 0.1% of half-scale
- Memory speed: 1 millisecond
- Dynamic range: -10 volts to +10 volts
- Facility for multiplexing
- All solid state
- Parts cost per module: less than \$5.00

## II. SYSTEM OPERATION

A simplified diagram of the closed-loop FM analog memory system is shown in Figure 3 and is seen to include two major units: a) the transfluxor and sensing circuitry, which is provided for each analog channel, and b) the comparator and drive circuitry, which is time-shared among all the transfluxor modules. It can be seen that the memory modules are very simple in design, containing only three transistors and one diode in addition to passive components including the transfluxor itself. The nature of each of the major elements in the memory system will now be described.

*Transfluxor:* The prototype transfluxor configuration currently employed in the memory is shown in Figure 4a. The core is fashioned from a disc of Indiana General type S-6 ferrite material, 25.55 mm in diameter and 1.57 mm thick. The S-6 type material was selected primarily because of the commercial availability

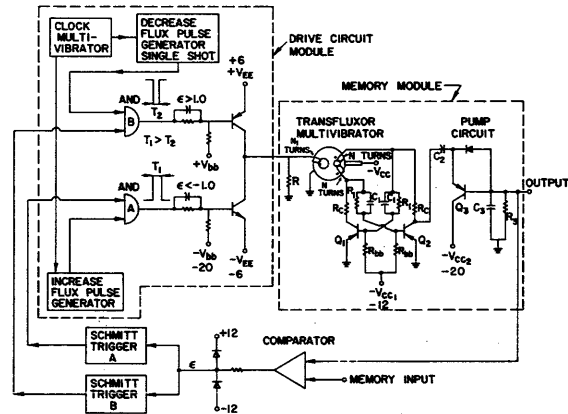


Figure 3. Simplified Schematic of Transfluxor F-M Memory System.

$Q_1, Q_2, Q_3 - 2N1309; R_1 = 390K; C_1 = 500 \mu\mu f;$   
 $R_c = 100\Omega; R_{bb} = 33K; C_2 = 30 \mu\mu f; C_3 = .05 \mu\mu f;$   
 $R_2 = 38K.$

of the 25.55 mm diameter disc employing this material. In addition, as evidenced by the B-H characteristics of Figure 4b, the low threshold coercive force of 0.2 oersteds of the S-6 material minimizes the power requirements of the transfluxor driving circuitry. An ultra-sonic drill is used to fashion the major and minor

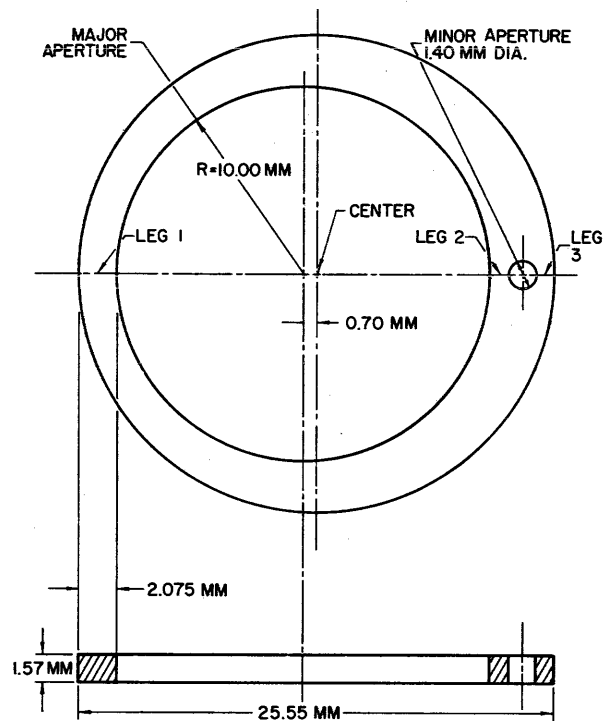


Figure 4a. Prototype Transfluxor Geometry.

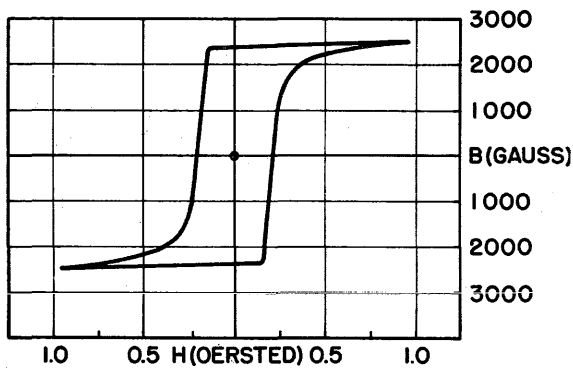


Figure 4b. Prototype Transfluxor Magnetic Characteristics.

Initial Permeability (IMC) = 110  
 Maximum Permeability ( $D_c$ ) = 6000  
 Saturation Flux Density ( $D_c$ , gauss) =  
 3450 at  $H_m = 25$  oersteds  
 Coercive Force ( $D_c$ , 60 cycle) = 0.20  
 Maximum  $B_r/B_s$  Ratio = 0.95 at 1  
 oersted

apertures which are 20 mm and 1.4 mm in diameter respectively. The disc is therefore quite large when compared to memory cores used in digital computer applications. This large size provides greater temperature stability and facilitates the application of the setting and the sensing windings. The major aperture winding consists of 75 turns of #34 wire. The number of turns on the minor aperture are given below. The theory underlying specification of the transfluxor geometry is summarized in Appendix A.

*Sensing Circuit:* The sensing winding in the transfluxor forms a portion of an oscillator circuit. Both sinusoidal and switching-mode oscillators were investigated and the latter selected for their greater stability. The control of multivibrator frequencies using ferrite cores has been described by Royer<sup>10</sup> and others.<sup>11, 12, 13, 14</sup> These units employ two transfluxor windings, the coupling coefficient between them determining the frequency of operation. The circuit shown in Figure 3 is a modified form of this type of multivibrator. In an alternative realization of the memory module, a multivibrator circuit similar to that

shown in Figure 5 has been used with considerable success. Here a single minor aperture winding is connected between the collectors of two transistors. The operation of transfluxor controlled multivibrators is discussed briefly in Appendix B. The output of the multivibrator is converted to a DC voltage, directly proportional to the frequency by means of a so-called "pump"<sup>15</sup> circuit.

*Comparator:* A high-quality chopper-stabilized, DC amplifier is employed to compare the analog input voltage to the analog output voltage. The difference between these two values is amplified 100 times and is identified as the error signal  $\epsilon$ . Not shown in Figure 3 are auxiliary circuits employed for closed-loop stabilization, to accelerate the response of the pump circuit to error signals, and to shift the DC output into the desired range. A considerable reduction in output impedance can be attained by the inclusion of an additional transistor.

*Driving Circuit:* It is the function of the driving circuit to translate the error signal into suitable setting pulses. Numerous experiments indicated that short voltage pulses constitute the most effective manner of controlling the magnetic field in the transfluxor. The optimum pulse-width for effecting increases in multivibrator frequency were found to be considerably larger than the optimum pulse-width for decreasing multivibrator frequencies. In order

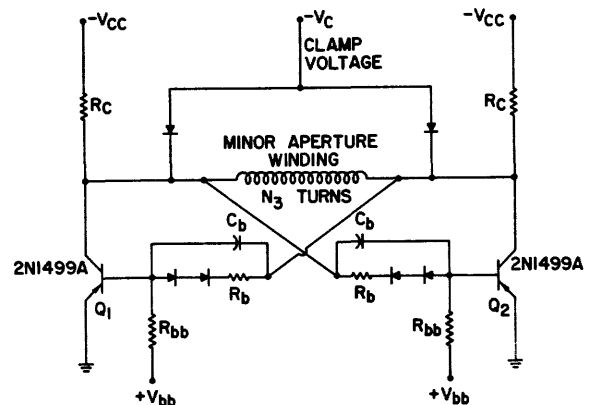


Figure 5. Transfluxor Multivibrator with Single Minor Aperture Winding.

$V_{cc} = -20$  Volts;  $V_c = -6$  Volts;  $V_{bb} = 20$  Volts;  
 $R_c = 1K$ ;  $C_b = 120 \mu\mu f$ ;  $R_b = 1.5K$ ;  $R_{bb} = 200K$ .



to facilitate the efficient traversing of a frequency range from 250 to 900 KC, separate "forward" and "backward" channels are employed.

A clock multivibrator generates a square-wave of constant frequency. This unit drives two separate monostable multivibrators A and B which produce trains of pulses with widths of 1.30 and 0.65 microsecond respectively. The former of these two pulse trains is applied to gate A which produces an output only if the error voltage is more negative than  $-1.0$  volts. Similarly, gate B permits the output of the single shot B to be transmitted only if the error voltage  $\epsilon$  is more positive than  $+1.0$  volt. Depending upon the polarity of the error signal, therefore, positive pulses 1.30 microsecond in width or negative pulses 0.65 microsecond in width are applied to the setting winding of the transfluxor. The optimum frequency range and pulse-width depend of course upon a number of considerations and can be expected to vary from application to application.

The overall operation of the memory unit can be summarized as follows: Prior to the application of an input signal, the frequency of oscillation in the sensing circuit is determined by the magnetic condition of the transfluxor unit. This frequency, in the range 250 KC to 900 KC results in a corresponding DC voltage in the range  $-10$  volts to  $+10$  volts. The driving circuit is now connected using two multiplexer switching poles, and a DC input signal in the range  $-10$  to  $+10$  volts is applied. Any difference between the DC output of the memory module and the applied input results in error voltage  $\epsilon$ . This error signal causes a train of positive or negative voltage pulses to be applied to the setting winding of the transfluxor. The effect of these pulses is to change the frequency in the sensing circuit until the difference between the output voltage and the input voltage becomes less than 5 millivolts. The drive circuit then becomes inoperative and can be switched to another memory module. Of particular interest in characterizing the performance of such a memory unit is the long-term stability of the memory module and the resolution or accuracy attainable. These aspects are considered in the following sections.

### III. STABILITY CHARACTERISTICS

The magnetic characteristics of square-loop ferrite materials are strongly affected by temperature. The coercive force, the saturation flux density and the remanant flux density all decrease in magnitude as the temperature of the core is increased.<sup>16</sup> In evaluating the effectiveness of transfluxors as analog memories, the effect of temperature variations upon memory operation must be carefully considered.

Of primary interest is the drift in the sensing circuit frequency and hence in the DC output, when a memory module is set to a specific output voltage and subjected to temperature variations typical of a laboratory environment. Accordingly, a series of such long-term drift experiments were conducted, with the transfluxor at different levels of saturation. The transfluxor and sensing circuit were placed in an air-conditioned laboratory without special efforts at temperature stabilization and the drift in sensing circuit frequency as a function of temperature and time was recorded. In the course of a typical three-hour run the laboratory ambient air temperature varied from 73 to 75°F. The results of these tests for three major-aperture flux states are summarized in Figure 6. As shown, the percent drift over a three hour or greater interval is in the order of 0.1% for all flux states. It should be recog-

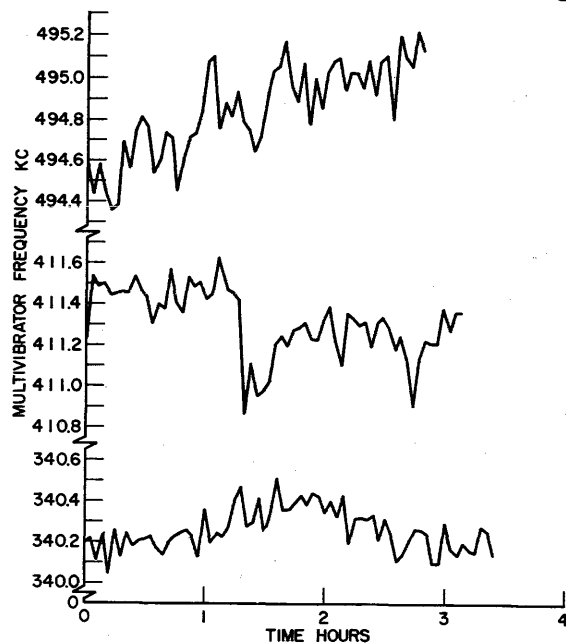


Figure 6. Long-Time Drift Characteristics of Transfluxor Multivibrator for Various Setting Conditions.

nized that in any specific application, the transfluxor memory is required only to maintain a constant output between successive analog inputs. In most hybrid computer systems, holding-time requirements rarely exceed one minute and are usually of the order of several seconds.

Additional series of experiments were conducted to determine the general sensitivity of the transfluxor circuit to temperature. Accordingly, the transfluxor core was placed in an oven and the variation in sensing circuit frequency was observed over a temperature range from  $-80^{\circ}\text{F}$  to  $+150^{\circ}\text{F}$ . These measurements were repeated with the transfluxor set to five different levels of saturation and are illustrated in Figure 7. In the course of these experiments it was observed that for each specific transfluxor geometry there exists a critical temperature beyond which irreversible effects take place. That is, if the critical temperature is exceeded information stored in the transfluxor memory becomes essentially lost. The reason for this effect lies in the fact that the coercive force<sup>17</sup>  $H_c$  decreases with frequency. If this decrease becomes large enough, it then becomes possible for signals about the minor aperture to switch flux about the major aperture. Such major aperture flux switching results in turn in a change in the sensing circuit frequency. This effect must be carefully avoided by suitable design of the transfluxor and by suitable

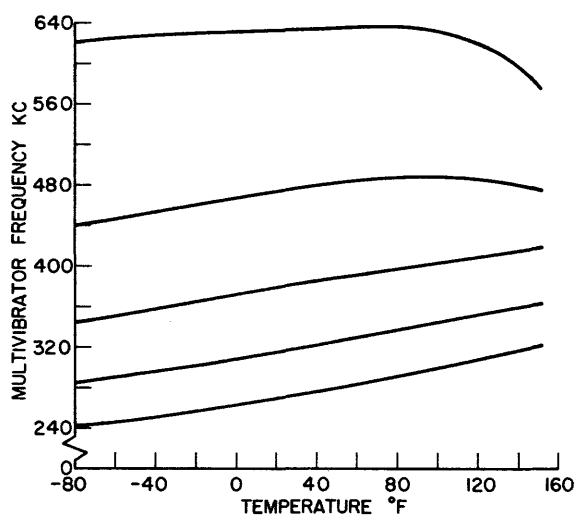


Figure 7. Frequency—Temperature Characteristics of Transfluxor for Several Major-Aperture Setting Conditions.

choice of the resistor  $R$  placed in parallel with the setting winding.

#### IV. DRIVING CHARACTERISTICS

The frequency of the sensing circuit, which determines the DC output, is varied and controlled by means of trains of positive or negative voltage pulses applied to the setting winding. In order to achieve a high resolution and hence a high accuracy, it is important that the increment in frequency produced by a single pulse be very small. For example, if the frequency is to range from 250 KC to 900 KC, it is necessary that a single pulse produce a frequency change of less than 650 cycles, in order that a 0.1% resolution be obtained. The problem of selecting an optimum pulse-width and repetition rate is complicated by the fact that the effect upon the frequency of a setting pulse of a given width is different when the frequency is being increased than when it is desired to decrease the frequency. This is illustrated in Figure 8 for a number of different pulse-widths. In each case the core was first saturated (high frequency), and a series of rectangular voltage pulses were applied one at a time so as to decrease the frequency to its minimum value. The polarity of the pulses was then reversed and the core brought gradually back to the saturated state. From these data it is apparent that for increasing frequencies the pulse-width should be at least twice as great as for decreasing frequencies. These curves also point out the desirability of oper-

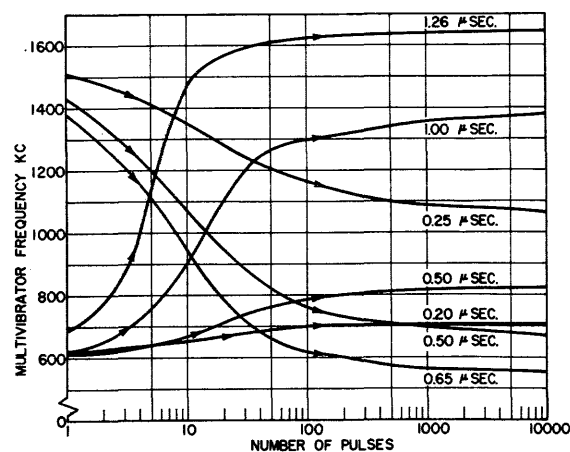


Figure 8. Typical Frequency—Resolution Characteristics of Transfluxor Multivibrator.

ating in the relatively linear low-frequency range rather than over the entire frequency domain.

The speed of response of the memory circuit is determined by a number of design parameters and compromises. Of particular importance is the frequency of the clock multivibrator in Figure 3, since this determines the rate at which control pulses are applied to the setting winding. The width of the control pulses is likewise important in determining the speed of operation, since wider pulses produce larger increments in frequency. A compromise between speed and accuracy must hence be made. The RC time constant at the output of the pump circuit is also important in determining the speed. Here the compromise is between speed and output ripple, since the low-pass filtering action of the pump output circuit varies directly with the time constant. The switching characteristics of the transfluxor core itself appear to have a relatively minor effect upon the response time of the memory unit.

## V. CONCLUSION

On the basis of experience with a number of prototype memory systems, it is concluded that the FM transfluxor analog memory offers unusual promise. The memory module is exceptionally simple, compact and economic, permitting the design of systems with large numbers of sample-hold channels. The usefulness of this type of memory in the hybrid application is further enhanced by the fact that the memory time under normal laboratory conditions is very long, permitting a great flexibility in digital-analog scanning cycles. Since the driving circuitry is time-shared among all the memory modules, great care and sophistication can be employed in its final designs, so as to attain an optimum compromise between response, speed and accuracy. A second potentially useful application of the transfluxor memory is in pure analog track-hold circuits. Here the low cost of the individual modules permits the utilization of a relatively large number of them to store a continuous function of time as a large set of discrete points.

*Acknowledgments:* The authors gratefully acknowledge the helpful suggestions offered by Mr. M. Palevsky of Scientific Data Systems,

Inc. and by Mr. G. Bachand of Sandia Corporation. Mr. P. Kibby, G. Keludjian and S. Ratner, students at U.C.L.A., contributed interesting experimental data.

## APPENDIX A

### Transfluxor Geometry and Driving Considerations

#### Major and Minor Aperture Signal Constraints

If the core material of a transfluxor has a magnetizing force  $H_c$ , the coercive force for any closed flux path j-k is

$$N_j I_{j-k} = H_c \bar{l}_{j-k} \quad (1)$$

where

$N_j$  = Number of turns around leg j

$\bar{l}_{j-k}$  = Mean circumference of path j-k

$I_{j-k}$  = Current required to establish coercive force for path j-k

Equation (1) can be used to establish the constraints imposed on the major and minor aperture signals by the geometry and material properties of the transfluxor. The range of allowable adjustment for the major aperture control signal in the oscillator mode of operation (Figure 9a) is limited by the flux path lengths  $l_{1-2}$  and  $l'_{1-2}$  of Figure 9c. Applying

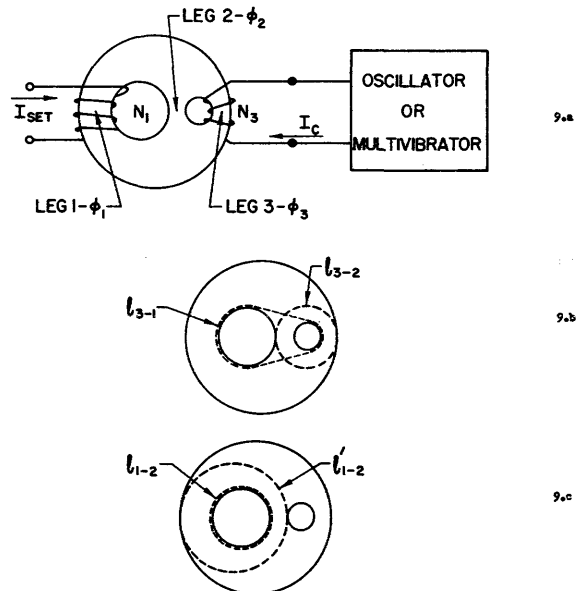


Figure 9. Transfluxor Multivibrator Mode of Operation.

Figure 9a. Oscillator Mode of Operation.

Figure 9b. Minor Aperture Path Lengths.

Figure 9c. Major Aperture Path Lengths.

Equation (1), with the assumption that the threshold mmf of the transfluxor material is  $H_{co}$ , yields

$$H_{co} l_{1-2} \leq N_1 I_{SET} \leq H_{co} l'_{1-2} \quad (2)$$

ampere turns

where

$H_{co}$  = Threshold mmf of transfluxor material in oersteds

$l_{1-2}, l'_{1-2}$  = Path lengths shown in Figure 9c in centimeters

$I_{SET}$  = Major aperture setting current — amperes

$N_1$  = Number of turns on major aperture

The path lengths  $l_{1-2}$  and  $l'_{1-2}$  can be regarded as the basic geometric parameters for the transfluxor since their ratio establishes the setting range of the major aperture.

$$\frac{l'_{1-2}}{l_{1-2}} \geq \frac{I_{SET \text{ MAX}}}{I_{SET \text{ MIN}}} \quad (3)$$

Similar constraints can be established for the minor aperture using the path lengths  $l_{3-2}$  and  $l_{3-1}$  of Figure 9b.

$$H_{co} l_{3-2} \leq N_3 I_c \leq H_{co} l_{3-1} \quad (4)$$

where

$l_{3-1}, l_{3-2}$  = Path lengths in Figure 9b in centimeters

$N_3$  = Number of turns on minor aperture

$I_c$  = Collector current of saturated or on transistor in multivibrator — amperes

The ratio of the path lengths  $l_{3-1}$  and  $l_{3-2}$  establishes the range of satisfactory adjustment for  $I_c$  according to

$$\frac{I_c \text{ MAX}}{I_c \text{ MIN}} \leq \frac{l_{3-1}}{l_{3-2}} \quad (5)$$

As indicated by Equations (4) and (5), the multivibrator parameters should be adjusted so that the maximum value of  $I_c$  is greater than that value of current necessary to just magnetize the unsaturated path  $l_{3-2}$ , but smaller than that value which would magnetize an unsaturated path  $l_{3-1}$ . These limit values should be separated by as large a margin as possible in a transfluxor to provide adequate isolation between the major and minor apertures.

For complete minor aperture saturation, a considerably higher value of magnetizing force than the threshold value  $H_{co}$  is required. Designating this magnetizing force by  $H_{cs}$  Equations (4) and (5) become:

$$H_{cs} l_{3-2} \leq N_3 I_c \leq H_{co} l_{3-1} \quad (6)$$

$$\frac{I_c \text{ MAX}}{I_c \text{ MIN}} \leq \frac{l_{3-1} H_{co}}{l_{3-2} H_{cs}} \quad (7)$$

Equations (6) and (7) place a severer restriction on the adjustment range of  $I_c$ , particularly if  $H_{cs} \gg H_{co}$ . Consider, for example, the prototype transfluxor used in the present research. The parameters of Equations (6) and (7) for this transfluxor are  $l_{3-1} = 6.50$  cm,  $l_{3-2} = 1.10$  cm,  $H_{co} = 0.2$  oersteds and  $H_{cs} = 1.0$  oersted. Using these values Equation (6) becomes

$$1.10 \leq N_3 I_c \leq 1.30 \text{ ampere turns} \quad (8)$$

Since Equation (8) satisfies the conditions of Equation (6), flux switching of the major aperture by the minor aperture should not occur. In the event that flux switching of the major aperture by the minor aperture does become a problem, the path length  $l_{3-1}$  must be increased. This can be accomplished by re-dimensioning the transfluxor; but this would lead to a reduction in the frequency range. A more practical approach is to place a resistance across the major aperture winding, which acts effectively to increase the path length  $l_{3-1}$  by decreasing the major aperture field available for flux switching.

#### Major Aperture Driving Characteristics

The primary requirement of the major aperture driving signal is that it be capable of changing the flux state of the aperture by very small increments so that the frequency of the minor aperture oscillator is also changed in correspondingly small increments. The accuracy of the memory is a direct function of the number of quantized flux steps  $M_i$  into which the major aperture flux range  $\Phi_1$  can be divided. For an ideal square-loop core material the number of flux increments is given by

$$M_i = \frac{N_1 \Phi_1}{V_s T} \quad (9)$$

where

$N_1$  = Number of turns on leg 1 of major aperture

$V_s$  = Amplitude of a rectangular setting pulse of width T applied to the major aperture

Unfortunately, the actual major aperture  $\emptyset$  - NI characteristics are not square but form a distorted hysteresis<sup>18</sup> loop due to the finite inner-diameter outer-diameter ratio of the major aperture.

Hawkins and Munsey<sup>19</sup> have analyzed the effects of core geometry on the driving signal and core-flux state of a toroidal core. Although their analysis is restricted to a toroidal shape, it can be extended to any closed-magnetic-path geometry possessing flux paths of different lengths such as the major aperture of a transfluxor. The primary result of this study is given by

$$\emptyset = \emptyset_x (1 - e^{-M_i/\tau}) \quad (10)$$

where

- $\emptyset$  = Flux stored in major aperture core
- $\emptyset_x$  = Lumped constant which is a function of applied volt-time integral and core geometry
- $M_i$  = Number of input voltage pulses applied to the core (in this case the main aperture)
- $\tau$  = Time constant which is a function of the core geometry and material properties

Equation (10) applies until  $\emptyset$  reaches saturation for the given core  $\emptyset_s$  and can be plotted as shown in Figure 10 with  $\emptyset = \emptyset_s$ . As shown, improved linearity can be obtained by making  $\emptyset_x$  large with respect to  $\emptyset_s$ . However this improvement in linearity must be traded for resolution since fewer pulses  $M_i$  are required to saturate the core as  $\emptyset_x$  is increased.

A second significant result of the analysis is an expression for the maximum number of pulses  $M_i$  as a function of core magnetic properties and geometry of the form

$$M_{i \text{ MAX}} = \frac{1/S_r}{\text{Ln} \left( \frac{r_o}{r_i} \right)} \text{Ln} \left[ \frac{1}{1 - P} \right] \quad (11)$$

where

$S_r$  = Squareness ratio of the core material

$r_o, r_i$  = Outer and inner radius of core (major aperture) respectively

P = Linearity and is defined as the fraction by which an incremental flux change at the maximum value of  $M_i$  differs from that at  $M_i = 0$ .

The quantity P may also be interpreted as the relative flux change at the worst point (near saturation) on the hysteresis loop.

As shown in Equation (11) for any desired value of P the maximum value of  $M_i$  is determined by core magnetic properties and geometry only. In particular Equation (11) indicates that the closer ( $r_o/r_i$ ) approaches unity the greater the resolution. Thus, ideally the transfluxor major aperture should be structured so that  $r_o/r_i$  is as close to unity as possible.

Also as nonlinearity increases (i.e.  $p \rightarrow 1$ ) the resolution increases. This is advantageous for the frequency mode of operation since the DC voltage output of the memory is inversely proportional to the minor aperture flux available for switching—(memory output voltage  $\sim 1/\emptyset$ ). The  $1/\emptyset$  dependence makes the memory output voltage a hyperbolic function of the major aperture control signal. By suitable control and choice of P it is possible simultaneously to improve resolution and to make the memory output linearly dependent on the major aperture control signal.

