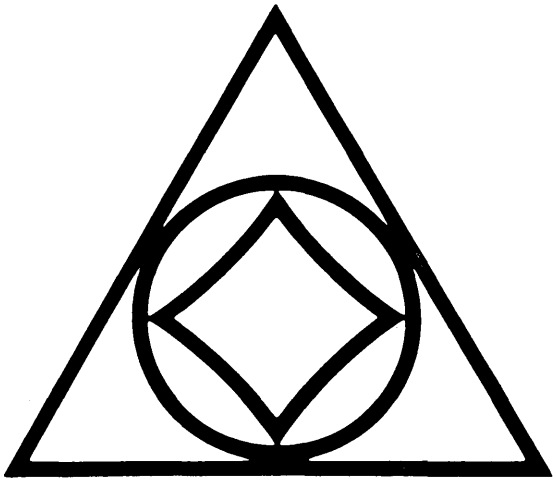


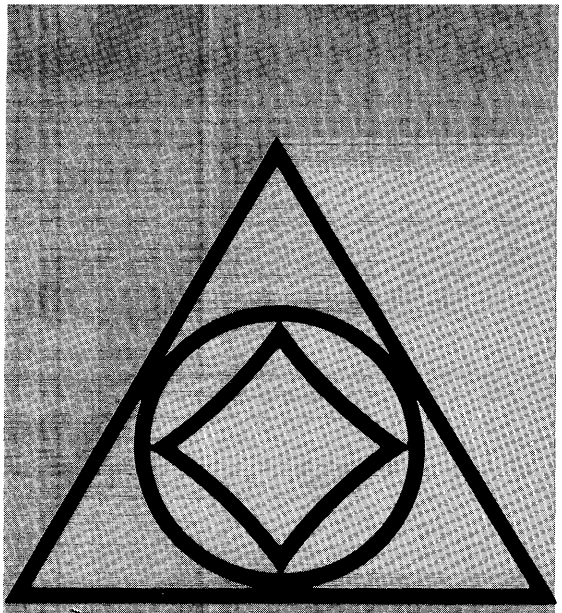
AFIPS

CONFERENCE
PROCEEDINGS

VOLUME 28

1966

SPRING JOINT
COMPUTER
CONFERENCE



AFIPS

CONFERENCE PROCEEDINGS

VOLUME 28

1966

SPRING JOINT COMPUTER CONFERENCE

1966
SPARTAN BOOKS
Washington, D. C.

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1966 Spring Joint Computer Conference Committee or the American Federation of Information Processing Societies.

Library of Congress Catalog Card Number 55-44701
Spartan Books, Div. of
Books, Inc.
1250 Connecticut Avenue, N. W.
Washington, D. C.

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

Macmillan Co., Ltd.
4 Little Essex Street
London W.C. 2

CONTENTS

COHERENT OPTICAL INFORMATION PROCESSING

TIME-SHARING

SIMULATION AND MODEL-BUILDING

PROCESSING LARGE FILES

WAVEFORM PROCESSING

PROGRAMMING LANGUAGES

BUSINESS APPLICATIONS

CURRENT DEVELOPMENTS IN PERIPHERAL HARDWARE

ANALOG/HYBRID TECHNIQUES

COMPUTER TECHNIQUES IN PATTERN RECOGNITION

COMPUTER APPLICATION OF ELECTRO-OPTICS

W. J. Poppelbaum
Department of Computer Science, University of Illinois

1. INTRODUCTION

The first few years of existence of electro-optics as a separate field have brought to light a wealth of novel ideas and has warmed up a host of old ones as well. It turns out that at this time there are two large classes of ideas which have relatively little overlap: the first class encompasses the development of time-honored techniques into practical low-cost designs; the second class contains the more futuristic devices using laser amplification, laser logic, laser source deflection, etc.[1-4] Unluckily the latter ideas have not yet reached that stage of development where they can compete with more classical designs as far as cost is concerned. Since the author has devoted a previous paper[5] to them, it might be useful to examine all those areas where patient development has produced reasonably low-cost designs that have proved themselves in practice or are about to do so.

The profusion of devices makes it mandatory to reduce the length of the list of subjects still further by eliminating arbitrarily the large field of techniques aiming at adaptive systems[6] or making use of nondissected forms of the incoming information.[7] This field is amongst the most exciting but, because of its links with biology, should perhaps be left to a separate discussion.