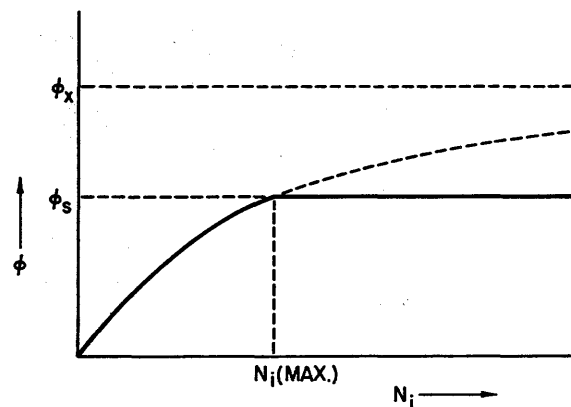


Figure 10. Variation of Major Aperture Stored Flux as a Function of Setting Resolution.

## APPENDIX B

## Transfluxor Multivibrators

*Two-Winding Oscillator*

The circuit diagram for the two-winding transfluxor multivibrator is shown in the system diagram of Figure 3. The basic operation of the unit is discussed in detail in references 13, 14 and 20, and consequently will not be described here. Two windings each with  $N$  turns are wound around the minor aperture, and the winding with  $N_1$  turns is placed around the major aperture. A control signal applied to  $N_1$  changes the switching flux of the multivibrator causing the frequency of oscillation to be a function of the magnetomotive force in the minor aperture windings  $N$ . If the major aperture is demagnetized the multivibrator operates at minimum frequency. When the flux state of the major aperture is equal to the saturation flux of the minor aperture by applying a control signal of sufficient magnitude, the multivibrator operates on remnant flux alone at maximum frequency.

Using the prototype transfluxor described above with 25 turns and a supply voltage of 12 volts, a frequency range of 250 KC to 1.0 MC was obtained for the circuit of Figure 3.

*Single Winding Multivibrator*

The single winding oscillator of Figure 5 permits a reduction in number of minor aperture windings and provides closer control over

the collector current. As indicated in Figure 5, the single winding multivibrator is actually a clamped flip-flop with the single minor aperture winding  $N$  connected between the collectors of the transistors  $Q_1$  and  $Q_2$ . This circuit has yielded satisfactory results under widely varying conditions. For example with a clamp voltage  $V_c = -6$  volts and  $N = 20$  turns on the transfluxor minor aperture, a six volt square wave output was obtained over a 100 KC to 1 MC frequency range. Typical output waveforms are shown in Figure 11.

*Frequency of Oscillation*

An expression for the frequency of oscillation of the single or two-winding multivibrator can be obtained by applying the definite integral form of Faraday's law to the transfluxor minor aperture winding. This operation yields the following expression for the frequency  $F$ —

$$F = \frac{V}{4NB_sA} \quad (12)$$

where:

$V$  = Voltage across the minor aperture winding volts

$N$  = Number of turns on single minor aperture winding

$B_s$  = Saturation flux density of transfluxor core material—webers/meter<sup>2</sup>

$A$  = Cross-sectional area of minor aperture leg assuming the areas of legs 2 and 3 are equal—square meters

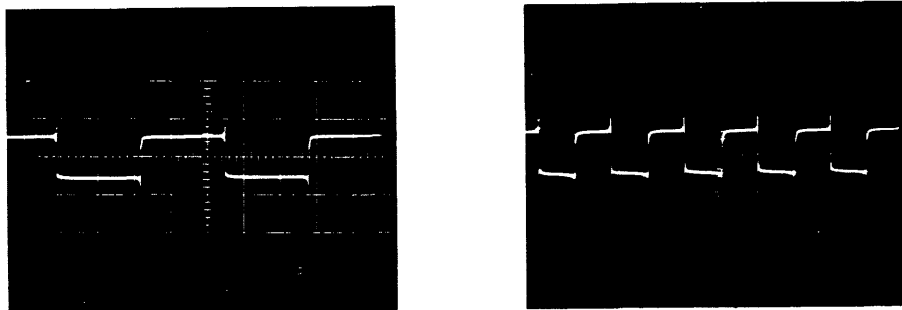


Figure 11. Typical Output Waveforms of Single Winding Transfluxor Multivibrator.

Figure 11a. Output Waveform From  $Q_2$  Collector; Frequency = 100KC; Scale; Amplitude = 5 Volts/CM; Time = 2  $\mu$  sec/CM.

Figure 11b. Output Waveform From  $Q_2$  Collector; Frequency = 500KC; Amplitude = 5 Volts/CM; Time = 1  $\mu$  sec/CM.

Equation (12) is an approximate expression for the minimum frequency of the multivibrator in the demagnetized state. Although the expression is idealized in the sense that it does not include the effects of such factors as switchover time, it is useful in estimating the minimum frequency of oscillation. For example, for the two-winding oscillator with  $V = 12$  volts,  $N = 25$  turns,  $A = 1.61 \times 10^{-6}$  square meters and  $B_s = 0.345$  webers per square meter Equation (12) gives 220 KC for the minimum frequency as compared to the measured value of 250 KC. A similar calculation for the single winding oscillator with  $N = 20$  turns and  $V = 6v$  yields 90 KC for the minimum frequency as compared to 100 KC for the measured value.

## REFERENCES

1. RAJCHMAN, J. A., and LO, A. W., "The Transfluxor—A Magnetic Gate with Stored Variable Setting," RCA Review, Vol. 16, pp. 303–311 (1955).
2. RAJCHMAN, J. A., and LO, A. W., "The Transfluxor," Proceedings IRE, Vol. 44, No. 3, pp. 321–332 (March 1956).
3. LI, K., "Application of the Continuous Range Properties of the Transfluxor for Control Purposes," RAC Report No. 1162 (December 30, 1957).
4. BOYAJIAN, C. F., "Transfluxor Analog Memory," Proceedings of Conference on Non-linear Magnetics and Magamps, September 23–25, 1959, Washington, D. C., pp. 313–323.
5. PALEVSKY, M., SLOCOMB, G., and BLOCK, D., "Digital to Analog Conversion for PCM Telemetry and Digital Data Transmission Systems," IRE Convention Record, Part 5, p. 158 (March 1959).
6. KRAUS, MURRAY, "Transfluxor Analog Memory," Control Engineering, Vol. 6, No. 12, p. 127 (December 1959).
7. HAASS, GUNTHER F., "Der Transfluxor als Analogspeicher," Nachrichtentechnische Zeit, Vol. 14, pp. 410–415 (August 1961).
8. WILKEN, A. C., "Interim Report on a Multivibrator Flux Interrogator with Memory," Sandia Lab Technical Memoranda, Report 46–63 (24), (April 1963).
9. COREY, F. B., "A Tunnel Diode Transfluxor Oscillator with Analog Memory Properties," Sandia Lab Technical Memoranda, Report SCTM 140–163 (24), (August 1963).
10. ROYER, G. H., "Switching Transistor D–C to A–C Converter," Transactions AIEE, Vol. 74–1, pp. 322–324 (1955).
11. UCHRIN, G. C., and TAYLOR, D. O., "New Self-Excited Square-Wave Transistor Power Oscillator," Proc. IRE, Vol. 43, p. 99 (January 1955).
12. VAN ALLEN, R. L., "Variable Frequency Magnetic Coupled Multivibrator," AIEE Transactions, Part I, Comm. and Electronics, Vol. 74, pp. 356–361 (July 1955).
13. MEYERHOFF, A. J., and TILLMAN, R. M., "High-Speed Two Winding Transistor Magnetic-Core Oscillator," IRE Wescon Convention Record, pp. 106–114 (1957).
14. MEYERHOFF, A. J., Digital Applications of Magnetic Devices, John Wiley and Sons, Inc., New York, New York, pp. 475–480 (1960).
15. BURTON, P. L., and WILLIS, J., "Unusual Transistor Circuits," Wireless World, pp. 107–110 (March 1958).
16. PASNAK, M., and LUNDSTEN, R. H., "Effects of Temperature on Magnetic Core Materials," Electrical Manufacturing, (October 1959).
17. ABBOT, H. W., and SURAN, J. J., "Temperature Characteristics of the Transfluxor," Proc. IRE, pp. 113–119 (1957).
18. ROBERTS, R. W., and VAN NICE, R., "Influence of ID–OD Ratio on Magnetic Properties of Toroidal Cores," Elect. Eng., Vol. 74, pp. 910–914 (October 1955).
19. HAWKINS, J. C., and MUNSEY, C. J., "A Magnetic Integrator for the Perception Programs," Aeronutronics Computer Operations Research Lab Publication No. 592, Quarterly Report No. 2, July 1–September 20, 1959.
20. STRAUSS, LEONARD, *Wave Generation and Shaping*, McGraw-Hill Book Co., Inc., New York, pp. 410–413 (1960).





# THE USE OF A PORTABLE ANALOG COMPUTER FOR PROCESS IDENTIFICATION, CALCULATION AND CONTROL

*L. H. Fricke and R. A. Walsh*  
*Monsanto Company*  
*St. Louis, Missouri*

In the design and development of controlled processes there are two areas of intense activity—the theoretical simulation of the total plant and empirical simulations involving the collection of reliable experimental data to assist in the construction of a special purpose model. The simulation of a newly proposed process, even from the best available theoretical basis, is usually only approximate. It requires use of large and expensive computer installations, either analog and/or digital, so that by direct programming of the design criteria, the optimum plant operating conditions may be determined. However, even if such a model were extremely reliable (pilot plant), the scale-up problems are quite complex. In some cases, it might be well-nigh impossible to maintain the exact relationships between certain intrinsic parameters, surface tension, heat transfer, etc.—and, of course, few theoretical models can anticipate all the significant process characteristics. As a result, many full scale plants are in need of partial redesign. The authors feel that techniques relying on plant data are indispensable for accomplishing this end.

It is the purpose of this paper to review some of the present in-plant techniques employed to determine both the static and dynamic characteristics and show how the results are used to select the correct process revisions

and achievable controller schemes for optimum operation. In particular, a newer method employing a portable analog computer to first determine system characteristics is described. Following an analysis of these results the method of reprogramming the computer to the proper controller characteristics is given.

To make the example realistic and to avoid the criticism that the methods are impractical, a real system was studied and made part of the discussion.

It is important to take a wide view to all aspects of the work here described. Thus the methods of procuring the basic data are simple and generally applicable to any physical system. The procedure of reducing data and the representation of dynamic behavior as a Laplacian transfer function is both routine and commonplace to electronics or servomechanism engineers. These methods are easy to use and are equally applicable to any linear system and any nonlinear system within the bounds of the perturbations of interest.

## DATA COLLECTION SYSTEMS

The heart of any experimental effort is the data collection apparatus. The sensing elements must be essentially free of any nonlinearities and dynamic delays so that a clear, accurate

diagnosis may be had. It is our experience that the best type of data collecting system (where possible) is multi-channel recording oscillographs coupled with very fast strain-gage transducers. Some of the distinct advantages of direct-writing analog recording are:

1. The data collection system provides easily adjusted signal conditioning such as filtering, suppression, and amplification.
2. The collection system provides signals that can be easily utilized in continuous analog computer calculation such as material balance, heat balance, etc.
3. The quality of both static and dynamic process data can be evaluated immediately.
4. The intuition of the testing engineer is strongly reinforced through the simultaneous observation of cause and effect.
5. Information which was not anticipated at the time of the tests is frequently discovered through subsequent studies of the records.
6. Malperforming process components or systems can frequently be located quickly by visual inspection of the recordings.

The ideal data system is one that combines the virtues of both analog recording and digital storage—the first to obtain information from a limited number of signals, the second to provide complete coverage and ease the burden of excessive data handling and computation.

#### SYSTEM IDENTIFICATION METHODS

Excluding those cases where the desired relationships can be derived completely by mathematical analysis, it is necessary to arrive at process descriptions by the analysis of input-output pairs. To determine these relationships one must make a correlation between the variables of interest under normal operating conditions. One way of obtaining such a correlation is to force a pertinent independent variable and observe the changes of its output. Another is to examine statistically auto and cross correlations of the noise spectra of perhaps many inputs and their respective outputs. This latter technique is extremely time consuming and although is theoretically superior for the analy-

sis of multi input-output systems, is definitely not recommended in practice. To date, few practical techniques have been developed for the process industries.

One of the earliest methods of system characterization was done by introducing sinusoidal fluctuations on the independent variables and measuring amplitude and phase ratio between the output and input for the series of frequencies of interest. From this a linear description can be obtained in the form of a Bode plot or Laplacian transfer function. However, this method is found to be objectionable to the process industries for two main reasons—first, the order of delays and time constants may be as long as 2 or 3 days and the measurement of sufficient data points could very well take weeks to several months and second, the perturbations necessary to obtain a good signal to noise ratio might very well tend to undulate the process deleteriously for too long a time.

Another of these methods, and one recommended in practice by J. O. Hougén, is the pulse technique. In this procedure, independent variables are disturbed with a closed input pulse of rectangular, displaced cosine, ramp or any other shape that will have sufficient harmonic content to excite the pertinent frequencies of interest. The unknown particular output is recorded simultaneously with one driven input. These input-output pairs are then reduced to a frequency response form by numerical evaluation of the ratio of Fourier transforms. Thus the performance function relating one pair is—

$$PF = \frac{-\int_{-\infty}^{+\infty} e^{-j\omega t} f(t)_{out} dt}{-\int_{-\infty}^{+\infty} e^{-j\omega t} f(t)_{in} dt}$$

which will yield an amplitude ratio and phase angle for different preselected values of  $\omega$ .

Care must be exercised both in conducting the plant tests and in reducing the data. If pure time delay is present as indicated from time history records, it should be removed by shifting the time axis prior to data processing. Hysteresis is a more difficult nonlinearity to handle and its effect on the results of pulse testing have not been thoroughly explored.

In any event, where permissible, the results of the reduction of pulse data are most conveniently presented as a Bode diagram, with amplitude ratio and phase angle plotted versus frequency. Figure 1 shows the comparative results of pulse testing and sinusoidal testing of a shell and tube heat exchanger.

Generally it is useful and instructive to derive a linear model which will describe the frequency response data. This is done by finding a combination of linear Laplacian forms which will reproduce the data as a Bode plot.

This procedure of fitting a linear model to the data of a system known to be nonlinear has been found, in our experience, to be extremely useful and is usually sufficient to permit subsequent synthesis of the correct control system. A single model is not expected to be valid for all levels or modes of operation of a process whose behavior changes drastically as the operating levels are altered. However, within a small range of normal operating conditions, these simple models have proved to be very useful and adequate for the purposes intended.

There are several advantages to this method—for one, the time required for the test is reduced to a minimum because only one dis-

turbance is required instead of a series of sinusoids. Another is that the plant can remain in the normal operating mode with little or no production time loss.

Conversely, there are some disadvantages—it is usually necessary to have available a large digital computer to generate the Fourier transform pairs, so the test engineer will not spend the major part of his time deriving the transforms by hand. Another is the degree of uncertainty involved in transforming chart readings by hand to digital data sheets, or if one wishes to be more sophisticated, the use of an analog to digital converter device to feed the computer. A third disadvantage is the necessity of fitting the resulting Bode plots to curves describing known linear models.

Still another approach to this problem of deriving simple linear models characterizing a controllable process is to augment the pulse approach by utilizing an analog computer programmed with a preselected set of general models with adjustable coefficients. The scheme of operation here would be to pulse both the independent variable input and the proposed model and then to compare the process and model outputs. A similar device has been marketed by the Wayne Kerr Company and is called "The Transfer Function Computer." The disadvantage of such a system in present state development has been the interdependence of the coefficients. Once one coefficient has been adjusted so that the best agreement of process and model has been attained; upon proceeding to adjust the second coefficient one discovers that the first must be readjusted again to further minimize the difference between process and model, and so on. To overcome this interdependency of the coefficients, 3, 4, 5, considerable work has been done to derive process identification models in a specific way. These models are described technically as orthonormal polynomials. A short discussion of the principles developed in the above references follows.

Consider a linear, stable process whose dynamic characteristics are unknown. This open loop system may be described diagrammatically:

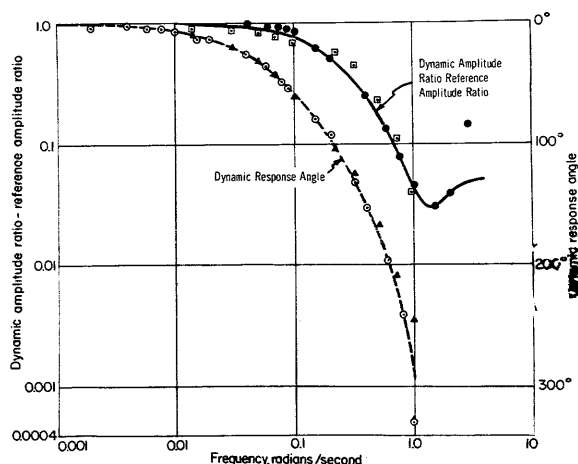
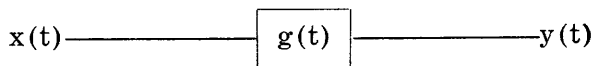
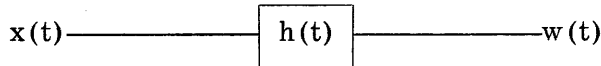


Figure 1. Comparison of Frequency Response Data Obtained by Direct and Pulse Testing Methods. Tube and Shell Heat Exchanger. Input: Water Valve Stem Displacement. Output: Effluent Water Temperature — ● and ○ from Pulse Test; ▲ and □ from Frequency Test. Source: Lees, S., and Hougen, J. O., *Industrial Engineering Chemistry*, 48, 1058 (1956). Frequency Test Data Supplied by S. H. Goodhue, Foxboro Company.

where  $x(t)$  is the input to the process,  $y(t)$  the process output, and  $g(t)$  the process impulse response function.

Similarly, associate with the process a model by the same input as follows:



where  $x(t)$  is an identical input to the model,  $w(t)$  is the model output, and  $h(t)$  the model impulse response function, which will be adjusted in such a way as to make  $w(t)$  as much like  $y(t)$  as possible.

Next, combine  $y(t)$  and  $w(t)$  through a summing device

$$e(t) = y(t) - w(t)$$

where  $e(t)$  is called the error signal. The error signal is squared and integrated, allowing a continuous evaluation of the following equation:

$$\begin{aligned} |E| &= \int_0^{\infty} [e(t)]^2 dt \\ &= \int_0^{\infty} [y(t) - w(t)]^2 dt \end{aligned} \quad \text{Eq. 1}$$

It is desired to adjust the model in a systematic manner such that  $|E|$  will be minimized. If the input is a pulse, one may write

$$X(j\omega) = \int_0^{\infty} e^{-j\omega t} x(t) dt$$

and if  $g(t)$  and  $h(t)$  are stable,

$$G(j\omega) = \int_0^{\infty} e^{-j\omega t} g(t) dt$$

$$H(j\omega) = \int_0^{\infty} e^{-j\omega t} h(t) dt$$

where  $G(j\omega)$  is the transfer function of the process and  $H(j\omega)$  is the transfer function of the process and  $H(j\omega)$  is the transfer function of the model. After considerable manipulation, Equation 1 is transformed as follows:

$$|E| = \frac{1}{2\pi} \int_{-\infty}^{+\infty} |X(j\omega)|^2 |G(j\omega) - H(j\omega)|^2 d\omega \quad \text{Eq. 2}$$

where

$$|X(j\omega)|^2 = X(-j\omega) \cdot X(-j\omega)$$

and

$$|G(j\omega) - H(j\omega)|^2 = |G(j\omega) - H(j\omega)| \cdot |G(-j\omega) - H(-j\omega)|$$

Equation 2 will be used to generate  $H(j\omega)$ , the frequency analog of  $h(t)$ . Now let

$$H(j\omega) = \sum_{i=1}^n a_i K_i(j\omega) \quad \text{Eq. 3}$$

If Equation 3 is substituted in Equation 2, it can be shown that a sufficient condition for  $|E|$  to be a minimum and that the  $a_i$  in Equation 3 to be noninteracting is for the  $K_i(j\omega)$  to be orthonormal. The restrictions produce

$$a_i = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \text{Re} |G(-j\omega) \cdot K_i(j\omega)| \cdot |X(j\omega)|^2 d\omega \quad \text{Eq. 4}$$

Equation 4 illustrates the dependence of the  $a_i$  on input, model, and process. If the input is measured with care, the adjusted  $a_i$  will insure a good representation of the process if the  $K_i(j\omega)$ 's are chosen properly. In the following the  $j\omega$  will be replaced by the Laplacian operator  $s$  in order to obtain more conventional forms. Three orthonormal functions have been constructed from functions of the type:

$$P_i(s) = \sqrt{\alpha} \frac{(1 - \alpha s)^i}{(1 + \alpha s)^{i+1}} \quad \text{Eq. 5}$$

Research in this area may be considered as an offshoot of ideas originating largely with the late Norbert Wiener<sup>3</sup> and A. N. Kolmogoroff<sup>4</sup> who were responsible for three crucial phases of the present work: the discovery of a convenient form for representing the discrepancy between process and model in the time domain, the conversion in the time domain to equivalent integrals in the frequency domain, and the introduction of orthonormal functions as an aid to minimizing the discrepancy of process and model.

To date, we have derived two sets of orthonormal polynomials to construct models of the general form  $H(s) = a_1 K_1(s) + a_2 K_2(s) + a_3 K_3(s)$ . These are: if  $T$  equals unity—

$$K_1(s) = \frac{1 - s}{1 + 2s + s^2}$$

$$K_2(s) = \frac{1 - 3.8266s + 2.8277s^2}{1 + 3s + 3s^2 + s^3}$$

$$K_3(s) = \frac{1 - 2.5841s + 4.407s^2 - 2.823s^3}{1 + 4s + 6s^2 + 4s^3 + s^4}$$

and if in Eq. 5,  $\sqrt{\alpha} = 1$  and let  $i$  be 0, 1, 2:

$$K_1(s) = \frac{1}{T_1s + 1}$$

$$K_2(s) = \frac{1 - T_1s}{(T_1s + 1)(T_2s + 1)}$$

$$K_3(s) = \frac{(1 - T_1s)(1 - T_2s)}{(T_1s + 1)(T_2s + 1)(T_3s + 1)}$$

The first set of the above was made orthogonal to the input weighting function of a rectangular input pulse of duration of  $T/2$ . If the duration of the unknown output is long compared to the extent time of the input pulse, the second set of polynomials may be substituted for the first. The second set has been used to find simple models of flow systems and other processes. The diagram of the functional scheme is given in Figure 2.

In actual practice, the values of  $a_1$  and  $T_1$  are first adjusted for a minimum time integral difference squared between the unknown and the model output, then  $a_2$  and  $T_2$  are adjusted, and still later,  $a_3$  and  $T_3$ . When the best fit is obtained by observing the time histories of both the unknown and model to the pulse input along with the value of the integral, the coefficients,  $a_1, T_1, a_2, T_2, a_3,$  and  $T_3$  are read out and a simple expression is obtained for the input-output pair of interest. The results of the application of these to unknown transfer functions are shown in Figures 3 and 4. Figure 4 shows how transport delay times can be handled—that is,

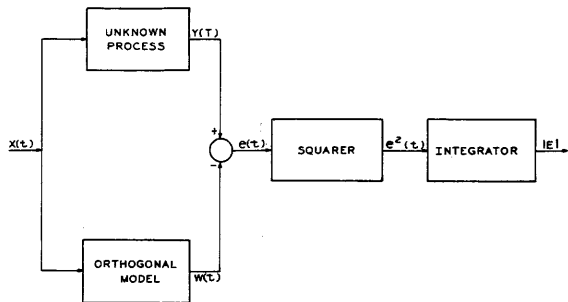


Figure 2. Functional Scheme.

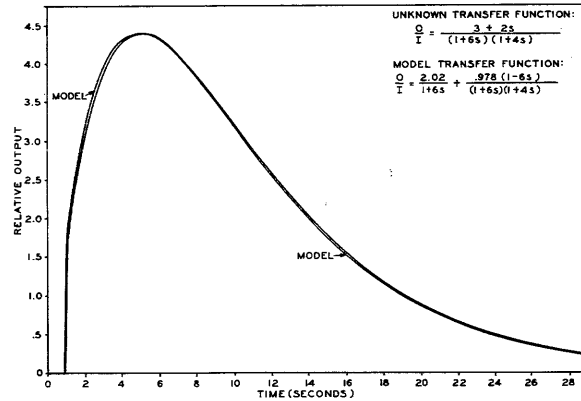


Figure 3. Time Responses of Unknown and Best Model Values. Input Pulse 100 Units .2 Seconds Duration.

the input pulse to the model is delayed so that the output of the unknown and the model will begin at coincident times.

### UTILIZATION OF PROCESS DATA

Once it has been established that the simple linear model is sufficient to describe the process in the region of interest, it is routine to determine the correct values of integral action, proportional action, and derivative action on a standard process controller. The methods of Bode and Black, Nyquist, root locus, etc. apply directly.

Moreover, the open loop adaptive control function can be determined by establishing a simple linear model for different levels of operation. Thus, the necessary controller characteristics for the entire set of model functions can be found as a function of these levels and with the application of these characteristics the

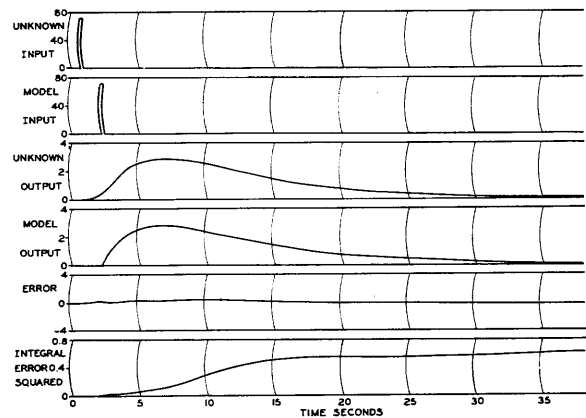


Figure 4. Model Results for Unknown Containing Transport Delay.

process will always be operating at an optimum.

Chemical systems are almost always non-oscillatory—that is, the transfer functions describing the dynamics are a series of time constants in the denominator with real roots. This makes it very easy to formulate a procedure to obtain a slightly more damped close loop response than the one cycle quarter amplitude ratio, one and one-half overshoots, or minimum error squared criteria. Stability apart, this is done by setting the transfer function of the closed loop system with unknown controller constants equal to unity and rewriting the expression in descending powers of  $s$  in numerator and denominator. One then equates the coefficients of like powers and solves for the controller characteristics.

The general controller characteristic equation used will be

$$\frac{e_o}{e_i} = \frac{K_i}{s} + K_p + K_d s$$

This type of control is easily simulated on the analog computer and is the one used in the following example.

**EXAMPLE**

In order to demonstrate the use of the orthogonal polynomials approach and still have something that is realistic, portable, and would utilize the repetitive operation mode of the analog computer, a system of two incandescent lights and photosensitive resistors was selected. The bulbs and photosensitive cells were arranged in two compartments of a box with each cell looking at a bulb.

This system of lights was selected as an example in preference to other applications (flow or pressure control) of this technique to chemical processes, and was from the beginning designed to be a part of a portable ensemble demonstrated to engineering groups. Many of the dynamic characteristics of actual chemical systems are exhibited by the incandescent lamp-photosensitive combination.

In this “process” the first bulb is operated at a constant light level as adjusted by the steady-state output of a computer amplifier.

The first photocell “looks” at this bulb’s light output and is arranged to diminish the second bulb’s input for an increase in the light it senses. The second photocell is arranged in a circuit to give an output proportional to the intensity of the second bulb. The light from the second bulb is isolated from the first photocell so that no system instabilities can exist. Thus, when the first bulb is pulsed, the second photocell produces an output change depending on the direction, gain, and dynamics of the whole combination. The analog diagram for the arrangement is shown in Figure 5.

If one were to try to consider the simulation of the dynamics of the system, the simulation might be as follows:

*Each light bulb:*

$$\text{Power in } (P_i) = \text{Power radiated } (P_r) + \text{Power absorbed } (P_a) + \text{Power conducted away } (P_c)$$

Thus:

$$\frac{E_{in}^2}{R} = K_1 e_t T^4 + K_2 mcp \frac{dT}{dt} + K_3 (T - T_{room})$$

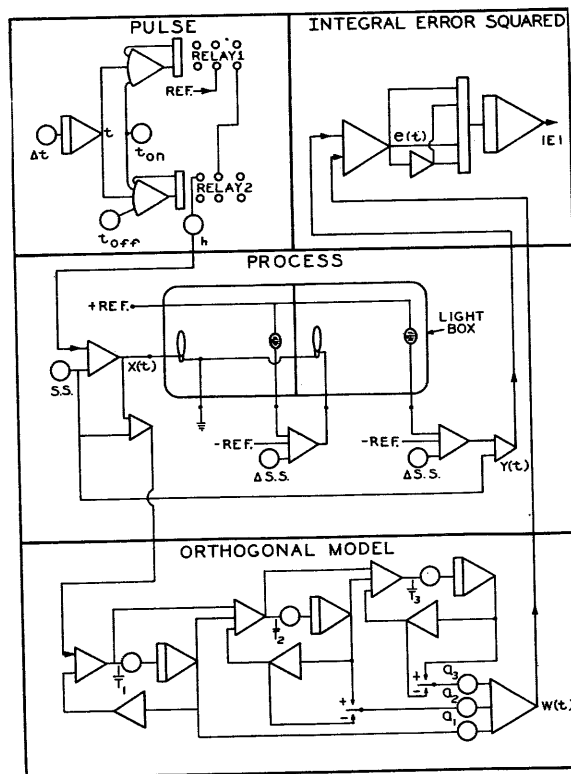


Figure 5. Computer Diagram During Process Identification.

- $e_t$  Emissivity of tungsten
- $R = \rho(1 + \alpha T)$
- $K_1$  Stefan-Boltzmann Constant  $5.67 \times 10^{-8}$  watt/meter<sup>2</sup>/o<sub>K</sub><sup>4</sup>
- $m$  Mass of Filament
- $cp$  Heat Capacity
- $K_3$  Coefficient of Conductivity

or

$$\frac{dT}{dt} = \frac{E_{in}^2}{\rho(1 + \alpha T) K_2 mcp} - \frac{K_1 e_t T^4}{K_2 mcp} - \frac{K_3 (T - T_{room})}{K_2 mcp}$$

*Each photocell:*

Upon consultation of the manual on CdS photoconductive cells, 7, one will see that there is a response time associated with the output current vs. the change in light level. These curves show that this apparent time constant is not constant with level or past history but a definite function of the history.

Thus it might be expected that the "system" will be at least fourth order.

To determine the "transfer function" experimentally, we will use the analog computer to pulse the first light, observe the "process system" output. Next, pulse the input to the orthogonal model, observe its output, and compare the differences between process and model output on an integral error squared basis. The orthogonal set programmed on the computer is:

$$\frac{O}{I} = \frac{a_1}{T_1 s + 1} \pm \frac{a_2 (1 - T_1 s)}{(T_1 s + 1) (T_2 s + 1)} \pm \frac{a_3 (1 - T_1 s) (1 - T_2 s)}{(T_1 s + 1) (T_2 s + 1) (T_3 s + 1)}$$

When the steady-state level of the lights is 80 volts and the input pulse height to the unknown is 10 volts, the values of the coefficients  $a_1, a_2, a_3$  and  $T_1, T_2, T_3$  were found to be:

For an increasing pulse	For a decreasing pulse
$a_1 = 3.13$	$a_1 = 3.29$
$T_1 = .0543$	$T_1 = .0538$
$a_2 = .657$	$a_2 = .892$
$T_2 = .0063$	$T_2 = .0081$
$a_3 = .579$	$a_3 = .632$
$T_3 = .0067$	$T_3 = .0075$

Gain =  $a_1 + a_2 + a_3 = 4.37$     Gain =  $a_1 + a_2 + a_3 = 4.81$

A static measurement at 80 volts with an input change from 79 to 81 volts gives a gain of 4.7. Other values for the orthogonal model constants at different operating levels are given in Table I.

The values in Table I can be used in the general expression of the model, and a "transfer function" written for each case. These are shown in Table II. Inspection of the numerator of each expression in the orthonormal column might lead one to think that the  $s$  terms are part of the  $e^{-Ts}$  series. Thus, ignoring the validity of the sign of the  $s^2$  term in some cases, the function might be written as shown in the exponential columns of Table II. Typical time histories of the application of this technique to the "process" are shown in Figure 6.

In process systems, the controller most always follows the error device being placed before the process in the forward loop. Thus, the form of the closed loop function is always

$$\frac{O}{I} (s) = \frac{G_c G_p}{1 + G_c G_p}$$

The above can be rearranged into the form:

$$\frac{\alpha s^n + B s^{n-1} + \dots + 1}{A s^n + B s^{n-1} + \dots + 1}$$

If we then set this expression equal to unity, and require that the models of the same be unity for all  $s$ , then, we must equate coefficients of like powers.

TABLE I. ORTHOGONAL MODEL CONSTANTS AT DIFFERENT OPERATING LEVELS

Level* (Lamp Volts)	Model Constants					
	$a_1$	$T_1$ sec.	$a_2$	$T_2$ sec.	$a_3$	$T_3$ sec.
40	52.2	.102	29.0	.040	2.78	.0073
50	31.2	.077	21.8	.033	1.77	.0086
60	10.78	.055	7.46	.024	.596	.0024
70	3.86	.041	3.00	.019	.368	.0069
80	3.13	.054	.657	.0063	.579	.0067
90	1.29	.043	.397	.0059	.208	.0075

\* Steady-state value of supply voltage with a rectangular pulse of + 10 volts for 20 milliseconds duration input disturbance in each case.

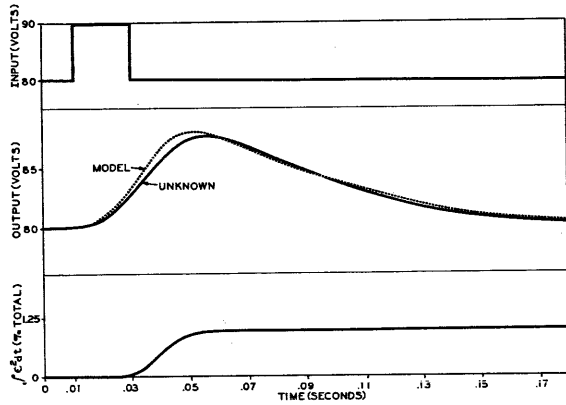


Figure 6. Typical Time Histories of Example Application.

Thus, if the process is of the same order as the controller, it is possible to completely compensate the system. If, however, as is the general case, the process order is higher, astute judgment must be used in selecting the predominate quantities. The values of  $K_i$ ,  $K_p$  and  $K_d$  at different levels can be calculated this way

using the data of Table II. These are shown in Table III.

The analog computer was used as a check of these values. The diagram is shown in Figure 7.

Equating like coefficients will often give a transient response slightly more damped than the integral error squared or the integral absolute value criteria. This is not a deterrent to the application of the technique to chemical processes, because in most instances the process is quite noisy and very tight control is probably impossible.

Close examination of Table III suggests that the controller parameters  $K_i$ ,  $K_p$  and  $K_d$  be made some function of the level of operation. Logarithmic plots of these functions demonstrate that there is approximately the same relative changes in each parameter for changes in the level (i.e., the controller constants calculated in each case are essentially only a function of the open loop gain). In the example cited, the approximate function of operating level compensation is to interpose in the error

TABLE II. THE APPROXIMATE LAPLACE FORMS OF THE MODEL FUNCTIONS

Level* (Lamp Volts)	Functional	
	Orthonormal	Exponential
40	$\frac{84 (1 - .012s + 5.9 \times 10^{-5}s^2)}{(.102s + 1) (.04s + 1) (7.3 \times 10^{-3}s + 1)}$	$\frac{84 e^{-.01s}}{(.102s + 1) (.04s + 1) (7.3 \times 10^{-3}s + 1)}$
50	$\frac{55 (1 - 7 \times 10^{-3}s - 2 \times 10^{-5}s^2)}{(.077s + 1) (.033s + 1) (8.6 \times 10^{-3}s + 1)}$	$\frac{55 e^{-.007s}}{(.077s + 1) (.033s + 1) (8.6 \times 10^{-3}s + 1)}$
60	$\frac{19 (1 - 8 \times 10^{-3}s + 2.2 \times 10^{-5}s^2)}{(.055s + 1) (.024s + 1) (2.4 \times 10^{-3}s + 1)}$	$\frac{19 e^{-.008s}}{(.055s + 1) (.024s + 1) (2.4 \times 10^{-3}s + 1)}$
70	$\frac{7.2 (1 - 3.4 \times 10^{-3}s - 7.8 \times 10^{-6}s^2)}{(.041s + 1) (.019s + 1) (6.9 \times 10^{-3}s + 1)}$	$\frac{7.2 e^{-.003s}}{(.041s + 1) (.019s + 1) (6.9s + 1)}$
80	$\frac{4.4 (1 - 5.8 \times 10^{-3}s + 2.1 \times 10^{-5}s^2)}{(.054s + 1) (.0067s + 1) (6.3 \times 10^{-3}s + 1)}$	$\frac{4.4 e^{-.006s}}{(.054s + 1) (.0067s + 1) (6.3 \times 10^{-3}s + 1)}$
90	$\frac{1.9 (1 - 3.7 \times 10^{-3}s - 9.6 \times 10^{-6}s^2)}{(.043s + 1) (.0075s + 1) (5.9 \times 10^{-3}s + 1)}$	$\frac{1.9 e^{-.004s}}{(.043s + 1) (.0075s + 1) (5.9 \times 10^{-3}s + 1)}$

\* As in Table I.



TABLE III. VALUES OF CONTROLLER CONSTANTS FOR VARIOUS OPERATING LEVELS

Level* (Lamp Volts)	Controller Constants		
	$K_i$	$K_p$	$K_d$
40	.11	.017	.0005
50	.24	.026	.0006
60	.96	.076	.0010
70	3.40	.203	.0030
80	4.20	.260	.0020
90	12.00	.620	.0040

\* For an increasing 10 volt step. It is possible to experience instability when the same values are used with a decreasing step.

signal a gain change equal to  $.0014 e^{L/10}$ , where  $L$  is the lamp level in volts. Coupled with this, the fixed controller values are to be adjusted to  $K_i = 12$ ,  $K_p = .62$  and  $K_d = .004$ . This will give good control at all levels of operation without readjustment.

## REFERENCES

1. HOUGEN, J. O., and R. A. WALSH, "Pulse Testing Method," *Chemical Engineering Progress*, Vol. 57, No. 3, March 1961.
2. HOUGEN, J. O., C. G. HAGBERG, L. H. FRICKE, and O. R. MARTIN, "Process Identification and the Design of Process Systems with Predictable Performance." Proceedings of the Twelfth Annual Instrumentation Conference, Louisiana Polytechnic Institute, Ruston, Louisiana. (To be published in *Chemical Engineering Progress*, August 1964).
3. WIENER, N., *The Extrapolation, Interpolation and Smoothing of Stationary Time Series with Engineering Applications*, Wiley, 1949.
4. EYKHOFF, P., "Process—Parameter Estimation," Technological University, Elec-

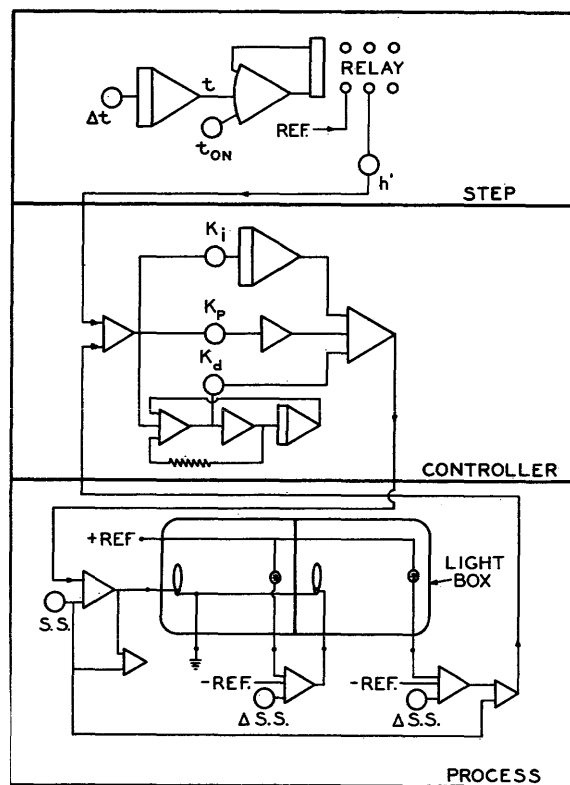


Figure 7. Computer Diagram During Process Control.

tronics Laboratory, Delft, Netherlands, May 1962.

5. KOLMOGOROFF, A. N., "Interpolation und Extrapolation von Stationären Zufälligen Folgen," *Bulletin de l'Academie des Sciences de USSR, Ser. Math.* 5, 1941.
6. KITAMORI, T., "Applications of Orthogonal Functions to the Determination of Process Dynamic Characteristics and to the Construction of Self-Optimizing Control Systems," *Automatic and Remote Control*, Proceedings of the First International Congress of IFAC, Moscow, 1960, Volume II, pp. 613–618, Butterworths, 1961.
7. RCA Photocells Circuits Data Solid-State Photosensitive Devices Booklet ICE-261, Radio Corporation of America, Electron Tube Division, Harrison, N. J.



# PROGRESS OF HYBRID COMPUTATION AT UNITED AIRCRAFT RESEARCH LABORATORIES

*Gerard A. Paquette  
United Aircraft Research Laboratories  
East Hartford, Connecticut*

## INTRODUCTION

Many of the present day simulation problems utilizing analog equipment exclusively require large, expensive computing facilities. The nonlinear calculations involved in modern simulations require a large portion of the all analog system. Increase in operation speed and reliability coupled with reduction in purchase cost have made the digital computer desirable for these nonlinear computations.

Prior to the end of 1962, the analog facility at United Aircraft Research Laboratories consisted of two large analog consoles, one small analog console, and associated analog equipment with emphasis on nonlinear devices. At this time, several problems required all of this equipment combined.

In hopes of improving the operation and capacity for highly nonlinear simulations, a hybrid facility<sup>1</sup> was added toward the end of 1962. This consisted of a general purpose digital computer, a D.C. analog computer, and interconnecting linkage. The hybrid system is presently being used to simulate V/STOL aircraft, aircraft,<sup>2</sup> engine, and space systems in real time. An example, Single Rotor Helicopter Simulation, is discussed in this paper.

Initial plans for the hybrid facility were to use the digital computer for most nonlinear functions and the analog computer for dynamics. As time progressed, advanced tech-

niques were developed to permit further digital application. These techniques include compensating calculations for sampling delay and practical integration schemes.<sup>3-5</sup> Thus in some systems presently simulated at United Aircraft, most or all computation is becoming digital.

In addition, the digital computer is used to support problem preparation and modification. Utility programs permit procedure and data changes from paper tape, typewriter, or digital scope and light pen input.

## THE UNITED AIRCRAFT HYBRID FACILITY

### *Analog*

A Beckman Ease 2133 Analog Computer is assigned to the hybrid installation. Its complement of equipment emphasizes linear analog operations in the expectation that the digital computer will process most nonlinear calculations. The computer contains 100 operational amplifiers, 150 potentiometers, 20 time division multipliers, 10 quarter square multipliers, and 20 11-segment, diode function generators.

Auxiliary equipment includes a DO/IT (Digital Output/Input Translator) which permits typewriter control of pot setting and component reading. A digital voltmeter is used for component reading through the DO/IT or by itself. A digital clock, part of the analog system hardware, is available for mode timing.

Mode switching is provided by reed relays to permit rapid, repetitive operation.

### *Digital*

A Digital Equipment Corporation PDP-1 Computer was selected as the best compromise of capacity, speed, and cost among 1962 computers useful for hybrid application. This computer resembles the TX-O computer at M.I.T. which has been used for real time simulation.<sup>2</sup>

The PDP-1 is a single address, single instruction, stored program machine with a word length of 18 binary bits. Basic memory contains 4096 words with a memory cycle time of 5 microseconds. Instructions available permit a wide range of logical operations and fixed-point arithmetic. Instruction execution requires 5 to 10 microseconds for most operations. Multiply and divide average 20 and 30 microseconds, respectively. These speeds would permit execution of approximately 2000 instructions in a program providing 50 solutions per second.

Input/output devices include analog linkage, typewriter, paper tape reader, paper tape punch, and digital scope. The photoelectric tape reader accepts eight level punched tape at a rate of 400 characters per second. The tape punch provides output at 63 characters per second.

An incremental digital scope can be used to display graphical and/or alphanumeric information. Various scope modes permit display of points, incremental data, vectors, or characters. Writing speeds vary from 35 microseconds per point in point mode to 1.5 microseconds per point in vector mode.

A 2048 word memory is used with the scope to maintain a fixed display while the computer is operating independently. The scope memory includes hardware to permit programmed character generation. Data transfer from the computer to this memory can be made as desired to modify the display.

Light pen input to the computer is available from the scope. On sensing light, the pen interrupts the computer and display sequence. The current scope coordinates can be read to locate the pen after the interrupt.

A real time, digital clock is included for timing of digital computation cycles. Two modes

of clock operation are used over a time range of 10 microseconds to 13 seconds. In one case, the clock can be preset to interrupt the program at a desired time interval within the above range. In the second case, the clock can be used as a data source with clock clearing and reading under program control.

### *Computer Linkage*

The linkage system can be divided into two distinct parts: the computing linkage and the control linkage. The computing linkage includes both data and logic converters from each computer to the other. All converters are under digital program control.

Computing data linkage includes 20 digital-to-analog and 20 analog-to-digital conversion channels. These utilize a word length of 14 binary bits corresponding to an analog range within plus and minus 128 volts. The 14 bit word provides a resolution of 1 part in 16,384 which is consistent with analog accuracy.

Digital-to-analog converters were designed and built at United Aircraft. The data word from the digital computer is transferred to a 14 binary bit, flip-flop buffer storage; analog gates; resistor network; and an operational amplifier. Programmed data transfer to the converter buffer requires 5 to 15 microseconds depending on the initial location of the data word. Overall conversion speed is primarily dependent on the response of the output amplifier.

Analog-to-digital conversion equipment consists of a Packard Bell, 20 channel multiplexer, a sample and hold amplifier, and a Multiverter. Programming sequence includes three instructions: select-multiplexer-channel, convert, and read-converter-buffer. The select-multiplexer-channel instruction addresses the analog input to be connected to the sample and hold. A delay of at least 15 microseconds must be programmed before the convert instruction to permit multiplexer switching. On execution of the convert instruction, the sample and hold enters hold mode in which it will remain until conversion is complete. The Multiverter converts the analog voltage one bit at a time and stores the resulting binary word in its buffer. Conversion requires approximately 75 microseconds. The conversion can be programmed for automatic

computer delay or the time can be made available for computation. The read-converter-buffer instruction transfers the data word into the computer. Overall analog-to-digital conversion time is 90 microseconds.

Logic linkage includes 4 digital-to-analog and 8 analog-to-digital conversions. The digital-to-analog logic converters permit program control of single-pole, double-throw relays at the analog console. A ground or open circuit at the input of each analog-to-digital logic converter sets a digital status bit to one or zero, respectively.

The control linkage allows the PDP-1 to set analog computer modes and operate or monitor all DO/IT functions. A single instruction corresponds to each analog computer mode. Existing modes at the analog computer set designated bits of a status word in the PDP-1 to close the information loop.

For DO/IT operation from the PDP-1, character codes similar to those of the DO/IT's own typewriter can be transmitted to and from the DO/IT hardware. The digital program can then become a replacement for this typewriter.

#### HYBRID APPLICATION AND PREPARATION

Real time simulation has long been associated with the analog computer. Preparation and modification of the circuit diagram, patch board, and potentiometer values is rapid. Improvements in analog components and automation of set up procedures have advanced analog computing capability. However, as the operating ranges of modern physical systems have increased, the simulation nonlinearities and accuracy requirements have grown. Dynamically, we still handle problems in 6-degrees of freedom but the needs for variable products, trigonometric functions, arbitrary functions, and other nonlinearities have increased to the point where the parallel analog system requires too many expensive, and sometimes insufficiently accurate, devices.

#### *Application*

The hybrid facility consists of digital and analog devices and uniting linkage. The problem equations are divided between the two com-

puters to take advantage of the capabilities of each device<sup>4,5</sup> as indicated in Table I.

The analog equipment is generally used for dynamics, implicit algebra, and simple diode controlled nonlinearities though the digital can often be extended into these areas as well. The evaluation of nonlinear operations, especially arbitrary function generation, is the primary application of the digital computer.

Analog signals are repetitively sampled by the analog-to-digital converters, operated on by the sequential digital program, and transferred to the analog through the digital-to-analog converters. The controlling factors in determining the operations to be performed digitally include:

1. Digital memory available for program and data
2. Conversion channels available to link the computers
3. Time permitted for each sample-calculate-output cycle

The quantity of digital memory and conversion channels to be provided is determined from a rough attempt at designing existing problems into the hybrid system.

Digital program cycle time is usually the primary limiting factor. The sample-calculate-output cycle presents a fictitious time delay to the simulation and must be sufficiently short relative to the periods of the real system. Some advance knowledge of the frequency spectrum

TABLE I. TYPICAL USES FOR EACH COMPUTER OF THE HYBRID FACILITY

Analog Computer	Digital Computer
Time Integration	Arbitrary Function Generation
Constant by Variable Products	Variable by Variable Products
Implicit Calculations	Trigonometric Calculations
Limits, Dead Zones, etc.	Logical Control
Flight Simulator Coupling	High-speed Display

of the real system is generally available but the delay influence is best determined experimentally. Application of the digital to the lower frequency portions of the problem allows lower point rate and more digital operations per cycle. Use of the digital in the higher frequency areas can be permitted if the fewer calculations available are more significant in their corresponding analog representation. Some problems require application of the digital computer to portions with different frequency bandwidths. For optimum digital utility, time sharing of the separate sample-calculate-output cycles can be provided by the program.

As an example of selecting initial cycle time, a reasonable choice is to use one-hundredth of the shortest period of interest. This would provide a phase error of less than four degrees at the selected frequency. The time delay error can be further reduced by corrective measures discussed below.

Effects of the sampling rate obtained with the digital program are tested experimentally. The nonlinear nature of most simulations does not permit easy evaluation and comparison of damping factors, frequencies, etc. A number of scattered dynamic test cases are run at real time. These are repeated with the analog time constants increased or with dummy time delays added to the digital program to represent relative slowing or speed-up. This form of comparison is also useful to indicate future capabilities of the digital. Final problem verification also includes, whenever possible, the simulation of a known vehicle and comparison with flight test before extrapolation to an advanced vehicle.

### Preparation

The preparation of the digital program<sup>6</sup> is quite similar to preparing the analog circuits, as seen in Figure 1, except that a different language is used. Similar data scaling operations exist in preparing the problem for either computer. Digital program sequence is linear, sacrificing space in order to avoid time consuming subroutine linkage. Symbolic, machine language is used in describing the operations to be performed and the locations of the information in memory. Basic routines such as sine,

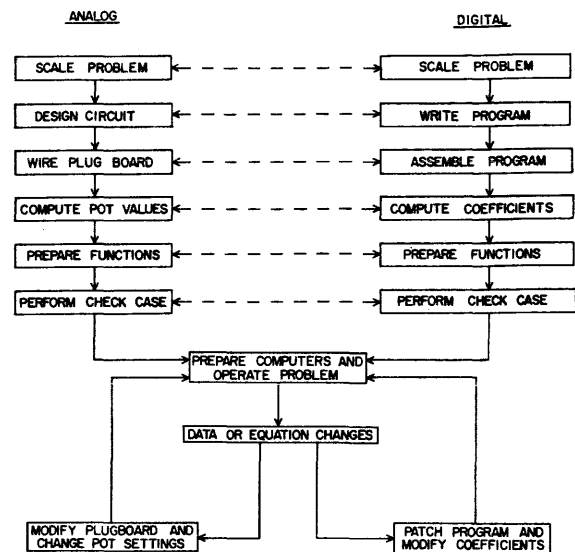


Figure 1. Problem Preparation for the Computers of a Hybrid Facility.

arctangent, function interpolation, square root, etc. are available as building blocks for the programmer. The symbolic information is converted to computer language by an assembly program.<sup>7</sup> Our experience at United Aircraft has shown the digital programming to be easily mastered by the same simulation engineers who design the analog circuits. The program can generally be prepared in the same amount of time as designing and wiring a corresponding analog circuit.

Though somewhat slower than analog patch board changing, digital calculations can usually be deleted, inserted, or changed during on-line problem operation by program patching techniques. To retain good system organization, a reassembly of the modified program is obtained after the simulation session.

Digital data changes, including function curves, are more rapid than can be obtained at the analog. This capability is provided by auxiliary digital programs discussed later.

Checkout of the digital and analog calculations is initially independent. Digital algebra and trigonometry are verified relative to the system equations with all logical branches independently checked. Analog check is performed by applying static voltage to various portions of the problem and testing component output values or time constants. The digital checking

requires more time than analog because of the larger number of calculations usually performed at the digital. Typical checkout of the program has taken from two to three 8-hour shifts. When each computer portion is independently checked, the system is closed and conversion channels are checked for proper wiring at the analog and proper timing at the digital.

**SOME DIGITAL TECHNIQUES FOR SIMULATION**

Relative to the analog computer, the digital computer is a new element in real time simulation and its application bears some examination. A few demonstrative techniques are discussed below.

*Delay Correction*

The digital computer is generally applied in the area of calculation from the motions of a system to its derivatives. The digital output will remain fixed over each sampling interval and, though the analog integrations vary continuously with time, the analog will be read at discrete levels each program cycle. Each output from the digital is dependent on data obtained at the start of the program cycle and represents a delay of at least one cycle, T.

Consider the time relations of a closed loop calculation involving one integration with digital feedback as shown in Figure 2. Initially, the digital computer should have cycled at least once before problem start to provide derivative output for zero time. The functional time sequence after the problem begins operation is shown in Figure 3. Assuming the computation begins with the digital sequence at the start of a cycle, the variable read at zero time will be used to compute the derivative output at time T. Likewise, the variable at time T will lead to derivative output at 2T. Thus the analog re-

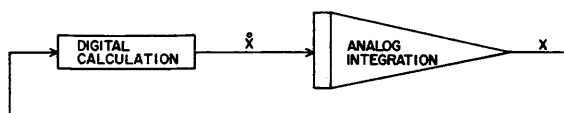


Figure 2. Typical Computer Application.