The ordering principle for the topics will be their occurrence in a highly symbolic system according to Fig. 1. It might astonish you to see storage and display in one functional block: the reason is that the majority of storage systems automatically give display and vice versa. It will also become apparent that there is a preponderance of subjects in this storage and display area and that further subdivision according to the method of access (electron beam, light beam, and static) is convenient. For simplicity's sake, areas, subdivisions and subjects are indicated on the figure.

2. INTERFACE LOGIC

The great contribution of electro-optics (or more precisely its all-optical branch) to information processing is probably the development of methods giving Fourier Transforms by optical means (see 2.1). Once this Fourier Transform is obtained, it is natural to operate in the frequency plane by appropriate masks in order to eliminate high- or low-frequency components or more generally to eliminate noise of given frequency distribution by matched filters. Such operations can be done by purely electronic means using matrix arrays of modulated elements (see 2.3). These techniques are usually applied to continuous input information (e.g., photographs), but there is no reason why the same methods should not be applied to discontinuous patterns of dots, leading to parallel processing of up to 10^6 bits of information.

Other developments in the area of interface logic are based upon the availability of glass fibers and the performance of OR and AND functions between the input medium and bundles of such fibers.

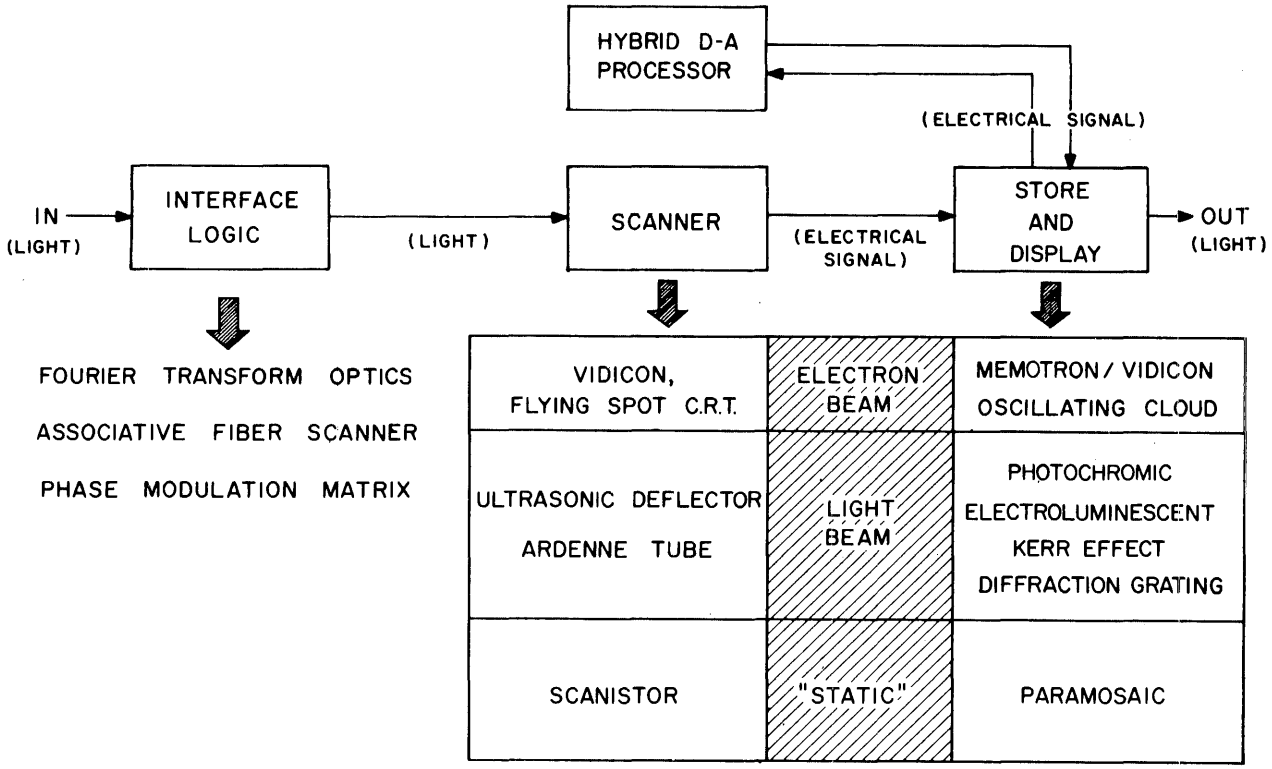


Figure 1. Practical electro-optical devices.