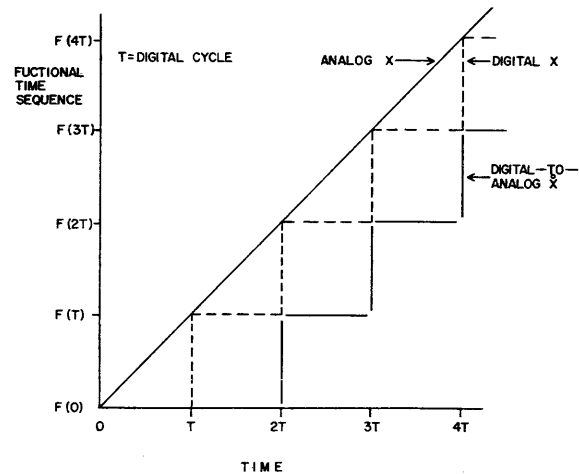


Figure 3. Function Timing Without Delay Correction.

ceives data one cycle time behind true time and must integrate this constant data until the completion of the next digital cycle.

$$x(nT) = x(nT - T) + Tx(nT - 2T) \quad (1)$$

$$n = 1, 2, \dots$$

Higher order analog integration is affected in much the same manner. Two integrators in sequence would operate as in equation (2).

$$x(nT) = x(nT - T) + \frac{T^2}{2}x(nT - 2T) \quad (2)$$

The effect of digital delay can be reduced, where necessary, by extrapolating problem variables forward one sample interval. The resulting influence, illustrated in Figure 4, changes Equation (1) to the form in Equation (3).

$$x(nT) = x(nT - T) + Tx(nT - T) \quad (3)$$

The order of extrapolation used provides a trade-off between accuracy of function fit and time for calculation and data handling. The simplest is linear extrapolation as in Equation (4).

$$x(nT + T) = 2x(nT) - x(nT - T) \quad (4)$$

The parabolic form is shown in Equation (5).

$$x(nT + T) = 3x(nT) - 3x(nT - T) + x(nT - 2T) \quad (5)$$

At zero time, the system is generally at rest allowing the variables at negative times to be

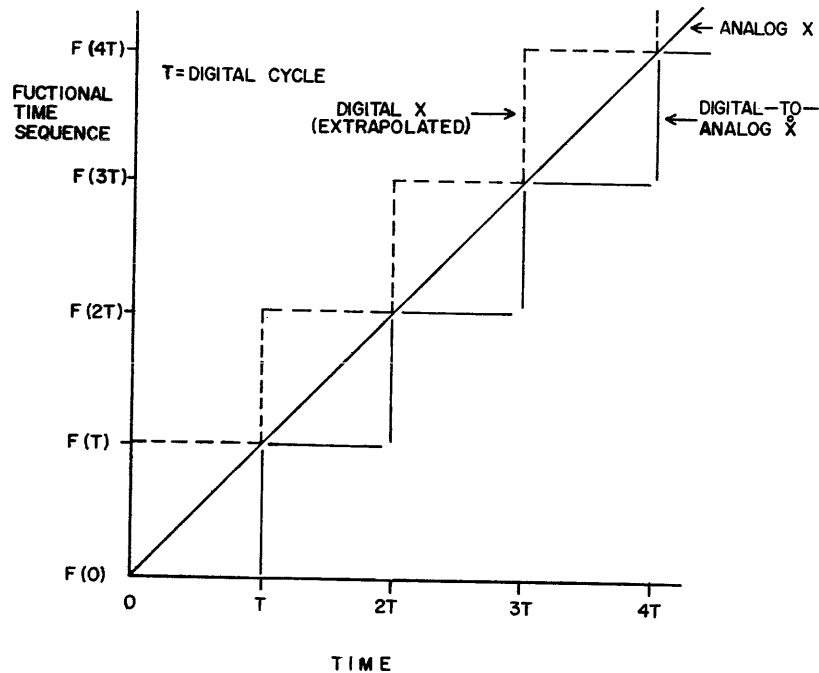


Figure 4. Function Timing with One Interval Delay Correction.

equal to the variable at zero time, i.e.,  $x(0) = x(-T) = \text{etc.}$  If initial derivatives are not zero, an iteration must be made to calculate the initial values at negative time and adjust the derivatives.

#### Digital Integration

An examination of Equation (3) above shows this to be identical to the Euler Forward Step Integration method used for time integration in digital computation. It is therefore possible to digitally simulate the operation of integration by this fast and easily programmed technique. Use of this method for more than a single integration in sequence must proceed from the last integration first. That is,

$$x(nT) = x(nT - T) + Tx(nT - T) \quad (6)$$

then

$$x(nT) = x(nT - T) + Tx(nT - T) \quad (7)$$

For two integrations, it is also possible to use Equation (8) which duplicates the equivalent analog integrations with digital feedback and delay correction.

$$x(nT) = x(nT - T) + \frac{T^2}{2}x(nT - T) \quad (8)$$

For linearly feedback analog transfer functions such as lags, damped second order networks, etc., a difference equation approach, discussed by Hurt,<sup>3</sup> can be used. By this technique, the Laplace transfer function is obtained and converted to an equivalent pulse transform from which a difference equation can be formulated.

#### Arbitrary Function Generation

Various digital techniques are available for evaluating function data. Among these are polynomial representations and table interpolations. The table approach is useful for the arbitrary shapes of aerodynamic curves. However, searching for the table arguments is undesirable because of the execution time required.

A direct function look-up technique,<sup>8, 9</sup> described for bivariate functions in Appendix I, is used at United Aircraft. This method uses a table of function values taken at equispaced input arguments. The input intervals are scaled to permit use of a portion of the binary argument word as a table reference address and the remainder of this word as the linear interpolation ratio.



The use of a single argument interval often requires an excessively large data table to adequately describe the function. Some reduction in table size can be accomplished by subdividing the function into regions with different argument modifying function to the problem argument, i.e.,

$$f(x) = g(h(x)) \quad (9)$$

#### *Function Algorithms*

Some calculations such as trigonometric or logarithmic functions can be reduced to polynomial evaluations. A fast converging and reasonably accurate polynomial algorithm is desired. In general, Chebyshev polynomial fits,<sup>10</sup> for these functions satisfy this need; e.g., sine function for angle  $\theta$  over two quadrants

$$\sin \theta = 1.5706268x - 0.6432292x^3 + 0.0727102x^5 \quad (10)$$

$$\text{where } x = \frac{2\theta}{\pi}$$

The results are accurate to within 0.012% over the range of  $\theta$ , consistent with current day analog resolvers. Execution time in PDP-1 computer averages 140 microseconds.

#### *Iterated Functions*

Some computations require iteration in their evaluation. The square root is a typical example. The square root can be obtained by using the Newton-Raphson method for iteratively solving roots of equations.<sup>11</sup>

$$x(n+1) = \frac{\frac{(x^2)}{x(n)} + x(n)}{2} \quad (11)$$

$$n = 1, 2, \dots$$

A fixed number of iterations can be used without convergence testing if the range of input variable change and accuracy of rooting are known.

In simulation activities, the problem variables should not change by a large ratio from one program cycle to the next unless the variables are insignificantly small within the range scaled. If the successive input variables are within 0.5 to 2 times the previous values and the preceding root is used as the initial output

estimate, four iterations of Equation (11) are known to provide less than 0.01% convergence error.

#### OFF-LINE DIGITAL SUPPORT

Off-line, digital utility programs support the simulation from problem preparation through output presentation. This is especially important when problem turn around time is as frequent as eight hours, a common procedure at United Aircraft. Typical utility programs are as follows:

1. Assembly<sup>7</sup> and debugging<sup>12</sup> systems for program preparation and modification.
2. Coefficient generation programs<sup>6</sup> for evaluating problem constants and potentiometer settings from engineering data.
3. DO/IT control programs<sup>13</sup> (UAC-15, S. Sharpe and J. Miller) for potentiometer setting and component reading.
4. Digital scope, arbitrary function display with light pen, curve modifier.<sup>14</sup>
5. Digital scope display<sup>9</sup> for graphically presented and alphanumerically identified time sequence, cross plot, or steady state data.

Some of these systems are discussed in more detail below.

#### *Coefficient Generator*

Equation coefficients used in the simulation mathematics are functions of parameters of the physical device and its environment. These parameters are part of the problem data language used by the engineer and are often changed during a study.

A digital program is used to compute the problem coefficients from the engineering parameters. All scale factors to be applied in converting coefficients to pot values for the analog computer or constants for the digital on-line program are included. Gain changes needed to adjust the range of computed pot values are applied by the program. Range changes for digital constants are generally not required since the digital on-line program can be scaled for wider range than analog due to the one in over 200,000 parts resolution available.

### *Pot Setting*

With the control linkage between the digital and analog computers, it is possible for a program to set analog potentiometers directly from the results of a coefficient generation program. The settings can be made and checked faster and more reliably than by a human operator. If a failure occurs, the user receives a typed output indicating the difficulty. If gain changes are required, the digital program types the information. For pots requiring frequent gain changes, the digital-to-analog logic linkage can be used to switch gains under program control.

### *Analog Check*

The digital control of the analog computer permits reading of all analog components. With this feature, a program (UAC-8, G. Paquette and J. Miller) has been developed for checking the analog circuits. This program accepts equations representing a static condition on the analog hardware. These are solved interpretively and results are compared with analog components within specifiable tolerances. Errors are typed out. Changes in analog circuitry or pot values can easily be updated in the check equations and data by typewriter control or direct input from the coefficient generator program. This system is discussed further in Appendix II.

### *Function Display and Modification*

Curves used with the hybrid simulation are generally included in the digital calculation. The few, if any, remaining on the analog are still subject to the slow set up and change procedures typical of diode function generators even with servo set capability.

Curves contained in the digital memory can be loaded from paper tape and changed by tape or typewriter input. In addition, the curves can be displayed on a digital scope individually or as maps for easy modification using the light pen. This technique permits the user to rapidly change function data without requiring a calculation of each point.

## OPERATIONAL HYBRID SIMULATION

The Single Rotor Helicopter, Real Time Simulation sponsored by the Sikorsky Division,

demonstrates an application of hybrid computation at United Aircraft. The simulation consists of three basic portions.

1. Fuselage aerodynamics, kinematics, and dynamics.
2. Main rotor aerodynamics and dynamics.
3. Fixed base, flight simulator.

The total force, 6-degree of freedom, aircraft mathematics include computation of fuselage and tail rotor forces and moments as functions of air speed and direction. Along with main rotor components, these forces and moments are applied in computing linear and angular motions. Most nonlinear aerodynamic and kinematic calculations are obtained at the digital computer. Flight path and attitude integrations are performed at the analog computer.

The rigid, hinged blade main rotor dynamic degrees include blade flapping and azimuth speed. Force calculations, including effects of mach number and stall, are obtained over the blade span and rotor azimuth. The entire rotor computation, including dynamics, is performed at the digital computer.

The flight simulator, used to study flying qualities and human factors, contains pilot controls; instruments; and a Norden Conalog, artificial world, television display. The amount of data required is fairly large but the calculations are reasonably simple and contain few nonlinearities. In order to minimize the number of digital-to-analog converters required, these computations are performed at the analog computer.

Auxiliary computation tools provide for rapid implementation of desired studies:

1. A digital coefficient evaluation program to convert engineering data into the digital program constants and analog pot settings.
2. Automatic controls at the analog computer to obtain aircraft trim at desired flight conditions.
3. Digital scope, display programs for aerodynamic functions and dynamic output data.

The digital computer provides the mathematical operations shown in Table II.

TABLE II. DIGITAL OPERATIONS FOR HELICOPTER SIMULATION

---

9 Bivariate Functions
2 Univariate Functions
15 Trigonometric Functions
2 Two-dimensional Coordinate Transformations
2 Square Roots
4 Time Integrations
Numerous Term Products and Sums

---

The digital program cycle time is 0.03 sec. providing five cycles of rotor calculation within each fuselage sample-calculate-output cycle. The rotor calculation is executed more frequently because of the higher frequency range of the rotor system relative to the aircraft. Memory requirements for this program approach 4000 words for the simulation problem and its data. The simulation program, coefficient generator, and arbitrary function display occupy the same memory space and must be used independently.

This problem, in a less sophisticated version, was once simulated exclusively with analog equipment. From this computer representation and estimates of the requirements for the expansion, the digital program alone replaces the following analog equipment:

- 172 Amplifiers
- 295 Coefficient Potentiometers
- 118 Variable Products
- 64 Arbitrary Function Curves

The function curves each contain 32 points, representing 31 linear segments. Exact replacement with diode function generators would require 192 11-segment units.

The analog equipment presently utilized in the hybrid version is as follows:

- 59 Amplifiers
- 67 Coefficient Potentiometers
- 16 Multipliers
- 4 11-Segment Diode Function Generators

An important distinction between the two systems, analog of the past and hybrid of the present, is the speed of set up and modification.

The original analog system required an average of four hours to set up and check out assuming approximately 40 function generators were set during the previous shift. Set up and check out of the hybrid system is less than 30 minutes. Part of the time saving is due to the reliability of the digital computer and the reduction in analog components with a corresponding reduction in the number of equipment failures encountered at check out.

Static trim data for the helicopter was obtained from the original analog simulation; the hybrid simulation; and a highly sophisticated, non-real time, performance program for IBM 7090. Relative to the 7090 results, the hybrid system produced values with one-fifth to one-tenth the differences exhibited by the original analog system.

Time histories of pilot control and resultant helicopter motion were obtained from Sikorsky flight tests. The same pilot control motions were applied to the hybrid simulation. Good correlation with the flight test was obtained.

#### FUTURE INDICATIONS

Trends at United Aircraft indicate the future will find real time calculations processed more extensively by the digital computer. Contributing to this trend are the following points:

1. Considering the helicopter problem above, essentially two consoles of analog equipment were replaced by a digital computer which costs significantly less than one of these consoles.
2. The digital equipment is highly reliable and it can be preventatively maintained during idle time.
3. Once checked out, a paper tape of program and data can be reloaded with high confidence in its subsequent operation.
4. Set up by prepared program tapes is much faster than analog set up.
5. Man-machine communication is improving with utility program support as discussed above.

Some real time problems at United Aircraft have already become essentially digital. The growth of the helicopter problem has been typical of this trend. The original hybridization

utilized the digital computer only in calculating the highly nonlinear, main rotor aerodynamics.

Competitively priced digital computers are now available with larger word size at effective speeds up to four times that of the PDP-1. Indeed, computers with floating point arithmetic, a step toward solving the simulation scaling problem, are becoming available at reasonable cost and provide floating point operation faster than the fixed point calculation of the PDP-1.

#### ACKNOWLEDGMENT

The author wishes to acknowledge the efforts contributed by members of the Analog Section, United Aircraft Research Laboratories, in developing the equipment and software techniques summarized in this paper.

#### APPENDIX I. BIVARIATE FUNCTION GENERATION

The desired function

$$z = f(x, y) \quad (12)$$

is obtained by cross-linear interpolation within a table of function values. The function table is stored in memory without argument values in the following sequence:

$$z_{i, j} = \sum_{j=0}^{K-1} \left( \sum_{i=0}^{L-1} f(x_i, y_j) \right) \quad (13)$$

where  $i, j = 0, 1, 2, \dots$

The  $y_j$  arguments must be selected at equal intervals,  $\Delta y$ , and the  $x_i$  arguments must be selected at equal intervals,  $\Delta x$ . Note that  $i$  and  $j$  can be obtained as:

$$i = \frac{(x_i - x_0)}{\Delta x} \quad (14)$$

$$j = \frac{(y_j - y_0)}{\Delta y} \quad (15)$$

The function interpolation for input arguments  $x$  and  $y$  proceeds as follows:

$$z_{x, j+1} = (z_{i+1, j+1} - z_{i, j+1}) \left( \frac{x - x_i}{x_{i+1} - x_i} \right) + z_{i, j+1} \quad (16)$$

$$z_{x, j} = (z_{i+1, j} - z_{i, j}) \left( \frac{x - x_i}{x_{i+1} - x_i} \right) + z_{i, j} \quad (17)$$

$$z = (z_{x, j+1} - z_{x, j}) \left( \frac{y - y_j}{y_{j+1} - y_j} \right) + z_{x, j} \quad (18)$$

where  $x_i \leq x < x_{i+1}$  and  $y_j \leq y < y_{j+1}$

The data for interpolation can be obtained from the sequence (13) if  $i$  and  $j$  are computed. From (14), (15), and the bounds on  $x$  and  $y$ , above,

$$i = I \left( \frac{x - x_0}{\Delta x} \right) = \left( \frac{x_i - x_0}{\Delta x} \right) \quad (19)$$

$$j = I \left( \frac{y - y_0}{\Delta y} \right) = \left( \frac{y_j - y_0}{\Delta y} \right) \quad (20)$$

where  $I$  denotes "integral part of." The location of  $z_{i, j}$  is then

$$\text{loc}(z_{i, j}) = \text{loc}(z_{0, 0}) + i + Lj \quad (21)$$

and further

$$\text{loc}(z_{i+1, j}) = \text{loc}(z_{i, j}) + 1 \quad (22)$$

$$\text{loc}(z_{i, j+1}) = \text{loc}(z_{i, j}) + L \quad (23)$$

$$\text{loc}(z_{i+1, j+1}) = \text{loc}(z_{i, j+1}) + 1 \quad (24)$$

The interpolation ratios in Equations (16) to (18) are obtained as the fractional parts left after  $i$  and  $j$  are removed in Equations (19) and (20).

$$F \left( \frac{x - x_0}{\Delta x} \right) = \left( \frac{x - x_0}{\Delta x} \right) - i = \left( \frac{x - x_i}{x_{i+1} - x_i} \right) \quad (25)$$

$$F \left( \frac{y - y_0}{\Delta y} \right) = \left( \frac{y - y_0}{\Delta y} \right) = \left( \frac{y - y_j}{y_{j+1} - y_j} \right) \quad (26)$$

The table addressing operations described above can be computed more rapidly if the number of function values taken for each curve at constant  $y$ , the  $x$  interval, and the  $y$  interval are chosen as integral powers of 2; i.e.,

$$L = 2^C \quad (27)$$

$$\Delta x = 2^N \quad (28)$$

$$\Delta y = 2^M \quad (29)$$

where  $C$  is a positive integer and  $N$  and  $M$  are positive or negative integers. The divisions in Equations (19), (20), (25), and (26) and the multiplication in Equation (21) are then replaced by word shifts about the binary point.

## APPENDIX II. ANALOG COMPUTER SYMBOLIC SET AND DEBUG

A group of three PDP-1 programs (UAC-8) provide problem checks for analog computer users. The program group includes an equation loader, an interpretive mathematical and logical equation solver, and an output printer or puncher.

The user communicates with the analog check program by paper tape or typewriter using an equation language which defines the components of his computer and their relations. These equations may be defined directly from the circuit diagram by technical aide personnel.

Analog computer components are denoted by a letter and a four digit address identifying the component type, console number, and component address. Component types include amplifiers, function generators, servo multiplier-resolvers, electronic multipliers, trunks, and potentiometers. In addition, dummy codes for switches, relays, test voltage, etc. are available for devices not read by DO/IT. All the above components except potentiometers may be defined by equations relating to other components. Potentiometers can be defined by numeric value only. In addition to assigning component or test values, numeric values are used to state gains, references, and function coordinates.

Mathematical or logical operations permitted are listed in Table III.

TABLE III. OPERATIONS EXECUTED  
BY UAC-8

---

Addition
Subtraction
Multiplication
Division
Square root
Function interpolation or extrapolation
Trigonometric operations sine, cosine, and arctangent
Dead zone, symmetrical or not
Limit, symmetrical or not
Logical term control by greater or less than test as with operational relays

---

A loading program is used to enter the equations and data from typewriter or paper tape. In addition, the user can change existing equations or add to the current check system with this loader.

An interpretive equation solving program processes all mathematically defined components. Equations to be solved must represent an open loop system as is common with static, initial condition, or hold testing procedures; i.e., equations are solved sequentially, not simultaneously. The order of equations presented is irrelevant as the program automatically determines the order in which solutions may be obtained. Digitally computed values can be compared with analog components within user specified tolerances.

An output program provides typed solution values with their component codes. In addition, the output program provides the user with an optional punched or printed copy of his updated equation language.

## REFERENCES

1. BELLUARDO, R., GOCHT, R. E., and PAQUETTE, G. A., "The Hybrid Computation Facility at United Aircraft Corporation Research Laboratories," DECUS (Digital Equipment Computer Users Society) Proceedings 1963, Maynard, Mass., 261-269 (1964).
2. KRASNY, L. M., "The Functional Design of a Special-Purpose Digital Computer for Real-Time Flight Simulation," Electronic Systems Laboratory, Final Report ESL-R-118, M.I.T., August 1961.
3. HURT, J. M., "New Difference Equation Technique for Solving Nonlinear Differential Equations," AFIPS Conference Proceedings, Vol. 25, 1964 Spring Joint Computer Conference, Spartan Books, Baltimore, Md., 169-179 (1964). Washington, D. C., 169-179 (1964).
4. *Introduction to Hybrid Computation*, Training and Education Group, Electronic Associates, Inc., Princeton, N. J., 1963.
5. *Real Time Data Processing Notebook*, Beckman Instruments, Inc., Fullerton, California.

6. PAQUETTE, G. A., "Some Operational Aspects of a Hybrid Facility," presented at Eastern Simulation Council Meeting, East Hartford, Conn., December 1963.
7. "MIT-3 Macro Assembly Program for Programmed Data Processor-1," Digital Equipment Computer Users Society, DECUS No. 5, Maynard, Mass., 1962.
8. PAQUETTE, G. A., "UAC-10 Fast Bivariate Generator," Digital Equipment Computer Users Society, DECUS No. 34, Maynard, Mass., January 1963.
9. KEMBLE, T. H., "Digital Bivariate Function Generator and Display," presented at Eastern Simulation Council Meeting, East Hartford, Conn., December 1963.
10. HASTINGS, C., HAYWARD, J. T., and WONG, J. P., *Approximations for Digital Computers*, Princeton University Press, Princeton, N. J., 1955.
11. SCARBOROUGH, J. B., *Numerical Mathematical Analysis* (Johns Hopkins Press, Baltimore, Md., 1950), 2nd ed., chapter 9, pp. 192-198.
12. KOTOK, A., "MIT-1 DEC Debugging Tape," Digital Equipment Computer Users Society, DECUS No. 6, Maynard, Mass., November, 1962.
13. MILLER, J. D., "Program and Computer Preparation," presented at Eastern Simulation Council Meeting, East Hartford, Conn., December 1963.
14. JACKSON, S. F., and PAQUETTE, G. A., "UAC-14 Digital Function Generate and Display," Digital Equipment Computer Users Society, DECUS No. 65, Maynard, Mass., September 1963.

# A STROBED ANALOG DATA DIGITIZER WITH PAPER TAPE OUTPUT

*R. L. Carbrey*

*Bell Telephone Laboratories, Murray Hill, New Jersey*

## INTRODUCTION

In a group doing research on communication techniques, a wide variety of analog signals are encountered which must be analyzed. The most effective general purpose tool available for carrying out the analysis is a high-speed digital computer. Thus the problem of converting large quantities of analog data to a digital form suitable for use on the computer arises frequently. A number of high resolution analog-to-digital converters which can handle data in the kilocycle range are commercially available, but the majority of signals to be analyzed are those with frequency components in the range from a few megacycles up to a gigacycle or more. For these, ordinary real time A-to-D conversion cannot presently be used.

Fortunately experiments in this range are usually set up in such a manner that the signals can be viewed on a sampling type oscilloscope which means that the high-speed signals must be repetitive in order to permit them to be sampled many times at some relatively slow rate. By designing a system which also performs the analog-to-digital conversion on these oscilloscope samples and systematically stores the resultant binary codes in a memory circuit, the range of operations has been extended to include most of the signals which are encountered. A one-to-one correspondence between memory address and sample time is obtained by converting the memory address

codes to an analog voltage and using this voltage as the horizontal position signal for the sampling oscilloscope.

The function of the digitizer is to convert either the direct analog signal or the strobed samples to a sequence of a selected number of twelve bit binary words. Twelve bit words permit resolving a full load signal to one part in 4096 or about 0.025%. These words are then stored in a coincident core memory with a capacity of 4096 words; so its address positions are also controlled by twelve digits. These are generated by a settable counter with six digits allotted to the X location of the memory and the remaining six to the Y location. Ultimately the stored digital data is used to control a combination punch and typewriter which punches the paper tape. This tape presently serves as the transfer medium between the digitizer and the digital computer. Ultimately, direct read-out to a magnetic tape is planned.

Before punching is started, however, one should observe the results of the digitizing operation to determine whether the data meets the requirements for processing. Therefore, digital-to-analog conversion is also included. The stored data can be displayed on the oscilloscope or an X-Y plotter. The results of several related experiments can be stored for observation at one time, or one run can be divided into a number of small segments each of which can be examined in minute detail. A

lamp display section provides a visual indication of both address locations and data in digital form.

The objective was to provide a digitizer which was both flexible in operation and compact enough to permit ready movement into the laboratory where the experiment was being run. To achieve this, the control problem was simplified by using three types of manual operation. The number of words in any sequence and their location in the memory is established by manually registering memory address start and end points. Octal thumbwheel switches are used. This same type of switch is used to manually set up any arbitrary binary word which is to be written into the memory instead of a sample value. Finally, illuminated push button switches are used to select the operation to be performed and to initiate a loading, transferring, plotting, or punching operation. Although thirty-three of the push button positions provided on the console are actually used to perform a variety of operations, many of which will not be discussed, the basic manual switching functions are those indicated schematically in the block diagram discussed below. A photograph of the complete unit is shown in Fig. 1.

In order to minimize the construction and debugging time, most of the blocks were implemented with commercially available equipment all of which has proven to be very satisfactory.

#### BLOCK DIAGRAM

A simplified block diagram of the digitizer is shown in Fig. 2. The heart of the system is the coincident current core memory which serves as a buffer unit between the varied input information rates and the fixed output punching rate or any desired display rate. The particular unit selected for the digitizer has a capacity of 4096 words of sixteen bits each. (The extra four bits are used for arithmetic or flagging operations.) The self-contained access circuitry was designed to operate in the Random/Sequential Interlace mode with full cycle operation. In the random mode, the internal address register for the writing operation can be set to any selected one of the 4096

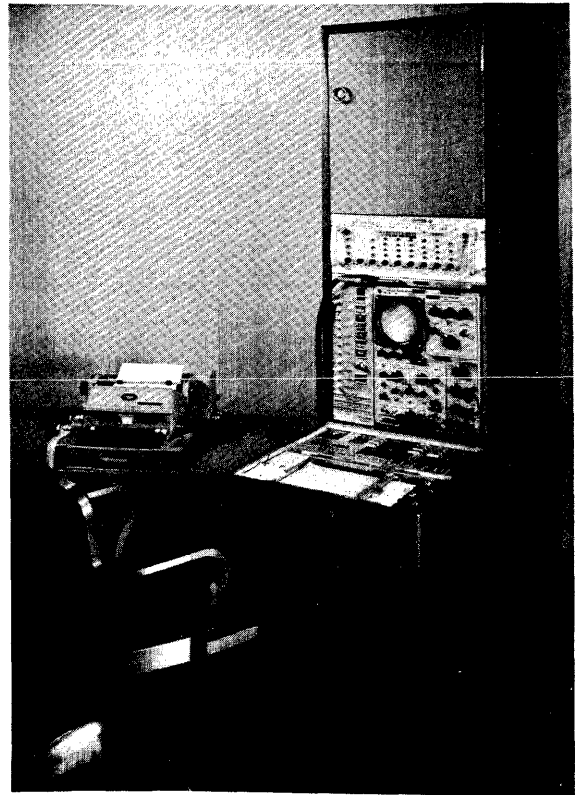


Figure 1. Digitizer and output typewriter with paper punch.

address positions by applying the desired twelve bit address word and then issuing a load command.

If the unit is then switched to the Sequential Interlace mode, it will proceed systematically from this address advancing once per load command. The unload address counter can be similarly indexed. Full cycle control requires that each load operation includes both clear and write while each unload operation includes both read and restore; therefore, the readout is essentially nondestructive. The load and unload operations can be carried on in any arbitrary sequence, but they cannot be performed simultaneously, and care should be exercised not to load in an area where wanted data is already stored. A full cycle requires 10 microseconds; so the maximum operation rate is 100,000 commands per second.

For purposes of block explanation, the memory has been divided into four parts—the load position address, unload position address, the



load into and the unload data from the common storage medium. The twelve bit position-addresses are handled in parallel as are the data bits. When switch S4 is in the position shown, the selected orderly sequence of address codes appears as the input to the X Position Digital-to-Analog converter. The resulting output is a staircase voltage with the number of steps equal to the difference between the start and stop addresses, the risers equal to  $1/4095$  of the full range, and the treads as long as the time between load commands. This staircase serves as the X axis input signal to the sampling oscilloscope replacing the internally generated "slow ramp."

As is usual in this type of oscilloscope, an internal "fast ramp" is triggered by the synchronizing signal. At the instant this latter ramp voltage becomes equal in magnitude to the address voltage, a strobe sample is taken of the Y axis input signal. The sample potential is then held until the fast ramp cycle is completed and a new trigger operation takes place.

Once a sample voltage has been established on the scope's storage circuit, a beam "un-

blanking" signal is provided which permits the sample magnitude to be displayed at the horizontal position corresponding to the memory address. This unblanking signal is also used to initiate each A-D conversion. Thus the conversion and subsequent loading operation is asynchronous with the maximum rate being limited to 40,000 conversions per second by the twelve bit A-D Converter. A conversion command causes the A-D Converter to sample and hold the signal stored in the oscilloscope. This double sampling process, once in the 'scope and once in the converter, is required because the storage circuits in the converter take longer to charge than the duration of many of the phenomena to be observed. The storage circuit in the converter must, on the other hand, be very precise if the full 4096 level resolution is to be obtained. Low-speed waveforms are connected directly into the A-D Converter; so they are only sampled once as in more conventional practice.<sup>1</sup>

Three microseconds prior to the completion of conversion, a program pulse is generated which initiates the load command for the memory. As a result, the "clear" part of the clear/write cycle is almost over by the time the

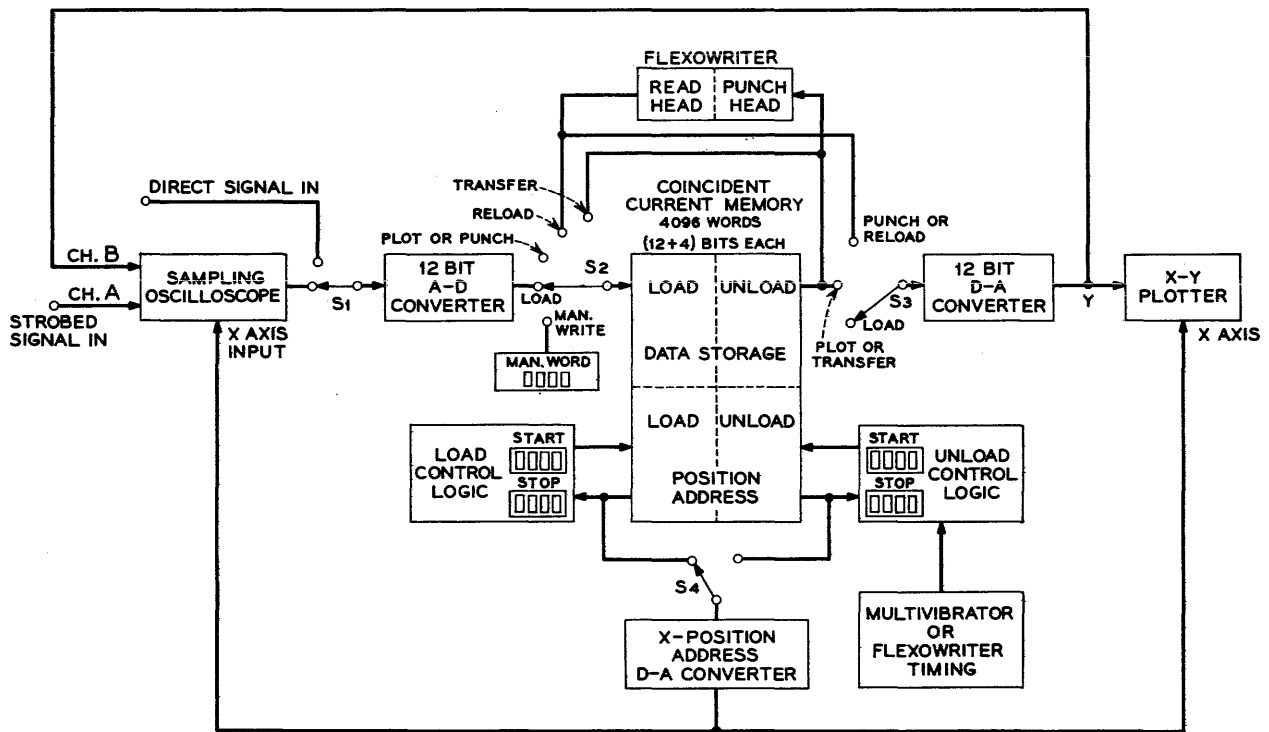


Figure 2. Simplified block diagram of Tape Output Strobed Analog Data Digitizer.

digitization is complete and the twelve bit word is available for writing into the memory. The strobe, convert, and load sequence continues until all of the assigned positions have been filled during one horizontal sweep. During setup, it is frequently desirable to observe repeatedly all or part of a display. This is accomplished in the auto-start mode by permitting a selected address to initiate a new start command thus resetting the sweep and starting over.

Once the data has been stored in the memory in digital form, the usual procedure is to check it by "playing it back." This is accomplished by transferring the switches to the plot mode, setting the Unload Start and Unload Stop thumbwheel switches to the desired range and pushing the auto-unload switch. Switch S4 connects the unload position address to the X Position D-A Converter. Output from the latter supplies the horizontal signal for both the sampling oscilloscope and the X-Y Plotter. The plotter is built into the control console under a sliding plexiglass writing surface.

Unload command rates are controlled by an internal multivibrator with a fast unload rate of about 40 kc permitting ten repeated oscilloscope displays per second of the full memory load or correspondingly more frequent displays of shorter memory sections. A variety of slower speeds can be selected for use with the plotter and visual readout. Conversion of the stored data is accomplished by switching the A-D Converter to its D-A mode.

#### PAPER TAPE PUNCHING AND CONVERSION TO MAGNETIC TAPE

A mixed alphabetic and binary data format is used on the eight column punched paper tape. Experiment titles and interspersed comments are typed in from the keyboard in a conventional manner except that an appropriate one of four characters is typed on each line just before the carriage return key is operated. The control characters used are: H (Holerith) for alpha-numeric characters, B for binary data, D for end of experiment, and X for end of tape. Anything which is not wanted in the digital computer's magnetic tape copy of the paper tape can be deleted by punching something other than one of these four.

Holes in column eight of the tape are punched only by the carriage return key.

The start of a binary data run is indicated by the letter "B" followed by the carriage return punch, and the end of the data is always terminated by another column eight punch. Each twelve bit data word is punched as two six-bit characters with the first half of the word identified by a space in column five and the second half by a hole. The odd parity function normally assigned to this column was ignored since the paper tape reader at the Murray Hill Computation Center generates its own parity bit. Instead a check on the punching operation is obtained by simultaneously reading the split words from the punched tape back into the digital-to-analog converter and plotting the result alongside the X-Y plot which was made prior to punching. Sixteen minutes are required to punch out a full memory load of 4096 words in sixty-eight feet of tape.

The paper tape reader which serves as the peripheral input device to the computer, can read a full memory load in sixteen seconds. Although the reader cannot transfer the column eight punch to the magnetic tape, it can recognize this "End of Line" indication, and use it to generate the proper flagging instruction for the paper to magnetic tape conversion program. Once the first generation magnetic tape is produced, a computer program is used to delete the parts so marked, unpack the data and text, block them into the standard computer format, and translate double-case typewriter symbols to appropriate digital computer symbols where different. When desired, a microfilm plot is provided for comparison with the original X-Y plot. The process can, of course, be reversed; so that computer data can be put into the digitizer memory by way of a punched paper tape.

#### SWEEP CALIBRATION AND LINEARITY

In the digitizer, one coded sample value is stored at each memory address position, so the standard sweep calibration is made in nanoseconds per address. The nominal 10-volt reference voltage for the X Position Address Decoder is set so that just eight complete

cycles of the oscilloscope's internal ten mc calibration wave will appear when 4000 (decimal) address positions are used with a Sweep Time/cm of  $0.1 \mu\text{sec}$ . This calibration corresponds to  $0.2 \text{ nsec/address}$ . Selection of other settings of the Ext. Horiz. Input and the Sweep Time/cm controls permits changing this over a range from  $4 \mu\text{sec/address}$  to  $0.4 \text{ picrosec/address}$ .

In normal sampling oscilloscope usage, a nonlinearity in the horizontal sweep cannot be observed because the sample is taken at a time proportional to the horizontal sweep voltage and the sweep positions the beam at the actual sweep value. The sweep can even be stopped or reversed enroute, and the waveform will be displayed with fidelity except for a brightness change. The high speed ramp and associated comparison decision must be linear, however. When the horizontal position is converted to an address as it is in the digitizer, each quantized horizontal increment is treated by the computer as a specified time increment. This will lead to distortion if the horizontal display is not linear. A calibration run can be made which will permit correcting the values in the computer, but adequate linearity is much to be preferred.

The linearity was measured by first calibrating the sweep for  $0.2 \text{ nsec/address}$  with the ten mc test wave and then switching to the 100 mc test wave. Figure 3 shows a plot of interpolated addresses or sampling intervals between adjacent positive going center axis

crossings as well as those for negative crossings. At  $0.2 \text{ nsec/address}$ , 50 addresses correspond to one cycle, and the center axis slope of a full load 100 mc sine wave is about 500 code levels per sample.

Although the actual axis crossing can be determined only to the nearest  $0.2 \text{ nsec}$  sample increment, the spacings interpolated from the coded amplitude values immediately before and after crossing should, ideally, plot on a horizontal straight line. The slight curvature of the fitted line shows an effective compression of about 1% for the combined sweep, scope amplifier, and threshold decision. The offset from the 50 sample periods shows that the initial manual calibration was also off by about 1%. This can now be adjusted more precisely with a ten turn potentiometer.

The dispersion of interpolated points indicates that both of the foregoing are masked by a combination of other factors which are loosely lumped into the 'scope sync. category. In normal sampling 'scope usage, repeated patterns are observed, and the eye or the camera averages the plotted points at any one place on the wave. The better the synchronization, the narrower will be the line of points. In the digitizer application, however, the amplitude samples are normally recorded only once at each address-quantized horizontal position. No averaging takes place. If the "synchronization" is not perfect, some of the points will be displaced slightly. A plot of the waveform will be somewhat irregular as illustrated

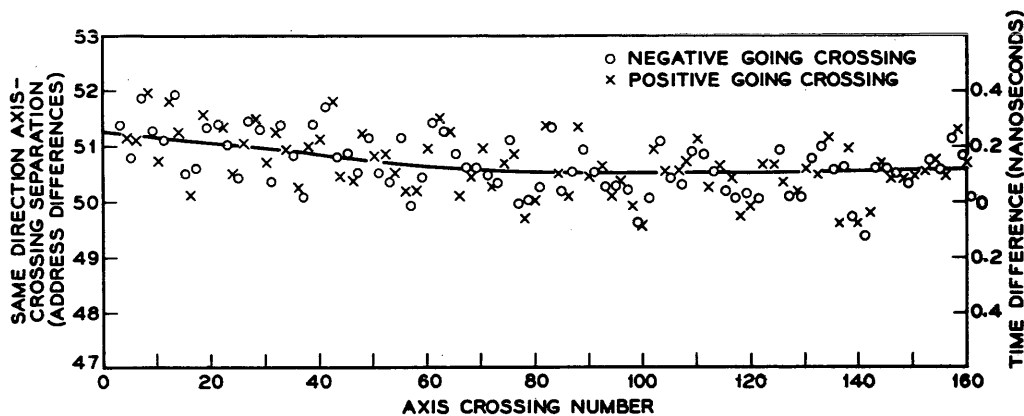


Figure 3. Sweep linearity calibration and synchronizing noise check by interpolated address differences between same direction axis crossings of a 100 mc sine wave. Straight line at address difference of 50 would be ideal.

for the 100 mc repetition rate snap diode pulse of Fig. 4a.

Small timing variations of this nature account for the dispersion shown in Fig. 3. Since the strobed sample is taken at the instant the triggered fast ramp becomes equal in magnitude to the decoded horizontal address voltage and each address is 1/4096 part of the total, the combined synchronization and amplitude decision must be better than this fraction to keep the point spread within  $\pm 1$  address.

Both experiment and synchronization noise can be averaged out by making repeated measurements of the data and averaging the results either in the computer or in the digitizer.<sup>2</sup> See Fig. 4b. The digitizer memory can store words as long as sixteen bits, thus allowing the extra four digits to be used for accumulation of sixteen repeated waveforms when full magnitude signals are coded or correspondingly more repetitions when fewer than twelve bit words are coded.

## SUMMARY

Perhaps the best summary of the digitizer's capabilities is an illustration showing a few of the varied types of data which have been con-

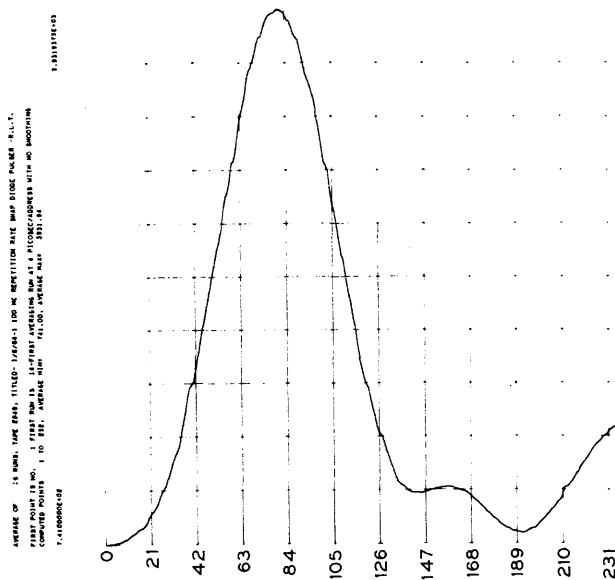


Figure 4a. One nanosec section of snap diode pulse at 4 picosec per sample with no sampling oscilloscope smoothing. This shows waveform irregularity due to noise and synchronization problems.

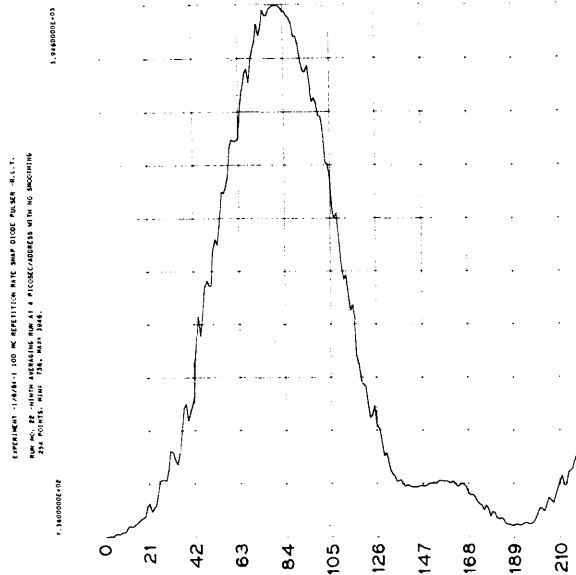


Figure 4b. Reduction of irregularity by computed average of sixteen repeated runs including that of Figure 4a.

verted from analog to digital form. The figures are microfilm plots made from precomputation tapes processed on the high speed digital computer. Experiment and run titles which were typed into the paper tape at the time the experiment was run are generated at the left of each microfilm plot along with supplementary information determined in the computer program such as the maximum and minimum value, but these are not large enough to be clearly legible in reduced reproductions. Neither are the horizontal scales which are decimal integer equivalents of the memory addresses. The plot program option to use the maximum and minimum as vertical plot limits was selected for all figures except number six.

Figure 5 shows a twenty-four sample section of a snap diode pulse similar to that shown in Fig. 4a, but here the samples are spaced only 40 picoseconds per address. The total interval is one sample short of one nanosecond. At this sensitivity, the horizontal signal is changing in large enough steps to minimize the synchronization differences. Because the microfilm plot program draws straight lines between sample points, the individual points can be observed. It is apparent from the slope at the top of the plotted pulse that the peak of the

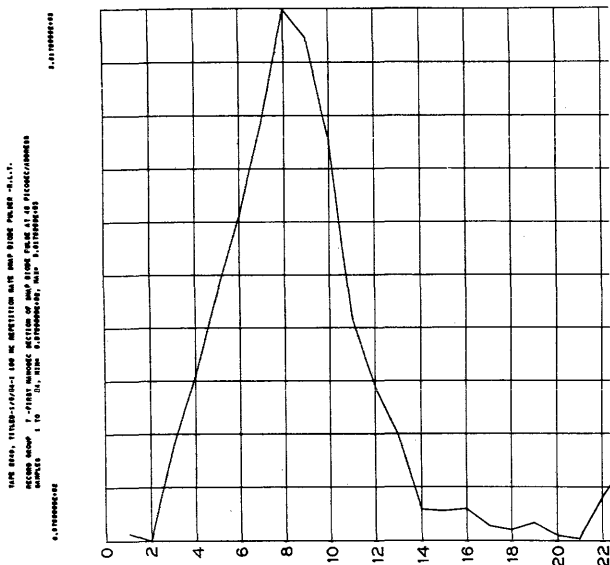


Figure 5. Twenty-four samples at 40 picosec per sample of one nanosec section of snap diode pulse. Slope indicates that peak was not sampled.

snap diode pulse was not sampled. This pulse is used as a 100 mc clock pulse for work on high speed digital circuits. It was generated by driving a silicon snap-off diode from a 100 mc source. The diode and input shunt capacitor were mounted in a 52-ohm BNC connector tee section with a short circuited plug in the

tee serving as an echo line. Fig. 4b is the most accurate representation of these 3-volt pulses which repeat at 10 nsec intervals and have a half amplitude width of 0.24 nsec.

Figure 6 shows the 10 nsec pulse to a 200-foot length of RG58A/U coaxial cable and the pulse reflected back to the input from a short circuit at the far end. The intervening 560 nsec, corresponding to decimal addresses 300 to 3100 at 0.2 nsec/address, were deleted from the strip of microfilm. Transmission characteristics of cables can be determined from this type of pulse response measurement. When the measuring probe is bridged across the near end, the same takeoff can be used for both the input and output, and the pulse must travel the full length of the cable twice before it reappears at the output.

Figure 7 is a plot of the smoothed samples of the peak output of the 40th, 424th, and 808th read pulses from a wire plated with a magnetic film.<sup>3</sup> In the test setup, a single write pulse was first applied at one location on the wire. Then about 940 consecutive read pulses of 100 nsec duration were applied. The horizontal position of the sampling oscilloscope was set manually to the peak of the resulting output pulses and the address decoder was disengaged because only the variation in peak magnitude

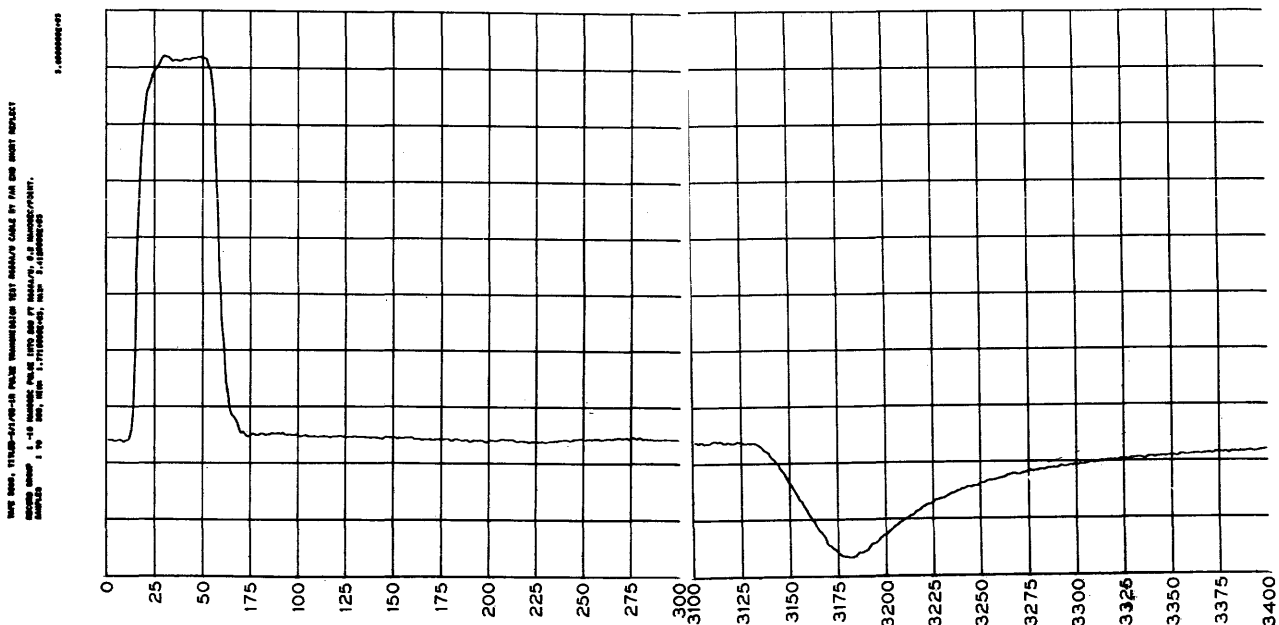


Figure 6. First 300 samples showing 10 nanosec input pulse to 200 ft. of RG58A/U coaxial cable, and last 300 samples, 0.6 usec later, showing return pulse reflected from short circuit at far end.

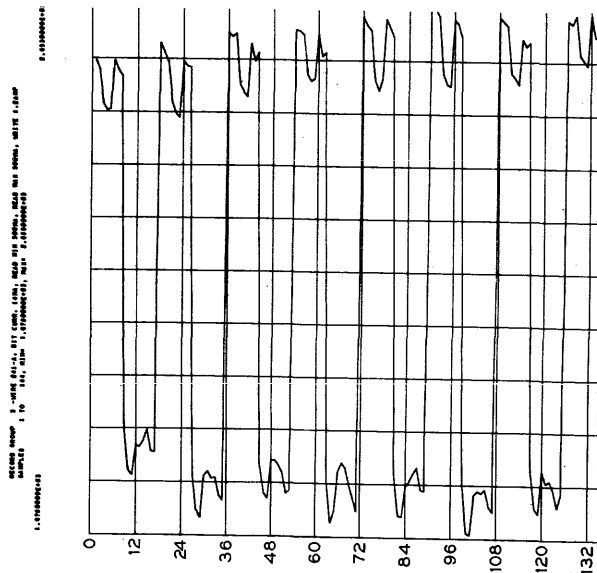


Figure 7. Oscilloscope smoothed samples of 40th, 424th, and 808th output pulses from eight adjacent sections of a wire plated with a magnetic film. With fixed sweep, samples are at pulse peak for three positive and three negative output conditions.

was wanted rather than the pulse shape. The internal smoothing control of the sampling oscilloscope was set for maximum smoothing; so the three selected samples actually loaded in the memory represented an average peak amplitude early, midway, and late in the consecutive read sequence. Following this, another block of 940 read pulses at a reduced current was applied, and the selected samples were loaded. The wire was then disturbed by simulating writing operations at other locations along the wire, and still another 940 consecutive read pulses at the original current were applied. The selected three samples were loaded. This sequence of nine loading operations was then repeated except that the initial single write pulse was negative. After the nine positive and nine negative signals were loaded, the test location was mechanically incremented and the test was repeated. The 4096 word memory permits eighteen samples to be loaded at as many as 227 wire locations. In the figure, only the first eight locations (cycles) tested on the wire are shown.

The retiming wave for regenerative pulse repeaters is usually recovered from the incoming pulse train by filtering techniques. As a

result, the amplitude and phase of the recovered wave is a function of the actual pulse pattern being transmitted and the number of repeaters in tandem through which the pulse train has passed. A single repeater can be used to simulate many aspects of a chain of repeaters by delaying a long pulse train output from the repeater and feeding the train back to the input in a self-timed closed loop.

The waveform for Fig. 8 is an envelope plot of the change in phase position of one selected pulse out of a 7500-bit pattern circulating around such a loop immediately following a change of 778 of those bits from alternate pulses and spaces to all pulses. In this case, a measurement was made only once per 750 usec round trip, and the envelope of the phase detector signal was fed directly to the A-D converter instead of to the sampling oscilloscope. Each vertical division represents  $3.8^\circ$ , and there are 14 round trips per horizontal division.

The response spikes shown in Fig. 9 were obtained by direct conversion at a 40 kc rate of the output signal from a laser analyzing cavity (interferometer) as the cavity spacing was changed by applying a 20-cycle sawtooth control signal to a one-inch barium titanate

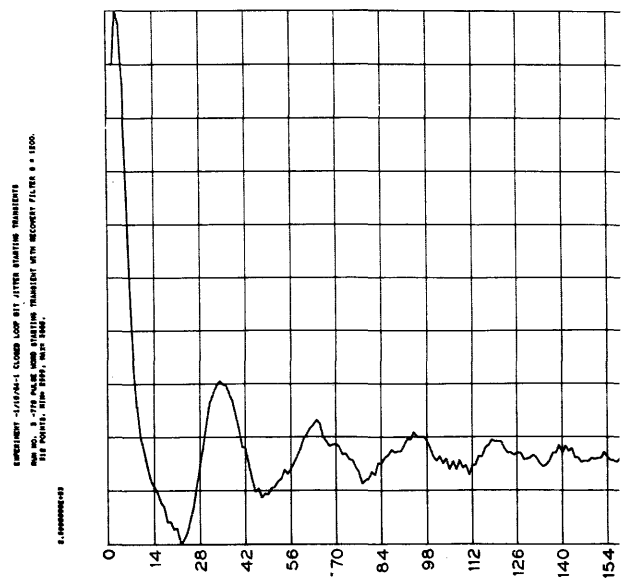


Figure 8. Example of direct A-D conversion at 750 usec per sample without sampling oscilloscope. Starting transient phase variation of one pulse circulating around a closed pulse repeater loop.

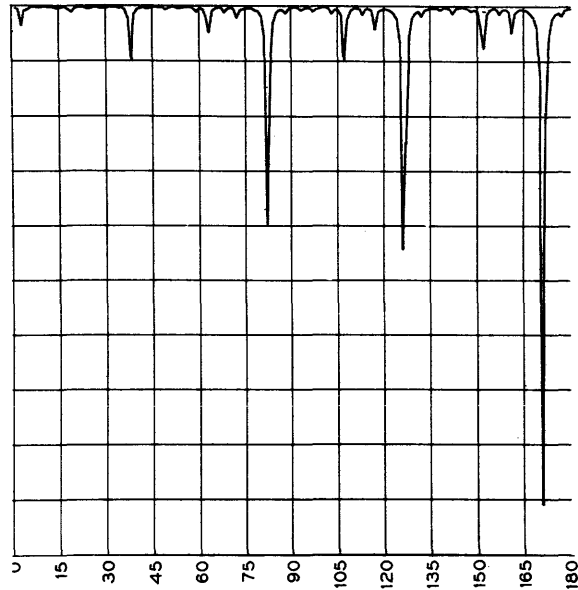


Figure 9. Digitized output signal from laser analyzing cavity as the cavity spacing is slowly changed. 40,000 direct A-D conversions per second.

cylinder with a mirror cemented to one end. Only the initial set of spikes is shown.<sup>4</sup>

#### ACKNOWLEDGMENT

The author wishes to acknowledge the helpful discussions with B. P. Bogert particularly at the Digitizer's inception and in connection with the averaging problem. The paper tape

format was developed in cooperation with J. F. Ossanna who also took care of all of the problems involving the digital computer; L. J. Cirincione programmed the paper tape reader; and R. W. Proudfoot did much of the actual assembly and debugging work. W. H. Pyle showed us how to make some minor modifications to the sampling oscilloscope which resulted in even better performance. The microfilm plots are the result of experiments by the following individuals: Figs. 4 and 5—R. L. Taylor, Fig. 6—L. O. Schott, Fig. 7—E. W. White and R. T. McGoldrick, Fig. 8—P. A. Reiling, and Fig. 9—A. J. Rack.