Encoding and decoding devices are particular examples of these techniques as is the associative fiber scanner described below (see 2.2).

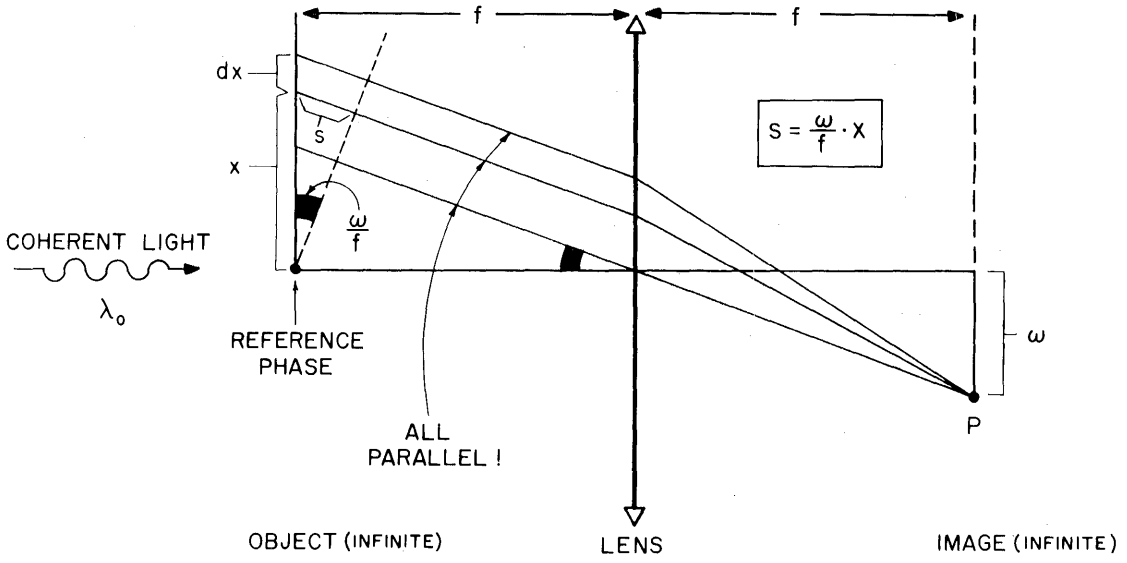
2.1 *Fourier Transform Optics*

The idea of using lenses to obtain the Fourier Transform of a planar density pattern has been discussed at length by its protagonists Leith,[8] Cutrona,[9] Parrent[10] and others. High-precision optics and strong coherent sources have made such a system highly practical and it is now entirely possible to process up to 10^6 bits of positional information in parallel, e.g., by spatial filtering (see below).

The fundamental principle of the (one-dimensional) optical Fourier Transform is shown in Fig. 2. The planes—or better, lines—of interest are the left and right focal planes: the left-hand one contains the object, the right-hand one what we shall here call the image *even if it is not the image of the object in the conjugate sense.* Let P be a point in the image plane which is ω below the optical axis: its illumination or amplitude is determined by the brightness and the phase of all beams coming from the object. Because P is in a focal plane, these beams are all parallel on the left side of the lens and form an angle of ω/f with the optical axis, where f is the focal length. The beam issuing from the section between x and $x + dx$ contributes an amplitude $a(x)\,dx \cos\frac{2\pi s}{\lambda 0}$ where s is the path delay with respect to some reference point—here chosen to be the intersection of the object plane with the optical axis. Geometry shows that s is a linear function of x: this is the crux of the matter. Assumption of an object and image of infinite extension and a change of notation (i.e., replacing cos by the real part of an exponential with an imaginary exponent) lead to the realization that the total amplitude in P (called illumination) is simply the real part of the Fourier Transform of $a(x)$. Of course it is well known that measuring devices—including our eye—actually see the intensity, i.e., the square of the amplitude. This leads to certain difficulties in practical systems, none of which, however, cannot be overcome.