#### REFERENCES

1. JASPER, Les and HALL, H. K., "Improving A-D Converter Accuracy," *Electronics* 36, 38 (1963).
2. KLEIN, M. P. and BARTON, G. W., Jr., "Enhancement of Signal-to-Noise Ratio by Continuous Averaging: Application to Magnetic Resonance," *Rev. Sci. Instr.* 34, 754 (1963).
3. LONG, T. R., "Electrodeposited Memory Elements for a nondestructive Memory," *J. Appl. Phys.*, 31, 123s (1960).
4. RACK, A. J. and BIAZZO, M. R., "A Technique for Measuring Small Optical Loss Using an Oscillating Spherical Mirror Interferometer," *BSTJ*, 43, No. 4 Part 2, 1563 (1964).





# HYBRID SIMULATION OF A LIFTING RE-ENTRY VEHICLE

A. A. Frederickson, Jr., R. B. Bailey, A. Saint-Paul  
 Electronic Associates, Incorporated, El Segundo, California

## 1. INTRODUCTION

During the past ten years a great deal of research and development work has been conducted on various types of re-entry vehicles. Numerous techniques for guiding and controlling re-entry vehicles have been proposed. The purpose of this simulation is to evaluate the capability of a new flight control system for re-entry vehicles which by its nature is simple, reliable and inherently insures a safe re-entry. The ensuing discussion explains the unique control system being studied, and includes a detailed discussion of the simulation equipment required and its programming.

To obtain a better insight into the need for a simple yet reliable guidance and control system for lifting re-entry vehicles, a brief discussion of the general re-entry problem follows.

The re-entry vehicle utilized in the study is an unpowered lifting vehicle with wings, which, unlike its ballistic counterpart, is highly maneuverable and can be landed on conventional runways.

The lift and drag coefficients ( $C_L$  and  $C_D$ ) of a typical high  $L/D$  lifting re-entry vehicle are shown in Fig. 1 as a function of angle of attack,  $\alpha$ . In addition, a plot of lift to drag ratio ( $L/D$  or  $C_L/C_D$ ) is shown since it is of importance in determining the range of the re-entry vehicle. During the high velocity portion of re-entry flight the vehicle operates on the high  $C_L$  (large  $\alpha$ ) side of ( $L/D$ ) max. ( $15^\circ < \alpha < 65^\circ$ ). These larger lift coefficients yield a re-entry trajectory with lower dynamic pressure, acceleration, and temperatures. Fig. 2

defines a simplified coordinate system and symbols. During most of the vehicle flight  $r_0 + h \approx r_0$  and  $h \approx 0$ . Thus, the following approximate equations can be used:

$$h \approx -g + \frac{V^2}{r_0} + \frac{L}{m} \quad (1)$$

$$V \approx -\frac{D}{m} = -q \frac{SC_D}{m} \quad (2)$$

For a lifting re-entry vehicle the lift is large enough to allow the vehicle to fly along an equilibrium glide path where  $h \approx 0$ . Along the equilibrium glide path,

$$\frac{L}{m} = g - \frac{V^2}{r_0} = q \frac{SC_L}{m} \quad (3)$$

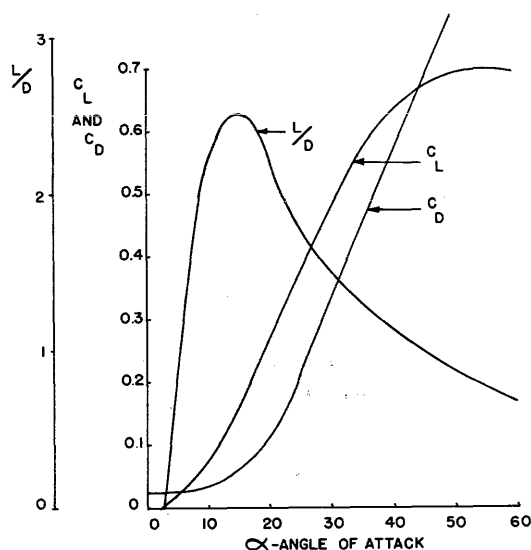
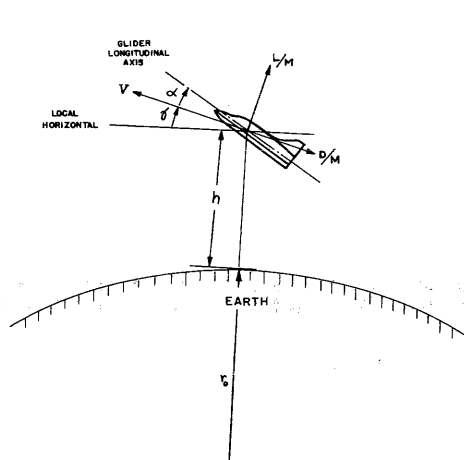


Figure 1



$h$  - ALTITUDE  
 $r_0$  - RADIUS OF EARTH  
 $r \approx r_0 + h \approx r_0$   
 $V$  - VEHICLE VELOCITY  
 $\gamma$  - FLIGHT PATH ANGLE  
 $\alpha$  - ANGLE OF ATTACK  
 $L/M$  - LIFT ACCELERATION  
 $\delta/M$  - DRAG ACCELERATION

Figure 2

where  $q$  is the dynamic pressure ( $1/2\rho V^2$ ) and  $S$  is the wing area. Thus for a given  $C_L$  there is a unique altitude-velocity profile. Fig 3 shows the altitude-velocity profile for two different equilibrium glide lines and the boundaries of the re-entry corridor. The lower boundary of the corridor is determined by the temperature and load limits of the vehicle and the upper boundary by the recovery ceiling. The recovery ceiling is defined as the maximum altitude (with altitude rate,  $h = 0$ ) from which the vehicle can recover without exceeding the temperature and load limits of the vehicle.

One of the primary re-entry problems is to control the vehicle so that temperature and load limits are not exceeded and a smooth equilibrium glide is established. The heavy trajectory shown in Fig. 3 is a typical uncontrolled re-entry with its familiar skipping oscillations which cause the vehicle to approach dangerously close to the heat limits. The temperature rate control system being simulated was developed to eliminate these skipping oscillations and reduce the peak temperatures during re-entry.

Another re-entry problem is to manage the energy of the vehicle as it re-enters so that the desired terminal point is reached. The range capability of the re-entry vehicle can be determined quite readily in the re-entry portion

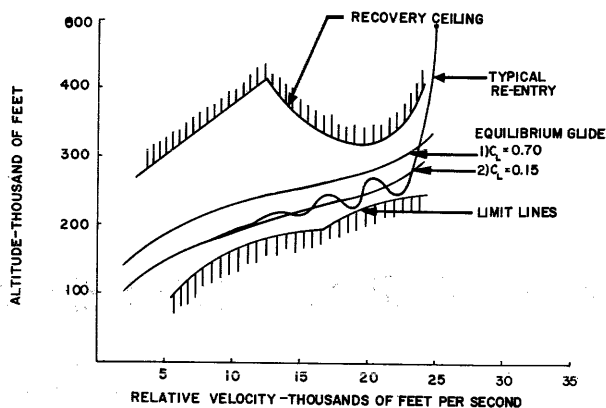


Figure 3

of flight since the vehicle is near equilibrium glide where  $h \approx 0$ . If both sides of Eq. 2 are

divided by  $L/m$  or  $g - \frac{V^2}{r_0}$  the following is obtained:

$$\frac{V}{g - \frac{V^2}{r_0}} = - \frac{D}{L} \tag{4}$$

If  $V \frac{dV}{dR}$  is substituted for  $V$ :

$$dR = (L/D) \frac{r_0 V dV}{V^2 - r_0 g} \tag{5}$$

If this is integrated from an initial velocity to zero, the range is obtained:

$$R = \left(\frac{r_0}{2}\right) \left(\frac{L}{D}\right) \ln \frac{1}{\left(1 - \frac{v^2}{r_0 g}\right)} \tag{6}$$

Thus, during the re-entry portion of flight the range is a function of velocity and  $L/D$ . Range control is accomplished by varying  $L/D$ , i.e., by varying angle of attack. From Eq. 6 it also should be noted that the range is very sensitive to initial velocity when the velocity is nearly equal to orbital velocity  $\sqrt{r_0 g}$ . For the re-entry studies made to date on the simulation, the sensitivity of range to initial velocity error is approximately 300 NM/fps.

Lateral maneuverability is obtained by banking the re-entry vehicle so that the aerodynamic lift vector is rotated, thus providing a lateral acceleration. Fig. 4 shows an energy management footprint for a typical re-entry flight. The lines of constant  $\alpha$  and  $\mu$ , bank angle, show

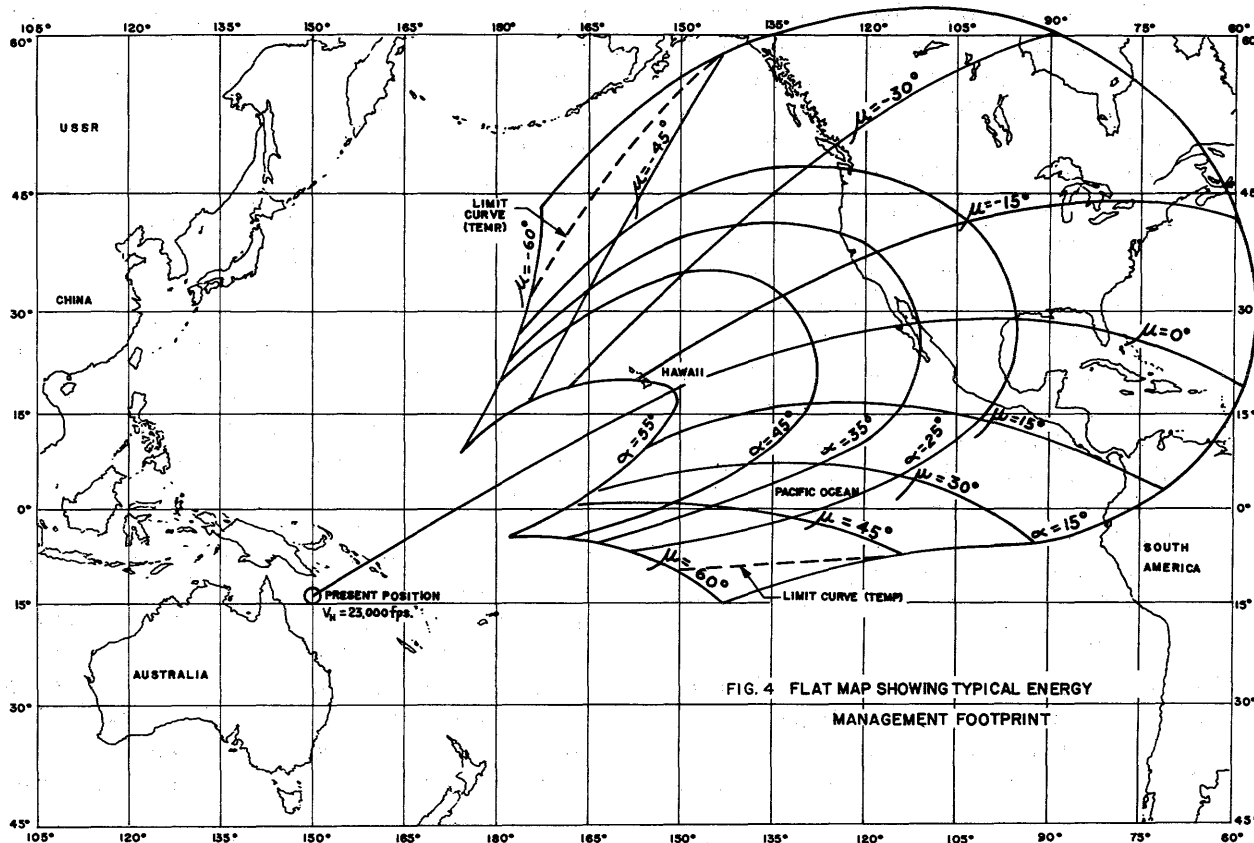


FIG. 4 FLAT MAP SHOWING TYPICAL ENERGY MANAGEMENT FOOTPRINT

Figure 4

what attitude must be maintained to reach a particular landing site. The dashed lines show temperature limits. This large maneuverability of a lifting re-entry vehicle requires a reliable guidance system which will perform accurately over the long re-entry and minimize errors at the desired terminal point. The temperature rate control system is being simulated to demonstrate its compatibility with different types of guidance systems. As will be pointed out in the next sections the TRFCS acts as a filter to the guidance signals to insure the safety of the vehicle at all times.

## 2. THE STATEMENT OF THE PROBLEM

### 2.1. Problem Background

The temperature rate flight control system (TRFCS), developed by the AC Spark Plug Division of the General Motors Corporation, is based upon the use of temperature sensors instead of conventional inertial instruments to provide both short-period stabilization and long-term guidance during the re-entry flight.<sup>1</sup> The

mathematical formulation of the re-entry problem was furnished by AC Spark Plug.

The new control system introduces several significant advantages:

- (1) Overall vehicle safety during re-entry. In the TRFCS design, successful effort has been made to separate safety of the vehicle from the task of accurate navigation. Because of the inherent nature of temperature rate feedback and certain selected limits on the control authority, the control system minimizes skin temperature peaks. The maximum "g's" and dynamic pressure are independent of initial conditions and maneuvers performed. This safety aspect of the TRFCS performance is entirely independent of the guidance commands and in fact, the TRFCS serves essentially as a filter for them.
- (2) Simple, reliable hardware. This separation of control and guidance also re-

sults in more reliable hardware. Simple thermocouple temperature sensors replace the conventional gyros and accelerometers. These sensors are used to control the flight path as well as the short-period oscillations in pitch and yaw. The only additional sensor required, besides the temperature sensors, is a vertical reference gyro, which, for the safety aspects of the re-entry, can be quite inaccurate.

- (3) Both manual and automatic modes. In case of automatic guidance system failure, the TRFCS can be controlled manually. The manual flight program to be followed by the pilot, is very simple and the resulting temperature peaks, dynamic pressure, and "g" loads compare favorably with those obtained in the fully automatic mode.

During past years, extensive simulation studies were conducted by AC Spark Plug to design the control system. A rather conventional simulation program was pursued: First analog simulations were performed to gain qualitative knowledge of the system, and to determine the practicability of this approach. Next, digital techniques were used to check the validity of the analog and to demonstrate the accuracy of the guidance through TRFCS.

In the analog simulation, a three degree of freedom simulation of the mass center of the vehicle was combined with equations describing the short-period pitch dynamics of the vehicle. Pitch axis controls and trajectory controls in three dimensions included an approximate, simple lag representation of the lateral response of the vehicle. A cockpit was used in the simulation to aid in the design of the manual system as well as the fully automatic system.

Another small analog simulation was used to study the uncoupled lateral dynamics for changing flight conditions and the lateral stabilization signals required from the TRFCS. This simulation also provided the simple lag representation used in the larger cockpit simulation study previously mentioned.

The reason for separating the simulation of the pitch dynamics and trajectory control from

the simulation of the lateral dynamics was to enable a better understanding of the pitch system alone and lateral system alone and to develop the TRFCS without the use of an extensive amount of equipment.

With the design and evaluation of the TRFCS completed for the simplified pitch plane and lateral plane the next logical step was to evaluate the system with a complete six-degree-of-freedom simulation. This would allow the investigation of cross-coupling between the pitch, lateral and roll dynamics of the vehicle.

In considering a six-degree-of-freedom simulation the question arises as to what computer or computers should be used. Past experience has shown that the conventional all analog or all digital approach has severe shortcomings. The previous all analog simulation suffered from a lack of repeatability (50 miles in range) and excessive day-to-day setup and checkout time. The digital simulation proved to be extremely slow even on a large high speed computer. Even for narrow ranges, determined by previous analog simulation, digital simulation was too time consuming, and therefore too expensive to optimize parameters. Reduction of the digital data to more graphic forms also proved to be a problem.

## 2.2 Problem Objective

The main objective of the problem is to evaluate a TRFCS controlled re-entry both in automatic and manual modes of operation. To fully explore the ability of the TRFCS to control the short period attitude of the vehicle throughout the re-entry, a complete six-degree-of-freedom simulation is required. To evaluate the ability of the TRFCS to guide the vehicle during re-entry to the desired terminal point, an accurate and repeatable simulation is required. Economy of analysis should be considered, especially in the automatic guidance studies where faster than real time simulation can be employed.

## 2.3 Computational Requirements

In order to attain the above problem objective, the following set of rigid computational requirements must be met:

- (1) High accuracy in trajectory calculations for the evaluation of the guidance capability of the TRFCS.

- (2) Very fast computing capability to faithfully simulate the high frequency parameters for the short-period dynamics of the vehicle.
- (3) Real time and faster than real time simulation for evaluation of the control system in both manual and automatic mode. (For economical evaluation of the control system in automatic mode, the time scale should be as high as possible).
- (4) Rapid day-to-day setup and checkout.

On the basis of experience gained during past simulations, it was concluded that these requirements can only be satisfied by a hybrid digital-analog computer. Such a computer would allow the programmer to choose either analog or digital solution for different portions of the problem, trading fast processing for high resolution etc.

A hybrid system makes high demands on both the digital and the analog elements as well as on the control section. The tasks to be performed by the control section, however, cannot be over emphasized. Some of these tasks are the basic timing, control operations, logic decisions and conversions which preferably are parallel with other computations. Only in this

way can the programmer truly utilize all elements of the hybrid to the fullest. To achieve these complex tasks, the "control center" should consist of programmable logic elements, such as flip-flops, counters, shift registers, parallel memory and converters. The EAI HYDAC 2400 system enabled the programmers to realize these desirable features as shown in the following sections.

### 3. HYDAC 2400 SYSTEM DESCRIPTION

In order to facilitate the understanding of the programming aspects of the re-entry problem on the HYDAC 2400 discussed in the next section, a brief description of the system is given here.

The HYDAC 2400 computer system is a successful integration of general-purpose analog and digital computers. It consists of three computers: the 231R-V analog computer, the DOS 350, and the 375 (3C DDP-24) digital computer. (For system block diagram, see Fig. 5) In the following material, a short outline is given of each section of the HYDAC 2400, and the communication between sections.

#### 3.1 231R-V Analog Computer System

The 231R-V analog computer consists of standard computing components under control

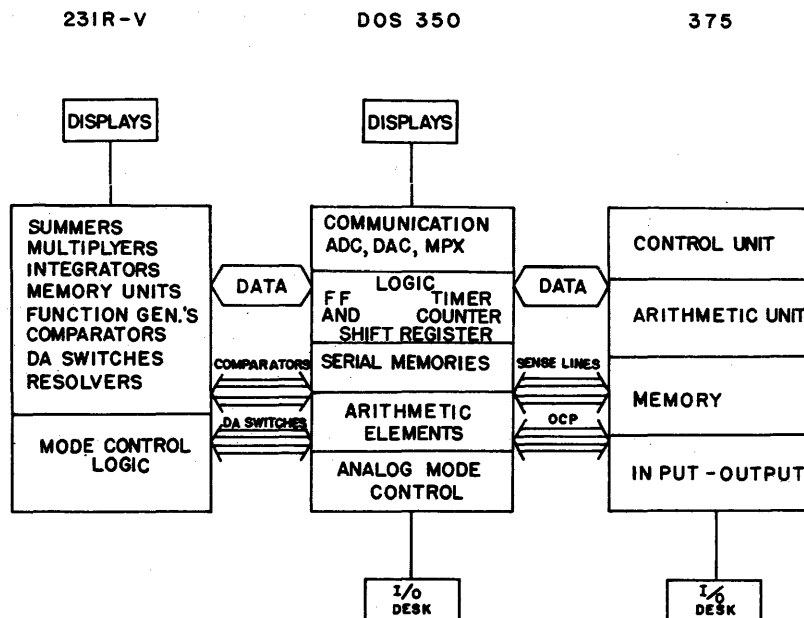


Figure 5

of a versatile, highly sophisticated control system. The heart of this control system is the analog memory and logic system (MLG). The MLG system adds a capability for programmed multiple time base and multispeed repetitive and iterative computer operation. The system utilizes electronic mode control and high-speed analog memory in order to maintain the accuracy necessary for "real-time" operation. Various forms of analog memory (microstore, track and store amplifiers) are essential for simple and efficient data transfer into the analog computer. The separate patchboard allows the controls to be pre-patched, or control commands from external digital computing equipment may be introduced to vary the program during problem solution. These control signals may affect the operation of the analog computer or some of its components many different ways:

- (1) change mode of operation
- (2) change time constant of integrators
- (3) control analog memory
- (4) set or monitor a potentiometer or other components
- (5) make logical decisions

The high accuracy solid state multipliers, resolvers, and function generators are also important contributors to the overall system performance.

### 3.2 375 Digital Computing System

The 375 is a high-speed, all solid state general purpose digital computer with 24-bit word length and sign-magnitude binary number representation. The size of the random access memory can range from 4,096 words to 32,768 words, with a memory cycle time of 5 microseconds. The 10  $\mu$  sec. addition and 31  $\mu$  sec. multiplication times are typical of the speed of the arithmetic unit.

Standard input/output capabilities include four data input/output channels: a buffered character input channel, a buffered character output channel, a parallel 24-bit input channel and a parallel 24-bit output channel. Automatic interrupt may be used in connection with any channel desired. In addition, 16 sense lines and 16 output control lines are provided

for efficient control signal communication as will be described later.

### 3.3 The DOS 350 Digital Operations System

The DOS 350 is the control and communication center for the HYDAC 2400. Its modular, functional structure of digital computing and conversion components satisfies the stringent requirements established by the basic differences of the parallel analog and the sequential digital computers.

The logic building blocks, counters, and arithmetic elements provide an extremely high speed parallel counting, decision making, control, and arithmetic capability that cannot be satisfactorily provided elsewhere in the system. The same is true for the optional storage units that may be included in the DOS 350.

The communication system provides for the flow of information throughout the system. The information handled can be classified as either whole word data or as bits or signals that must be communicated between the sections. The conversion of whole word data is performed by the analog-to-digital and digital-to-analog converters. The appropriate channels for conversion are selected and sampled by the multiplexer.

In addition to the transfer of whole word data, provision must be made for the communication of logic and control signals. The sense lines enable the DOS 350 to exercise control over the digital program. The 375, through the use of a specific test instruction (SKS) is able to determine their status and make consequent decisions. When the DOS must have immediate access to the 375, an interrupt line has to be activated. When this happens, the 375 is forced to store its present computation and proceed to a program designed to handle the interrupt situation. The system has up to eight interrupt lines each of which has a specified priority in relation to the others. Similarly, the 375 also has the ability to transmit signals to the digital section. Output control pulses are generated by the 375 and terminated on DOS 350 patch panel.

The control signal exchange between the 231R-V and DOS 350 computers is accomplished by the use of analog comparators and

the digital-analog (DA) switches. The analog comparator permits the DOS 350 to be made aware of the occurrence of certain events within the analog section. This circuit generates a logical ONE on the DOS 350 patchboard when the algebraic sum of two selected input voltages exceeds zero. The DA switch is basically an analog gate under DOS control, by which the path of an analog signal may be open or closed. Thus the logical program is able to accomplish numerous functions, such as changing parameters, providing automatic rescalings, etc.

While the above description of the HYDAC 2400 is extremely brief, it is hoped that it would aid the reader in understanding subsequent discussions. More detailed writings on the systems are available from numerous EAI publications.

4. PROBLEM MECHANIZATION

All advantages of the hybrid computer are in vain, unless very careful consideration is given to the programming of the physical system

under study. This phase of the simulation transforms the general purpose computer into a simulator of the specific physical system.

4.1 Allocation of Tasks

The first step towards a successful hybrid program is the allocation of tasks on the computer. The underlying philosophy is to subdivide the physical system into sections, and assign these to various parts of the computer, where their speed and accuracy needs are best satisfied.

The assignment of these sections to various elements of the HYDAC 2400 is shown in Fig. 6. The attitude control loop, consisting of the vehicle rotational dynamics, the TRFCS and the short period sensor equations, are programmed on the analog section. In addition, the displays and cockpit simulator are tied into the analog since continuous analog signals are required. The translational equations of motion, long period heat sensor equations and guidance equations are programmed on the 375 because of the stringent accuracy requirement.

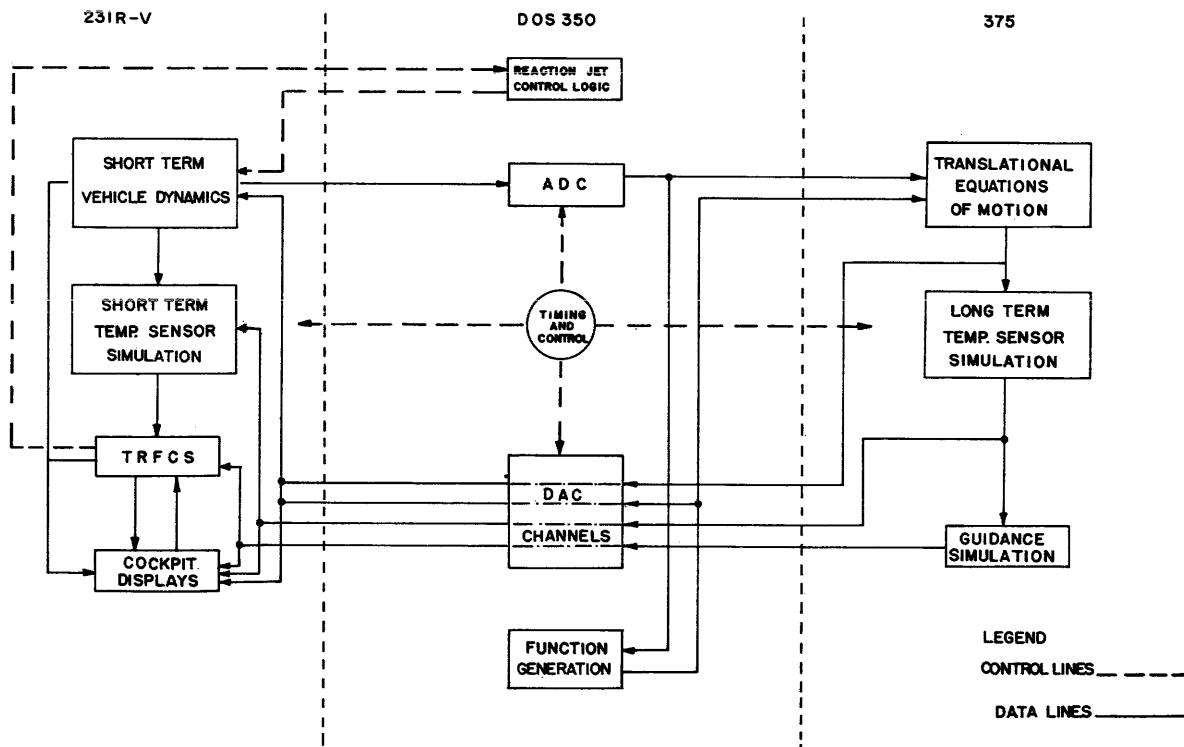


Figure 6

The DOS 350 provided the master timing, data conversion, function generation, and reaction jet control logic.

The DOS 350 timing and control is essential because of the operational differences between the analog and digital sections. The analog is a parallel continuous computer with computing time independent of problem size. The digital is a serial, discrete interval computer with computing time directly dependent on the size of the problem. The DOS, through its timing and controls, synchronizes the calculations on each computer and controls the flow of information between sections. Function generation and the reaction control jet logic are ideally suited to the DOS since these operations can be performed rapidly in parallel with the 375-general purpose digital computer, so that the digital computation time is minimized.

#### 4.2 DOS 350 Program

Fig. 7 shows a detailed block diagram of the DOS program. The DOS performs the five following functions: 1) Timing, 2) Mode Control, 3) Data Transfer, 4) Function Generation, and 5) Reaction Jet Control Logic. These functions are described in the following sections.

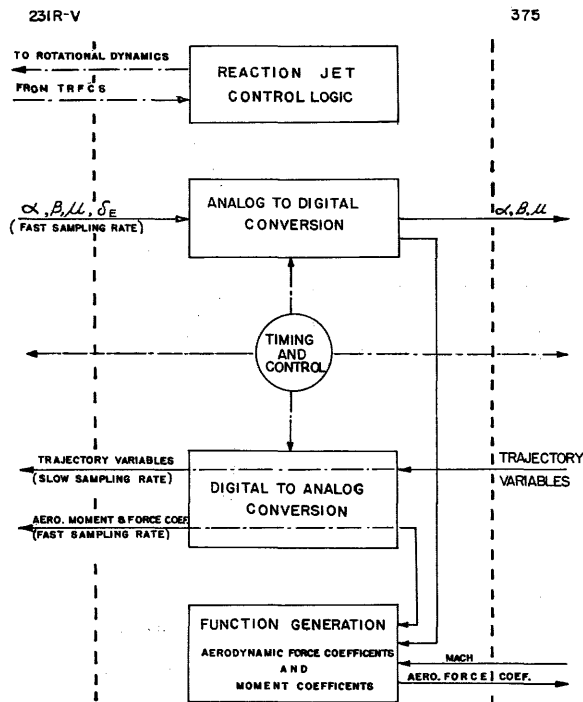


Figure 7

#### 4.2.1 Timing

Timing is required on a hybrid computer for the following reasons:

- (1) To synchronize the calculation time in the digital program to the physical time scale used on the analog computer. This synchronism is accomplished by sending a periodic master time pulse T1 which initiates the calculations for each step in the digital computer.
- (2) To time information transfers between the analog section and the digital section. Not all transfers are at the same rate, since the serial memories of the DOS are used for function generation of aerodynamic moment and force coefficients. These variables must be transferred at a high rate, since the functions are used in the short period rotational dynamics of the system. On the other hand, those variables relating to long term trajectory variations are transferred to the analog section at a lower sampling rate.

The general timing of the simulation is shown in Figure 8. Timing is controlled from the DOS by a BCD counter called the master timer. This counter counts 25 accurately timed pulses occurring every 2 ms, resets, and repeats indefinitely. The reset signal which occurs every 50 ms is the master timing signal T1 which is used to synchronize the analog and digital. The timing for the A to D and D to A conversion is obtained by decoding the appropriate states of the master timer.

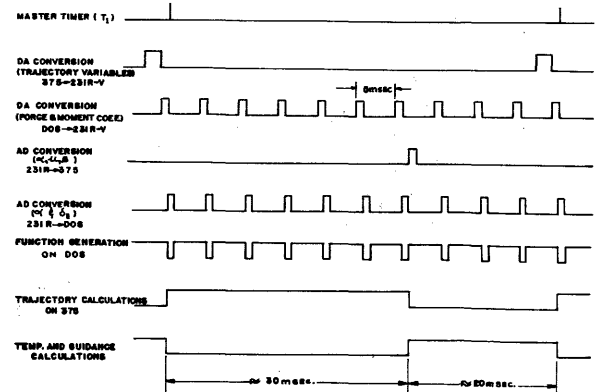


Figure 8



#### 4.2.2 Mode Control

A very important function of the DOS 350 is to control the modes of operation of the system. Communication between the DOS and the 375 occurs through 16 sense lines which are set from the DOS and sensed on the digital section. In the other direction, eight flip-flops on the DOS can be set from the 375 with special OCP instructions (output control pulses). Of these, four can also be reset from the digital console.

All modes are controlled by pushbuttons from the DOS 350 and the following list summarizes the state and function of the analog and digital sections when the indicated pushbutton is depressed.

- (1) IC (initial condition)
  - a. 231R in IC
  - b. 375 in IC

In IC 375 goes through all computations with the exception of the integration routine.
- (2) TYIC (type in initial conditions)
  - a. 231R in IC
  - b. 375 is ready to accept new initialization data from the typewriter. From this mode the program returns automatically to the IC loop.
- (3) TYTI (type titles)
  - a. 231R in IC
  - b. The 375 types out title block and line headings for the 26 variables chosen for print out.
- (4) TRA (transfer only)
  - a. 231R in IC
  - b. 375 in IC

This mode is for single stepping through the D to A and A to D transfers and was found very useful for problem checkout.
- (5) TDAC (transfer D to A check)
  - a. 231R in IC
  - b. In the 375 a fixed block of data consisting of positive and negative maximum values (corresponding to  $\pm 100v$  on the analog) is transferred continuously. This was found very convenient for a quick check on the D to A conversions.
- (6) OP (operate)
  - a. 231R in operate
  - b. 375 in operate

This is the normal mode of operation of the system.
- (7) TS (time scale)
  - a. 231R in operate
  - b. 375 in operate

This pushbutton changes time scales, there being two arbitrary time scales available, i.e., real time and twenty times real time.
- (8) HOLD
  - a. 231R in hold
  - b. The 375 performs all the calculations contained in the operate loop except those in the integration routine.
- (9) DUMP
  - a. 231R in hold during actual dump operation otherwise in IC or operate as previously selected.
  - b. 375 goes to output routine at periodic intervals determined by TP, the print-out time interval. The digital jumps to the output routine and types out the present values of the 26 variables. The DUMP command can be given in either IC, HOLD or OP and the system resumes in which ever mode it was in at the time of execution.

#### 4.2.3 Data Transfer

The data transfer for this problem is very demanding since two basic sampling rates are required, one for the short period aerodynamic functions and the other for the long period trajectory variables. The data transfers will be discussed in two parts: 1) the analog-to-digital conversion and 2) the digital-to-analog conversion.

##### 1) Analog-to-Digital Conversion

Two variables,  $\alpha$  and  $\delta_E$  are converted every 5 milliseconds since they are used on the DOS for function generation. Once every 50 milliseconds  $\alpha$ ,  $\beta$ , and  $\mu$  are converted and transferred to the 375 for use in the long period trajectory calculations. Fig. 8 in the previous section shows the timing for these two different types of conversions.

Upon command from the master timing program on the DOS, the ADC control logic increments the multiplexer to the proper channel and sends a convert signal to the AD converter. When the conversion is complete, the converted data is either loaded into a serial memory on the DOS for use in the function generation program or sent on to the 375 for use in the digital calculations.

## 2) Digital-to-Analog Conversion

Updated values of the aerodynamic force and moment coefficients which are generated on the DOS are transferred to the analog every 5 milliseconds. The trajectory variables (altitude, temperature, guidance errors, etc.) are transferred from the 375 to the analog once every 50 milliseconds. Fig. 8 in the previous section shows the timing of these transfers.

The digital-to-analog transfers are initiated by the DOS control logic upon command from the master timer. If data is to be transferred from the 375 the DOS sets the output channel ready flip-flop on the 375. The 375 will then output information to the buffer register and the DAC logic on the DOS loads it into the proper DA converter. To minimize digital computer time on the 375, four data words are loaded into the four DAC's (under the control of the DOS) and while this data is being converted and demultiplexed on the analog under control of the DOS, the 375 is formatting the next four words to be transferred. By use of this technique, the total processing time consumed for the transfer operation (except for formatting) is only 200  $\mu$  secs. The force and moment coefficients are transferred by loading the four DAC's from the serial memories which store the latest computed values of these coefficients. The data is then converted and demultiplexed on the analog computer. The process is then repeated for the last four coefficients.

### 4.2.4 Function Generation

The present simulation requires the generation of eight aerodynamic force and moment coefficients. Among these are four functions of one variable ( $C_{n\beta}(\alpha)$ ,  $V_{1\beta}(\alpha)$ ,  $C_{n\delta_a}(\alpha)$ ,  $C_{l\delta_a}(\alpha)$ ), and four functions of two variables, ( $C_m(\alpha, \delta_E)$ ,  $C_L(\alpha, M)$ ,  $C_D(\alpha, M)$ , and  $C_{Y\beta}(\alpha, M)$ ).  $\alpha$ ,  $M$ , and  $\delta_E$  are angle of attack, mach number, and elevator deflection respectively. These

functions are generated on the DOS for the following reasons:

- (1) The functions of two variables are extremely difficult to generate in the analog section and would at best require a number of sums and products of functions of one variable. The functions of one variable could be generated on the analog but were programmed on the DOS because of the ease of setup and the speed at which the functions could be changed to study other vehicle configurations.
- (2) The functions are also very difficult to generate on the 375 because of the fast sampling rate required. The sampling rate of the functions should be at least 10 samples per second since they are used in the short period attitude loop of the vehicle. Thus in the twenty times real time mode, the functions should be sampled at least every 5 milliseconds. To meet this requirement the 375 would have to be interrupted to compute these functions a great number of times during the major computation cycle. (In this problem the major computation cycle is 50 milliseconds). It is estimated that the computation time required to compute these functions on the 375 is about 4 milliseconds; therefore, if the program is stopped every 5 milliseconds for function generation, only one millisecond of the five millisecond interval can be spent on the solution of the long period problem. Hence the major computation cycle would be 5 times longer and twenty times real time runs would be impossible.

The function generation is accomplished on the DOS by the use of 2 serial memory units which allow 32 curves with 16 points each to be stored: Linear interpolation between points is utilized, and for the functions of two variables, several curves are used with linear interpolation between them. For the present problem the four functions of one variable utilize 1 curve each, and the remaining four functions of two variables are generated with 7 curves each.

Each function is computed once every 5 msec so that in a 20 times real time run this would correspond to 10 samples per second.

4.2.5 Reaction Jet Control System

Since portions of the flight are outside the atmosphere in regions where the dynamic pressure is too small to make use of aerodynamic surfaces for control, a reaction jet system is required. To make the reaction jet control easier, a pulse width modulation scheme is used which makes the moment proportional to the error signal. To prevent continuous pulsing of the jets even for small errors a deadzone is also built into the controller.

The logic operations required to decode the commands to activate the proper jets, and to simulate the pulse width modulator are performed on the DOS in parallel with all other operations in the digital section. The simulation of such a system by conventional analog techniques is a formidable task (dozens of switches and relays would be required). The use of the 375 for such an operation would result in a significant increase in digital computation time because of the large number of logic operations and the fast sampling rate required.

4.3 Digital Calculation on the 375

The digital program was written to make the digital calculation time as small as possible and to stay within a memory capacity of 4,096 words. The material which follows gives a description of the equations which are solved on the 375 and the details of the digital computer programming.

4.3.1 Summary of the Digital Calculations

Fig. 9 shows the block diagram of the system of equations to be solved on the 375.

The translational equations of motion are solved in a local horizontal coordinate system with axes along the north, east, and radial directions. The gravity and altitude calculations are based upon an oblate model of the earth, and the U.S. Standard Atmosphere, 1962, was stored in table form on the 375. Most of the equations are conventional and quite straightforward, with the exception of the heat transfer, temperature, and guidance equations which will be discussed in more detail.

The heat transfer and temperature equations are calculated on the 375 because of the large number of complex functions required (i.e., cosine, log, exponential, square root, etc.) as

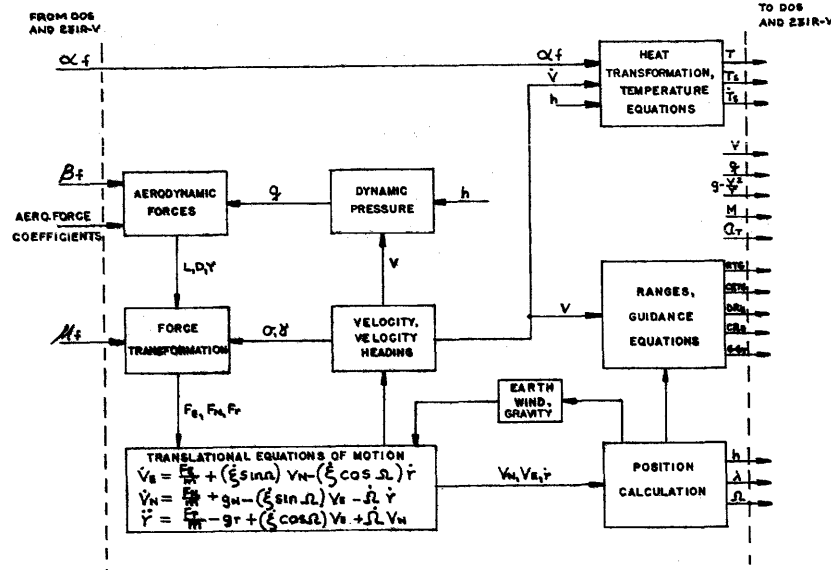


Figure 9

illustrated by a few of the equations which follow. Fortunately these equations describe slow-varying trajectory dependent quantities so that the slower serial computation on the digital is acceptable.

$$q_{CS} = \frac{2.70898}{\sqrt{R}} \sqrt{q} \left( \frac{V}{10^4} \right)^2 \quad (7)$$

$$q_{RS} = 144.9 R \rho^{1.57} \left( \frac{V}{10^4} \right)^{17} \quad (8)$$

$$T_s = \left[ \frac{1}{\epsilon \sigma_B} (q_{RS} + q_{CS}) \right]^{1/4} \quad (9)$$

$q_{CS}$  and  $q_{RS}$  are the conductive and radiative heat transfer rate at the stagnation point.  $T_s$  is the temperature at the stagnation point.

Because of their critical accuracy requirements, the guidance calculations are also performed on the digital section. These can be separated into two parts: (1) the determination of the range-to-go, (RTG), heading angle to target, ( $\sigma_T$ ), cross range-to-go, (CRTG), and heading error, and (2) the generation of down range and cross range errors ( $DR_E$  and  $CR_E$ ), which are sent to the TRFCS and subsequently controlled to zero. The range-to-go, cross-range-to-go, and heading angle to the target are

determined from spherical trigonometric relations. The down range error is the difference between the range-to-go and the desired range-to-go. The cross range error is calculated in a like manner. The desired range-to-go and cross-range-to-go are stored as functions of relative velocity.

#### 4.3.2 General Description of Digital Program

In addition to solving the translational temperature, and guidance equations, the digital program must accept mode control and timing commands from the DOS, scale variables as they are transferred in and out of the digital section, and finally, provide various input-output functions as described in the DOS section.

Fig. 10 shows the flow diagram of the digital program. The executive program is discussed in detail in the section that follows. The dashed lines in the diagram link together the calculations which are performed when the IC mode of operation is selected. All operations are performed except the Runge-Kutta integration shown in block 9. When the operate mode is selected, the integration loop is entered, and four passes around the loop are made (see description of numerical integration which follows). At the end of four passes, the positions

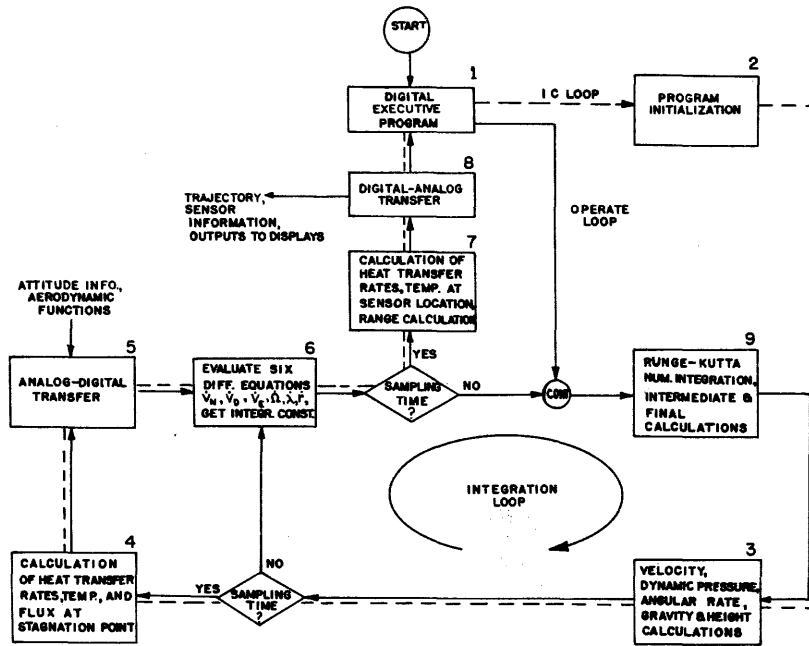


Figure 10

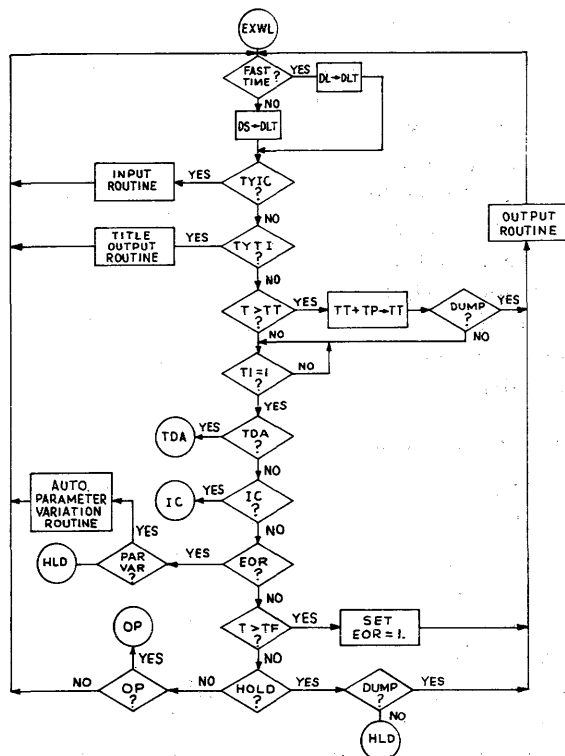


Figure 11

and velocities, temperatures and guidance functions are updated, and the analog-to-digital and digital-to-analog transfers are made. The executive loop is then re-entered, and the cycle continues until a different mode is selected. The following section discusses this executive loop in detail.

### 4.3.3 Digital Executive Program

The main media of communication, through which the 375 receives the mode commands and timing from the DOS 350, is the digital executive program. (See Fig. 11). This program is essentially a chain of test instructions, through which the digital section interprets the DOS 350 commands and executes them by jumping to the respective portion of the stored program.

Since some of the symbols are not identified in Fig. 11, a short explanation of the executive program follows. The first test determines whether fast or slow time scaling is requested. If fast time scale is requested the increment size for integration (DLT) is set equal to

1 sec; otherwise it is set to 0.05 sec. The subsequent two test instructions simply switch the digital program into the TYIC or TYTI modes as so commanded by the mode control logic. (See Section 4.2.2 for a definition of these modes). The  $T > TT$  decision determines if it is time for the periodic dump of pre-selected digital parameters. The variable TT holds the time for the next dump, say 20, 30, 40 etc. seconds. TP is the printout time interval and increments TT when the dump time is reached. If the dump switch is on and  $T > TT$ , the output dump is performed; if not, the program proceeds to the next decision. The next test keeps the entire digital program under the timing control of the master timer. The 375 cannot proceed further until the next T1 pulse indicates the beginning of the next computational cycle. Further testing takes place only after the T1 pulse arrives. The TDA test enables the programmer to place the digital program into a loop where only DA transfer is performed. The IC test is self explanatory. The next two tests are used to sense the end of run, produce a printout of the final conditions, and then put the digital computer in the hold mode waiting for a mode command from the DOS. The last few tests are self-evident and require no explanation with the exception of the seemingly superfluous second test for dump. The test for dump in hold mode makes it possible to dump any time by putting the computer in hold mode prior to the dump request, in addition to or in lieu of periodic dumps. In this manner the programmer can determine parameter values with digital accuracy at any time during the simulation.

### 4.3.4 Numerical Integration

Unquestionably one of the most complex and time-consuming parts of the digital program is the solution of the six simultaneous differential equations which provide the position (altitude, latitude, and longitude) and the velocity (radius rate, velocity east, velocity north) of the vehicle.

The numerical technique selected for this calculation was the fourth order Runge-Kutta method. The basic method as applied to a single differential equation is described briefly.

Let  $\frac{dy}{dx} = f(x, y)$  represent any first order

equation and

$$K_1 = hf(x_n, y_n) \quad (10)$$

$$K_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{K_1}{2}\right) \quad (11)$$

$$K_3 = hf\left(x_n + \frac{h}{2}, y_n + \frac{K_2}{2}\right) \quad (12)$$

$$K_4 = hf(x_n + h, y_n + K_3) \quad (13)$$

$$\Delta y = \frac{1}{6} (K_1 + 2K_2 + 2K_3 + K_4) \quad (14)$$

then  $x_{n+1} = x_n + h$  and  $y_{n+1} = y_n + \Delta y$

The increment for the second interval is computed in a similar manner by means of the same formulae. The preceding technique was extended to a system of six equations in an obvious manner. The fourth order Runge-Kutta method was chosen since it is self starting, accurate, and will allow the use of large time steps in the integration. The self starting feature simplifies the programming and reduces the memory requirement imposed to start other numerical integration techniques. The error of the fourth order Runge-Kutta method is of the order  $h^5$ , where  $h$  is the step size, and will allow the use of large time steps. While the processing time is considerably more than second order techniques the increase in allowable step size more than compensates for it.

While preparing the digital program an effort was made to combine the calculations due to the integration with other necessary calculations to minimize the processing time as well as the memory space requirements. This approach makes it difficult to trace the integration on the flow diagram of the digital program (Fig. 10). For example, the Runge-Kutta constants are determined in block 6 and the variables are incremented for the calculation of the next constant in block 9 (such as  $V_E + \frac{K_1}{2} \rightarrow V_E$ ) etc., and the equations are evaluated with the incremented variables in blocks 3, 4, 5, 6, and 7. The new value of the integral at the end of each period is calculated in block 9.

#### 4.3.5 Utility Programs

In order to satisfy the various data handling needs (type in, type out, punch tape, convert binary to decimal, etc.) an excessive amount of digital programming must be done. Fortunately, all these programs are already available, tested, and clearly described in the 375's "software package". This is no small feat as one considers that the above programs, together with the numerous subroutines, normally add up to about 75% of all digital programming required.

#### 4.4 Analog Section

The analog computer is the one link of the HYDAC 2400 system ideally suited for control system simulation by virtue of its capability for high speed, parallel computation and its input-output flexibility. Output data can be displayed in a multitude of forms such as X-Y plots, strip chart plots, oscilloscope displays, auxiliary meters, etc. Special purpose input equipment can easily be adapted for compatible operation with the analog computer.

The block diagram of Fig. 12 delineates the mechanization of the analog program.

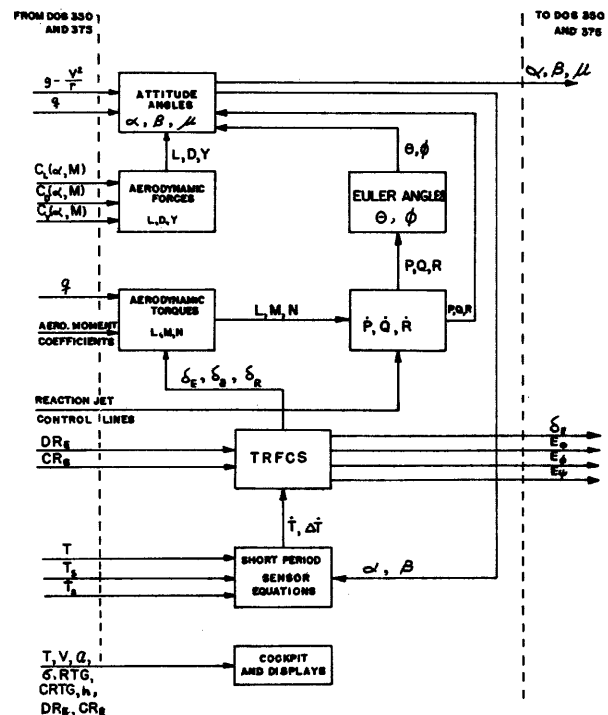


Figure 12

As previously mentioned the TRFCS and rotational dynamics were programmed on the analog due to their rapidly varying characteristics. When the vehicle is in the atmosphere, the vehicle attitude is controlled by aerodynamic control surfaces. The aerodynamic control moments are calculated from the surface deflections and the moment coefficients generated on the DOS 350 and transferred to the analog where the moments are produced and fed into the angular acceleration equations. The angular rates, generated from the angular acceleration equations, are used to calculate the Euler angles  $\theta$  and  $\phi$ . These angles are used to resolve gravity into the body axis for use in the  $\alpha$  and  $\beta$  equations.

The redundant force equations for  $\alpha$  and  $\beta$  are computed on the analog since they are required in the short period sensor equations which follow. The aerodynamic force coefficients in the  $\alpha$  and  $\beta$  equations are generated on the DOS 350.

The short period sensor equations are:

$$T = T_s (1 - .1875 \alpha^2) - .375 \alpha \times T_s \quad (15)$$

$$\Delta T = .021 (T_s \beta + T_s \beta) \quad (16)$$

$T$  is the temperature rate at the nose sensor; and  $\Delta T$  is the temperature rate differential between the two wing sensors. The stagnation point temperature information  $T_s$  and  $T_s$  which is used in these sensor equations is calculated in the digital section. The  $T$  and  $\Delta T$  are used in the TRFCS control equations shown below:

$$\delta_E = f_1 (T) \quad (17)$$

$$\delta_a = K_1 (\mu - \mu_c) + K_2 P + K_3 \delta_R \quad (18)$$

$$\delta_R = f_2 (\Delta T) + K_4 \delta_a \quad (19)$$

$$\mu_c = f_3 (T) \quad (20)$$

The  $T$  and  $\Delta T$  terms supply damping to the control equations alleviating the heating problems associated with and undamped trajectory. Closed loop guidance is achieved by adjusting the pitch axis controls with a compensated down range error,  $DR_E$ , and adjusting the  $\mu_c$  to compensate for the cross range error,  $CR_E$ . The pilot can manually control the range by adjusting his temperature rate profile to eliminate the displayed down range and cross range errors.

A cockpit simulator was utilized to evaluate the TRFCS in the manual mode, and is trunked

directly to the analog computer. Computer outputs drive display meters on the TRFCS CONTROL PANEL which monitor the vehicle temperature rate, attitude, accelerations, control surface position and range errors.

As seen in the analog program block diagram, many of these parameters are transferred from the digital section. Some of the parameters, though not necessary for control, indicate trajectory status and therefore maintain the pilots confidence in his control information. The above parameters and other pertinent data are recorded on strip charts and X-Y plots for permanent record of each flight.