2.2 *Associative Fiber Scanner*

The use of GaAs lamps and photodiodes together with fiber light conductors offers some rather interesting possibilities. As an example, consider the



LET a(x) = TRANSMISSIVITY IN x, THEN

$$\text{TOTAL ILLUMINATION IN } P = \int_{-\infty}^{+\infty} a(x)\,dx\,\cos\frac{2\pi s}{\lambda_0} = \int_{-\infty}^{+\infty} a(x)\,\cos\underbrace{\left(\frac{2\pi\omega}{\lambda_0 f}\right)}_{\text{call } \omega_x}\,dx = R\int_{-\infty}^{+\infty} a(x)\,e^{j\omega_x x}\,dx =$$

$$= R\left\{F\left[a(x)\right]\right\} = RA(\omega_X)$$

Figure 2. Fourier Transform optics (one-dimensional).

arrangement in Fig. 3. It is assumed that we have a film frame containing an $m \cdot n$ matrix of dark and light spots as well as the $m \cdot n$ matrix of the *negative* (i.e., the digit-wise complement). We want to build a device which can decide in *one* operation whether a given key word $(x_1 \ldots x_i \ldots x_n)$ is among the m words $(a_{\lambda 1} \ldots a_{\lambda i} \ldots a_{\lambda n})(\lambda = 1 \ldots m)$ in the frame, and if so, in what position.

The principle is to inject into the fibers going into the first digits the signal $\bar{x}_1$ via lamp L_1, etc., so that digits $a_{\lambda 1}$ are illuminated by $\bar{x}_i$. Similarly the digits $\bar{a}_{\lambda i}$ on the negative are illuminated by x_i. If we now collect—by a second set of fibers—the outputs of each row of photocells, the left-hand cell $\#\lambda$ will receive a signal $V_i a_{\lambda i} \bar{x}_i$ while the corresponding right-hand cell will receive $V_i \overline{a_{\lambda i}} x_i$. If we check the cells by pairs via OR circuits, only that OR circuit for which both inputs are 0 will correspond to $a_{\lambda i} = x_i$: its position gives the λ we search for. If no such OR circuit exists, the key word is not contained in the frame.

The system described above has been realized at the Department of Computer Science of the University of Illinois[11] in slightly modified form. It is presently being converted in such a fashion that the key word does not necessarily contain all the digits: provision is then made to read out the remaining digits upon coincidence. It will also be possible to change the key word (and the digits involved) as a function of the last word read out. This obviously will lead to a Turing-machine–like behavior of the system.

2.3 *Phase Modulation Matrix*

One of the astonishing photolithographic feats of recent times is Anderson's[12] production of varactor diodes having overall dimensions of the order of 10 microns. By suitable doping and reverse biasing, it is possible to make the depletion region behave like an optical transmission line, i.e., to use the regions of high carrier density as mirrors at a distance of a few thousand angstroms. It is not too hard to convince oneself that light going through the plane of the junction will suffer a phase change Φ determined by the thickness d of the depletion layer where d is

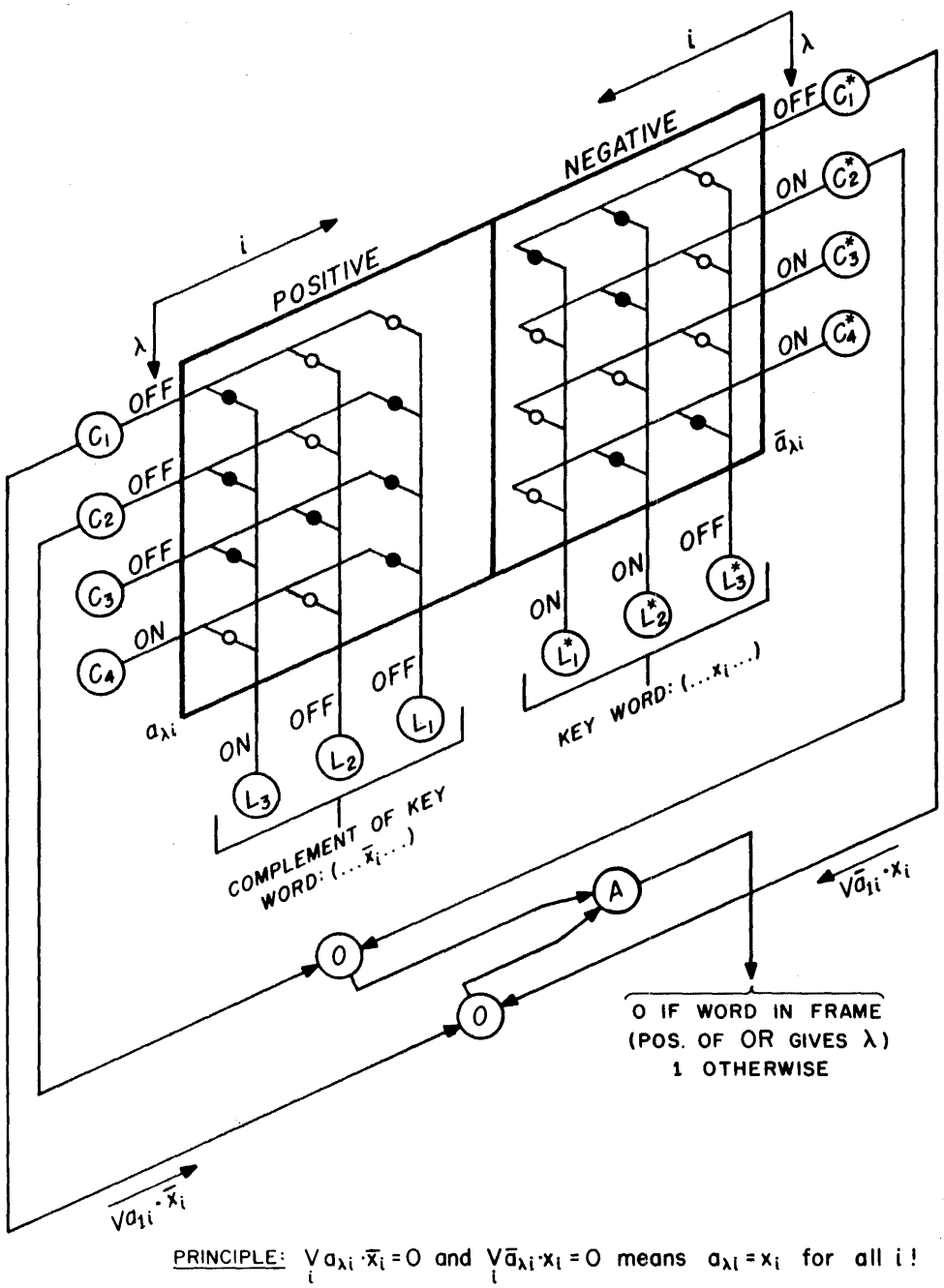


Figure 3. Associative fiber scanner.

approximately given by the Early-formula $d = \sqrt{3V\epsilon_r\epsilon_0}/2aq$, q being the charge of one electron and a the gradient of the donor density minus that of the acceptor density. Figure 4 contains the expression Φ as a function of d.

Instead of using the diodes as transmission optical phase shifters, it is also possible to expose one surface of the depletion layer by a transparent substrate and to shine the light into the device from the side. Such reflective optical phase shifters lend themselves more naturally to two-dimensional arrays, although it is not impossible to use transmission shifters in a matrix. A possible use of reverse biased diodes as storage elements makes such phase