## 5. CONCLUSION

A large number of test trajectories were used to evaluate the performance of the hybrid simulation. An extensive check of the digital trajectory calculations with integration time step sizes from 0.05 sec to 2.0 sec was made. Comparison with trajectories calculated on an IBM 7094 revealed errors less than 100' in altitude, 0.1 fps in velocity, 1NM in range in the worst case (a 45 minute orbit transfer case). Repeatability tests on the complete system including the flight control system were also made on unguided re-entry trajectories using the TRFCS to hold a fixed angle of attack (except for small excursion for damping). The range dispersion at the terminal point was found to be less than 1.5 nautical miles. When this is compared to the open loop sensitivity of range to angle-of-attack (365 NM/degree) it indicates that the angle-of-attack error (including analog and conversion) is less than 0.004. This corresponds to less than 20 mv error in equivalent analog voltage.

The true test of the simulation program is the overall performance and cost compared to other simulation techniques. For this purpose a careful study of the equipment complement required for an all analog and all digital was made. Table I shows the equipment complement for the hybrid simulation and an estimate for the all analog and all digital methods. Table II shows the cost comparison of the three simulation methods. The cost per hour is based on the relative purchase price of the equipment. The analog equipment is 1.5 times the hybrid and the digital .75 times the hybrid.

TABLE I. EQUIPMENT COMPLEMENT FOR HYBRID, ANALOG AND DIGITAL SIMULATION

HYBRID	ANALOG	DIGITAL
1-231R-V with 205—amplifiers 150—servo set pots 38— $\frac{1}{4}$ square mult. 3—electronic resolvers (each contains 4— $\frac{1}{4}$ sq. mult. 2—sine-cosine gen.)  1-DOS-350 with 1—serial memory field 1—counter field 1—arithmetic field 3—logic fields 1—Input-Output field 1—AD converter 1—multiplexer 4—DA converters  1-DCS 375 with 4K—memory 16—sense lines 8—OCP lines 2—interrupt lines	3-231R-V's with 600—amplifiers 350—servo set pots 90— $\frac{1}{4}$ square mult. 30—servo set diode function generators 10—electronic resolvers (each contains 4— $\frac{1}{4}$ sq. mult. 2—sine-cosine gen.)	1-DCS 375 with 8K—memory 16—sense lines 8—OCP lines 2—interrupt lines  1-DOS 350 with 1—counter field 2—logic fields 1—Input-Output field 1—AD converter 1—multiplexer 20—DA converters

The cost comparison is made on the basis of 100 hrs of running time on the hybrid system. It is estimated that half this time was run at real time and one half at 20x faster. While both the hybrid and analog are capable of 20x real time operation the digital is capable of

only 2x real time. This is based on a digital calculation time for the short period of 40 milliseconds, which allows 10 samples per second for the short period when running 2x fast.

From Table II is seen that the hybrid simulation of the re-entry problem reduced costs

TABLE II. COST COMPARISON BETWEEN HYBRID, ANALOG, AND DIGITAL

ITEM	HYBRID	ANALOG	DIGITAL
Rel. Cost/hr of operation (assured to be directly proportional to purchase price)	1 unit	1.5 units	.75 units
Cost of Real Time Runs	50	75	37
Cost of Fast Time Runs	50 (20x fast)	75 (20x fast)	375 (2x fast)
Cost of Day to Day Setup, checkout & Downtime	10 (10%)	50 (30%)	20 (5%)
Total Cost	110	200	432
Total Rel. Cost	1	1.82	3.93



over an all analog method by 50% and over an all digital by 75%. This saving can be attributed to the high accuracy digital calculation, high speed analog computation, rapid automatic setup and checkout features, concurrent 2-variable function generation, and high speed logic operations available in the HYDAC 2400 system. The operational and cost saving features of the hybrid system add to its status as a powerful simulation tool.

#### REFERENCES

1. STALONY-DOBZANSKI, J., "Temperature Rate Flight Control System," Lecture given at University of California, Los Angeles, August 1963
2. STALONY-DOBZANSKI, J., "Application of Temperature Rate to Manual Flight Control of Re-entry Vehicles and Energy Management," Proceedings National Aerospace Electronics Conference, Dayton, Ohio, 1962
3. CHAPMAN, DEAN R., "An Analysis of the Corridor and Guidance Requirements for Supercircular Entry Into Planetary Atmosphere," NASA TR R-55, 1959.
4. FREDERICKSON, A. A., "Analog Computer Mechanization for Guidance Law Studies," The Boeing Company, Report #D2-8117, November 1960
5. LEES, LESTER, HARTWIG, F. W., and COHEN, C. B., "Use of Aerodynamic Lift During Entry Into the Earth's Atmosphere," Jet Propulsion, Vol. 29, No. 9, September 1959, p. 633
6. WISNESKI, M. L. "An Analog Computer Simulation X-20 Glide Phase Guidance Studies," The Boeing Company, Report #D2-90234, August 1962



# REVIEWERS, PANELISTS, AND SESSION CHAIRMEN

*AFIPS and the 1964 Fall Joint Computer Conference Committee would like to express their sincere appreciation to those listed below for their contribution toward the formulation and execution of the technical program.*

## SESSION CHAIRMEN

D. A. BAUMANN  
R. BELLUARDO  
L. C. CLAPP  
H. E. EDEN  
D. C. EVANS

H. G. KOLSKY  
G. S. MITCHELL  
D. B. PARKER  
W. J. QUIRK  
H. M. TEAGER

J. H. TURNOCK, JR.  
R. VICHNEVETSKY  
P. R. WINTERS

## REFEREES

C. M. ABLow  
G. N. ARNOVICK  
P. B. BAXENDALE  
R. W. BORNEMAN  
J. R. BROWN, JR.  
C. H. BURNS  
S. G. CAMPBELL  
W. C. CARTER  
T. E. CHEATHAM, JR.  
J. G. CLARK  
A. B. CLYMER  
D. J. CRAWFORD  
G. J. CULLER  
W. J. DIXON  
A. R. EAGLE  
T. S. EASON  
M. EDEN  
B. ELSPAS  
F. ENGEL, JR.  
O. FIRSCHEIN  
L. E. FOGARTY  
A. R. FURMAN  
E. O. GILBERT  
E. G. GILBERT  
R. C. GOLD  
J. GOLDBERG  
M. GORFINKEL

J. GRIFFITH  
R. S. GRISETTI  
N. HARDY  
C. HENSLEY  
J. R. HERNDON  
G. HILL  
W. J. HOLLIS  
R. M. HOWE  
C. C. HURD  
E. M. KING, JR.  
G. A. KORN  
J. P. LAZARUS  
R. M. LEE  
W. W. LICHTENBERGER  
L. B. LUSTED  
T. MARILL  
C. H. MAYS  
M. E. MCCOY  
J. MERSEL  
R. C. MINNICK  
M. MONTALBANO  
J. J. MURPHY  
E. NAGLE  
I. D. NEHAMA  
J. NELSON  
B. G. OLDFIELD  
W. M. OVERN

A. M. PIETRASANTA  
E. M. PIPER  
N. E. POBANZ  
J. H. POMERENE  
L. C. RAY  
L. G. ROBERTS  
A. I. RUBIN  
J. A. G. RUSSELL  
L. A. RUSSELL  
T. R. SAVAGE  
O. H. SCHMITT  
R. SILVER  
C. H. SINGLE  
R. C. SINGLETON  
W. R. SMITH  
R. SPINRAD  
R. STACY  
T. G. STOCKHAM, JR.  
J. F. UNDERWOOD  
R. VICHNEVETSKY  
I. A. WARHEIT  
H. R. WARNER  
R. R. WHEELER  
W. E. WIEBENSON, JR.  
J. W. YOUNG, JR.

## PANELISTS

G. A. BEKEY  
G. J. CULLER  
E. G. FUBINI  
J. T. GILMORE, JR.  
R. C. GOLD  
R. M. HOWE

E. M. KING, JR.  
G. A. KORN  
R. LORD  
J. MCCARTHY  
M. E. MCCOY  
T. J. MOFFETT

A. G. OETTINGER  
G. A. PAQUETTE  
D. T. ROSS  
D. SINNOTT  
B. WIDROW



# AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES (AFIPS)

211 E. 43rd Street, New York 17, New York

## *Officers and Board of Directors of AFIPS*

### *Chairman*

DR. EDWIN L. HARDER\*  
Westinghouse Electric Corp.  
700 Braddock Avenue  
East Pittsburgh, Pa.

### *Treasurer*

FRANK E. HEART\*  
Lincoln Laboratory  
P. O. Box 73  
Lexington 73, Mass.

### *Secretary*

SAUL T. GASS  
IBM Corporation  
1800 Yosemite Road  
Berkeley 7, Calif.

### *Chairman-Elect*

DR. EDWIN L. HARDER  
Westinghouse Electric Corporation  
700 Braddock Avenue  
East Pittsburgh, Pa.

### *ACM Directors*

H. S. BRIGHT  
Philco Corporation  
Computer Division  
5900 Walsh Road  
Willow Grove, Pa.

HOWARD BROMBERG\*  
2024 Addison Street  
Philadelphia 46, Pa.

EUGENE H. JACOBS  
System Development Corporation  
2500 Colorado Avenue  
Santa Monica, Calif.

DR. WALTER BAUER  
Informatics, Inc.  
15300 Ventura Boulevard  
Sherman Oaks, Calif.

### *IEEE Directors*

WALTER L. ANDERSON\*  
General Kinetics, Inc.  
2611 Shirlington Road  
Arlington 6, Va.

G. L. Hollander  
Hollander Associates  
P. O. Box 2276  
Fullerton, Calif.

DR. ARNOLD A. COHEN  
Univac  
St. Paul 16, Minn.

CLAUDE A. R. KAGAN  
Western Electric Co.  
P. O. Box 900  
Princeton, N. J.

### *Simulation Council Director*

JOHN E. SHARMEN  
Simulation Councils, Inc.  
P. O. Box 504  
Sunnyvale, Calif.

### *Executive Secretary*

REDMOND GARDNER  
AFIPS  
211 E. 43rd St.  
New York 17, N. Y.

### *American Documentation Institute Director*

PETER LUHN  
1000 Westchester Ave.  
White Plains, N. Y.

\* Executive Committee

*Standing Committee Chairmen*

*Admissions*

BRUCE GILCHRIST  
Service Bureau Corporation  
424 Park Avenue  
New York 22, N. Y.

*Award*

SAMUEL LEVINE  
Teleregister Corporation  
445 Fairfield Avenue  
Stanford, Conn.

*Conference*

KEITH UNCAPHER  
The RAND Corporation  
1700 Main Street  
Santa Monica, Calif.

*Constitution and By-Laws*

Eugene Jacobs  
System Development Corporation  
2500 Colorado Avenue  
Santa Monica, Calif.

*Finance*

WILLIAM D. ROWE  
Sylvania Electronics Systems  
189 B. Street  
Needham Heights, Mass.

*International Relations*

PROF. JOHN PASTA  
Digital Computer Lab  
University of Illinois  
Urbana, Illinois

*Planning*

JACK MOSHMAN  
C.E. I.R.  
One Farragut Square  
Washington, D. C.

*Publication*

STANLEY ROGERS  
General Dynamics  
Astronautic Division  
P. O. Box 166  
San Diego 12, Calif.

*Public Relations*

ISAAC SELIGSOHN  
IBM Corporation  
Dept. 1-812  
590 Madison Avenue  
New York 22, N. Y.

*Social Implications of Information  
Processing Technology*

DR. MORRIS RUBINOFF  
Moore School of Electrical Engineering  
University of Pennsylvania  
Philadelphia 4, Pa.

*Education*

RICHARD M. BROWN  
University of Illinois  
Coordinated Science Lab  
Urbana, Illinois

*1964 FJCC Chairman*

DR. RICHARD TANAKA  
Lockheed Missiles and Space Co.  
3851 Hanover Street  
52-40 Bldg. 201  
Palo Alto, Calif.

# 1964 FALL JOINT COMPUTER CONFERENCE COMMITTEE

*Chairman:*

RICHARD I. TANAKA, Lockheed Missiles & Space Company

*Vice Chairman:*

ARTHUR J. CRITCHLOW, IBM Corporation

*Secretary-Administrator:*

R. GEORGE GLASSER, McKinsey & Company, Inc.

*Treasurer:*

ED C. DODGE, Computer Sciences Corporation

*Technical Program:*

DAVID R. BROWN, Stanford Research Institute, Chairman

SAUL I. GASS, IBM Corporation

HAROLD E. PETERSEN, IBM Corporation

PROF. DANIEL TEICHROEW, Case Institute of Technology

*Exhibits:*

ARTHUR SCHOLAR, Pacific Telephone & Telegraph Co., Chairman

RON R. BANDUCCI, Pacific Telephone & Telegraph Co., Vice Chairman

*Local Arrangements:*

RAYMOND D. SMITH, SCM Corporation, Chairman

THOMAS C. BIEG, IBM Corporation, Vice Chairman

OSCAR M. PALOS, Bank of America

ALBERT C. PORTER, California Public Utilities Commission

*Registration:*

ED C. DODGE, Computer Sciences Corporation, Chairman

THOMAS R. DINES, Control Data Corporation, Vice Chairman

JOSEPH R. PARKER, The Service Bureau Corporation

*Printing and Mailing:*

CHESTER A. CREIDER, JR., Philco Corporation, Chairman

*Publications:*

WILLIAM H. DAVIDOW, General Electric Computer Laboratory, Chairman

*Public Relations:*

WALT KEESHEN, JR., IBM Corporation, Chairman

JOHN B. HATCH, Ampex Corporation, Vice Chairman

*Ladies' Activities:*

MRS. MARY LOU HOWEY, IBM Corporation

MRS. DONNA PROCTOR, IBM Corporation

MISS DANNY SIROSKEY, IBM Corporation

*Education Program:*

ROBERT J. ANDREWS, IBM Corporation, Chairman

NORTON O. SALBERG, IBM Corporation, Vice Chairman

HARRY ARNOLD, Control Data Corporation

MRS. KATHERINE WHITE, Physics International Co.

*Consultants:*

JOHN L. WHITLOCK, J. L. Whitlock Associates, Exhibits

WILLIAM C. ESTLER, Public Relations





# EXHIBITORS

## 1964 FALL JOINT COMPUTER CONFERENCE

- ADAGE, INC., Cambridge, Mass.  
AERONUTRONIC, Division of Philco Corp.,  
Newport Beach, Calif.  
AMERICAN TELEPHONE & TELEGRAPH  
COMPANY, New York, N.Y.  
AMPEX CORP., Redwood City, Calif.  
ANELEX CORPORATION, Boston, Mass.  
APPLIED DYNAMICS, INC., Ann Arbor, Mich.  
AULT, INC., Minneapolis, Minn.  
BECKMAN INSTRUMENTS, INC., Computer  
Operations, Fullerton, Calif.  
BENSON-LEHNER CORPORATION, Van  
Nuys, Calif.  
BRUSH INSTRUMENTS, Cleveland, Ohio  
BRYANT COMPUTER PRODUCTS, Walled  
Lake, Mich.  
CALIFORNIA COMPUTER PRODUCTS, INC.,  
Anaheim, Calif.  
THE CALMA COMPANY, Los Gatos, Calif.  
KENNEDY COMPANY  
TEL-A-DEX CORPORATION  
CAMBRIDGE COMMUNICATIONS CORPO-  
RATION, Cambridge, Mass.  
C-E-I-R, INC., Washington, D. C.  
COLLINS RADIO COMPANY, Dallas, Texas  
COMCOR, INC., Subsidiary of Astrodata, Inc.,  
Anaheim, Calif.  
COMPUTER CONTROL COMPANY, INC.,  
Framingham, Mass.  
COMPUTER DESIGN, West Concord, Mass.  
COMPUTER SCIENCES CORPORATION, El  
Segundo, Calif.  
COMPUTERS AND DATA PROCESSING &  
DATA SYSTEMS DESIGN, New York, N.Y.  
CONDUCTRON CORPORATION, Ann Arbor,  
Mich.  
CONTROL DATA CORPORATION, Minneapo-  
lis, Minn.  
CYBETRONICS, INC., Waltham, Mass.  
DATA DISC, INC., Palo Alto, Calif.  
DATAMATION, New York, N.Y.  
DATAMEC CORPORATION, Mountain View,  
Calif.  
DATA PROCESSING DIGEST, INC., Los  
Angeles, Calif.  
DATA SYSTEMS, INC., Subsidiary of Union  
Carbide Corp., New York, N.Y.  
DI/AN CONTROLS, INC., Boston, Mass.  
DIGITAL EQUIPMENT CORPORATION, May-  
nard, Mass.  
DIGITRONICS CORPORATION, Albertson,  
L.I., N.Y.  
DYMEC, Division of Hewlett-Packard Com-  
pany, Palo Alto, Calif.  
ELCO CORPORATION, Willow Grove, Pa.  
ELECTRONIC ASSOCIATES, INC., West Long  
Branch, N. J.  
ELECTRONIC MEMORIES, INC., Hawthorne,  
Calif.  
ENGINEERED ELECTRONICS COMPANY,  
Santa Ana, Calif.  
FABRI-TEX, INC., Amery, Wisc.  
FERROXCUBE CORPORATION OF AMER-  
ICA, Saugerties, N.Y.  
FRIDEN, INC., San Leandro, Calif.  
FUJITSU, LYD., c/oThe Nissho Pacific Corpora-  
tion, San Francisco, Calif.  
GENERAL ELECTRIC, Computer Depart-  
ment, Phoenix, Ariz.  
GENERAL KINETICS, INC., Arlington, Va.  
IBM CORPORATION, White Plains, N.Y.  
INDIANA GENERAL CORPORATION, Chi-  
cago, Ill.  
INFORMATION INTERNATIONAL, INC.,  
Cambridge, Mass.  
INVAC CORPORATION, Waltham, Mass.  
KLEINSCHMIDT, Division of SCM Corpora-  
tion, Deerfield, Ill.  
LIBRASCOPE GROUP, GENERAL PRECI-  
SION, INC., Glendale, Calif.  
LITTON INDUSTRIES, Beverly Hills, Calif.  
McGRAW-HILL BOOK COMPANY, New York,  
N.Y.  
MEMOREX CORPORATION, Santa Clara,  
Calif.  
MICRO SWITCH, Division of Honeywell, Inc.,  
Freeport, Ill.  
MIDWESTERN INSTRUMENTS, Tulsa, Okla.  
MILGO ELECTRONIC CORP., Miami, Fla.  
THE NATIONAL CASH REGISTER COM-  
PANY, Dayton, Ohio  
NORTH AMERICAN AVIATION, INC., El  
Segundo, Calif.  
OMNITRONICS, INC., Philadelphia, Pa.  
POTTER INSTRUMENT COMPANY, INC.,  
Plainview, L.I., N.Y.  
PRENTICE-HALL, INC., Englewood Cliffs,  
N. J.  
RADIATION, INC., Products Division, Mel-  
bourne, Fla.

RAYTHEON COMPUTER, Formerly Packard  
Bell Computer, Santa Ana, Calif.  
RCA ELECTRONIC COMPONENTS & DE-  
VICES, Harrison, N. J.  
RCA ELECTRONIC DATA PROCESSING,  
Cherry Hill, N. J.  
RECORDAK CORPORATION, New York, N.Y.  
RHEEM ELECTRONICS, Hawthorne, Calif.  
ROTRON MANUFACTURING COMPANY,  
INC., Woodstock, N.Y.  
ROYAL McBEE CORPORATION, New York,  
N.Y.  
SCIENTIFIC DATA SYSTEMS, INC., Santa  
Monica, Calif.  
SCM CORPORATION, New York, N.Y.  
SOROBAN ENGINEERING, INC., Melbourne,  
Fla.  
SPARTAN BOOKS, INC., Washington, D. C.  
STRAZA INDUSTRIES, Las Vegas, Nev.

STROMBERG CARLSON, Division of General  
Dynamics, San Diego, Calif.  
SYSTRON-DONNER CORPORATION, Concord  
Calif.  
TALLY CORPORATION, Seattle, Wash.  
TELEMETRICS, INC., Gardena, Calif.  
TELETYPE CORPORATION, Skokie, Ill.  
TEXAS INSTRUMENTS, INC., Houston, Texas  
TEXAS INSTRUMENTS, INC., Dallas, Texas  
TRANSISTOR ELECTRONICS CORP., Min-  
neapolis, Minn.  
UGC INSTRUMENTS, Division of United Gas  
Corporation, Shreveport, La.  
VICTOREEN INSTRUMENT COMPANY,  
Cleveland, Ohio  
JOHN WILEY & SONS, INC., New York, N.Y.  
WYLE LABORATORIES, Products Division,  
El Segundo, Calif.  
ZELTEX, INC., Concord, Calif.

# LIST OF JOINT COMPUTER CONFERENCES

1. 1951 Joint AIEE-IRE Computer Conference, Philadelphia, December 1951
2. 1952 Joint AIEE-IRE-ACM Computer Conference, New York, December 1952
3. 1953 Western Computer Conference, Los Angeles, February 1953
4. 1953 Eastern Joint Computer Conference, Washington, December 1953
5. 1954 Western Computer Conference, Los Angeles, February 1954
6. 1954 Eastern Joint Computer Conference, Philadelphia, December 1954
7. 1955 Western Joint Computer Conference, Los Angeles, March 1955
8. 1955 Eastern Joint Computer Conference, Boston, November 1955
9. 1956 Western Joint Computer Conference, San Francisco, February 1956
10. 1956 Eastern Joint Computer Conference, New York, December 1956
11. 1957 Western Joint Computer Conference, Los Angeles, February 1957
12. 1957 Eastern Joint Computer Conference, Washington, December 1957
13. 1958 Western Joint Computer Conference, Los Angeles, May 1958
14. 1958 Eastern Joint Computer Conference, Philadelphia, December 1958
15. 1959 Western Joint Computer Conference, San Francisco, March 1959
16. 1959 Eastern Joint Computer Conference, Boston, December 1959
17. 1960 Western Joint Computer Conference, San Francisco, May 1960
18. 1960 Eastern Joint Computer Conference, New York, December 1960
19. 1961 Western Joint Computer Conference, Los Angeles, May 1961
20. 1961 Eastern Joint Computer Conference, Washington, December 1961
21. 1962 Spring Joint Computer Conference, San Francisco, May 1962
22. 1962 Fall Joint Computer Conference, Philadelphia, December 1962
23. 1963 Spring Joint Computer Conference, Detroit, May 1963
24. 1963 Fall Joint Computer Conference, Las Vegas, November 1963
25. 1964 Spring Joint Computer Conference, Washington, April 1964
26. 1964 Fall Joint Computer Conference, San Francisco, October 1964.

Conferences 1 to 19 were sponsored by the National Joint Computer Committee, predecessor of AFIPS. Back copies of the proceedings of these conferences may be obtained, if available, from:

- Association for Computing Machinery, 211 E. 43rd St., New York 17, N. Y.
- Institute of Electrical and Electronic Engineers, Inc., Box A, Lennox Hills Station, New York, N. Y. 10021

Conferences 20 and up are sponsored by AFIPS. Copies of AFIPS Conference Proceedings may be ordered from the publishers as available at the prices indicated below. Members of societies affiliated with AFIPS may obtain copies at the special "Member Price" shown.

<i>Volume</i>	<i>List Price</i>	<i>Member Price</i>	<i>Publisher</i>
20	\$12.00	\$7.00	Macmillan Co., 60 Fifth Ave., New York 11, N. Y.
21	6.00	6.00	National Press, 850 Hansen Way, Palo Alto, Calif.
22	8.00	4.00	Spartan Books, Inc., 1106 Connecticut Avenue, N.W. Washington, D. C. 20036
23	10.00	5.00	Spartan Books, Inc.
24	16.50	8.25	Spartan Books, Inc.
25	16.00	8.00	Spartan Books, Inc.
26			Spartan Books, Inc.

## NOTICE TO LIBRARIANS

This volume (26) continues the Joint Computer Conference Proceedings (LC55-44701) as indicated in the above table. It is suggested that the series be filed under AFIPS and cross referenced as necessary to the Eastern, Western, Spring, and Fall Joint Computer Conferences.



# AUTHOR INDEX

- ALLEN, T. R., 387  
AMEMIYA, H., 123  
ANIZIENIS, A., 663  
BACKMAN, C. W., 411  
BAILEY, R. B., 717  
BARANY, J. T., 435  
BERGIN, G. P., 45  
BITTMANN, E. E., 93  
BOBRAW, D. G., 591  
BARETA, D., 1  
BRENNAN, R. D., 299  
BROWN, J. L., 205  
CAPOBIANCO, J. A., 81  
CARBREY, R. L., 707  
CARSON, D., 615  
CHANANNIS, T. E., 69  
COHLER, E. U., 175  
COLE, G. L., 363  
COLE, M. P., 351  
CUNNINGHAM, B. E., 423  
DAVIS, M. R., 325  
DAVIES, P. M., 147  
DICKINSON, M. M., 501  
DORN, P. H., 351  
DOODY, D. T., 205  
EDEN, M., 333  
ELLIS, T. O., 325  
EWING, R. G., 147  
FAGG, P., 205  
FAIRCLOUGH, J. W., 205  
FITZWATER, D. R., 465  
FOATE, J. E., 387, 397  
FOSS, E. D., 363  
FREDERICKSON, A. A., 717  
FREEMAN, D. N., 15  
FRICKE, L. H., 685  
GABOR, A., 435  
GALL, R. G., 159  
GARNER, E. G., 517  
GIMPELSON, L. A., 233  
GLUSKIN, R. S., 631  
GRAY, R. G., 363  
GREENE, J., 205  
HALPERN, M. I., 57  
HARGREAVES, B., 363  
HARNETT, R. T., 313  
HIPPI, J. A., 205  
HOWARD, J. A., 673  
JACKS, E. L., 343  
JACKSON, J. B., 501  
JACOBY, M., 631  
JOHNSON, E. L., 251  
JOYCE, J. D., 363  
KARPLUS, W. J., 673  
KINSLOW, H. A., 443  
KOPPEL, R. L., 81  
KRULL, R. L., 397  
LANZKRON, R. W., 489  
LEWIS, C. R., 351  
MCATEER, J. E., 81  
MACINTYRE, R. M., 69  
MAYHEW, T. R., 123  
MAYS, C. H., 623  
MERMELSTEIN, P., 333  
METZGER, L. G., 435  
MURATA, K., 187  
NAKAZAWA, K., 187  
NEWHALL, N. S., 481  
NUTTING, B. W., 527  
OCKER, W., 291  
OROZCO, E. G., 477  
ORR, W. K., 693  
OSEAS, J., 517  
PAQUETTE, G. A., 695  
PETERSEN, H. E., 313  
PICK, G. G., 107  
POUMAKIS, E., 435  
PRYOR, R. L., 123  
PYLE, W. I., 69  
RANDA, G. C., 501  
RAPHAEL, B., 577  
READER, T. D., 631  
ROBINSON, G., 615  
ROY, R. J., 527  
RUBINSTEIN, H. R., 175  
RUDIE, D. D., 251  
SAINT-PAUL, A., 717  
SANO, H., 299  
SANSOM, F. J., 313  
SCHWEPPE, E. J., 465  
SHARP, E. M., 363  
SHAW, J. C., 455  
SIPPEL, R. J., 363  
SPELLMAN, T. M., 363  
SPITZE, J. M., 643  
TALKIN, A. I., 539  
TEITILMAN, W., 559  
TEGER, S., 291  
THARPE, A., 363  
TRILLING, D. R., 277  
TOU, J. T., 651  
UHR, L. E., 35  
WALLI, C. R., 545  
WALSH, R. A., 685  
WARSHAWSKY, L. M., 313  
WEBER, J. A., 233  
WILLIAMS, S. B., 411  
WOS, L., 615  
YANG, C. C., 651  
YOUCHAH, M. I., 251