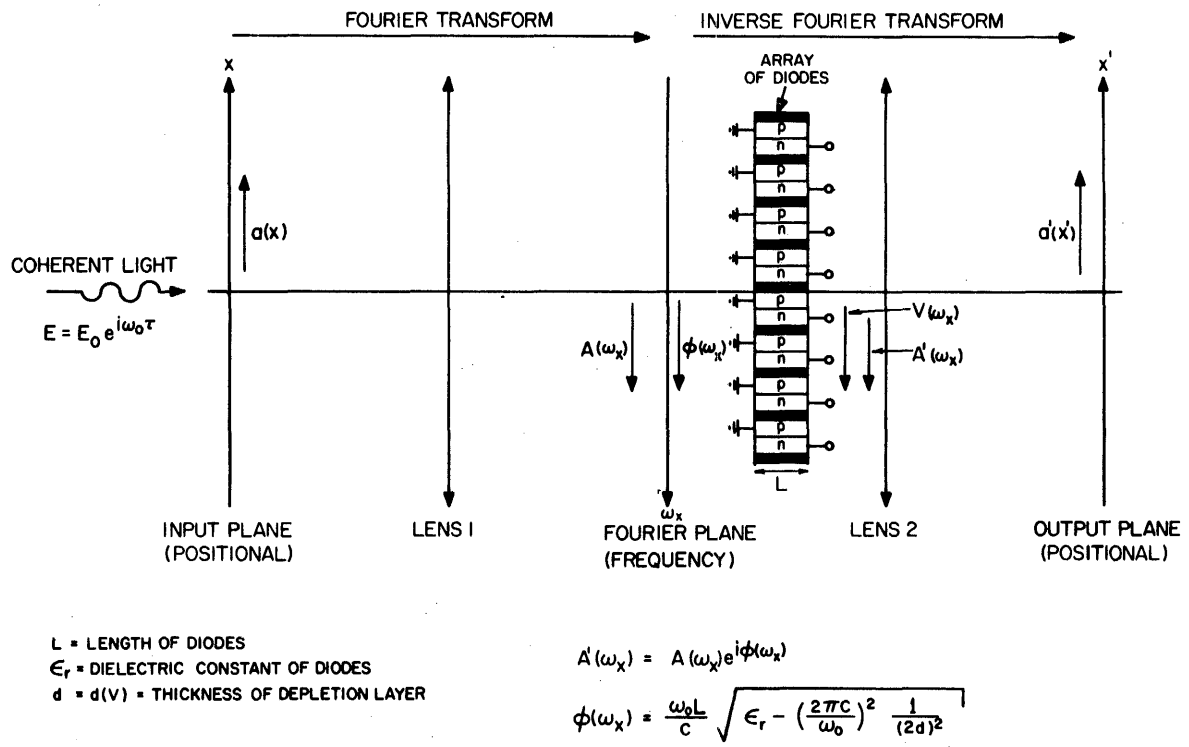


Figure 4. Phase modulation matrix used as a spatial filter.

modulation matrices especially attractive, since it is not necessary to connect a given diode to the input control voltage at all times: a charging mechanism using a scanner would be adequate.

Figure 4 shows a possible application of a phase modulation matrix in the Fourier Transform Plane: this is a form of spatial filtering. In the example the frequency-analyzed version of the input information in the ω_x-plane is subjected to phase delays in the array of diodes, the control voltage $V(\omega_x)$ being an impressed function of ω_x. If $A(\omega_x)$ is the amplitude in the ω_x-plane, spatial filtering will produce $A'(\omega_x) = A(\omega_x)e^{i\Phi(\omega_x)}$ and a reconstitution by means of an inverse Fourier Transform will lead to a filtered version of the input information. It is possible to convert the phase modulation into amplitude modulation by mixing the output with a reference phase as is done in holography.[13]

3. SCANNERS

The purpose of scanners is to convert incoming parallel pictorial information into serial information for subsequent transmission or processing: they are serializers. At the same time, using persistence of phosphors, the human eye, etc., they can be used to reconstitute the parallel information: they are also staticizers.

The scanning mechanism is usually an appropriately deflected beam of electrons or a beam of light. Neglecting the well-known examples of electron beam deflecting in flying spot analyzers (for slides) or vidicons (in which the optical information is converted to a charge distribution by a photocathode) our discussion will be limited to practical low-cost light deflectors. These are presently of the ultrasonic (see 3.1) and the KDP-CRT Type (see 3.2) if one neglects unfashionable mechanical devices like vibrating mirror galvanometers and rotating mirrors. This unluckily temporarily leaves out such exciting developments as those proposed by Pole[14] and Fleisher.[15]

A recent technique, using the Scanistor, is also in a stage where practical applications can be considered. This device is essentially static in nature and the scanning is done by applying appropriate voltages to the photosensitive element to be interrogated (see 3.3).

3.1 *Ultrasonic Deflector*

One of the most efficient systems for deflecting light through several degrees with an optical path inside the device of a few centimeters is the ultrasonic deflector shown schematically in Fig. 5. Here a quartz transducer injects an acoustical wave into

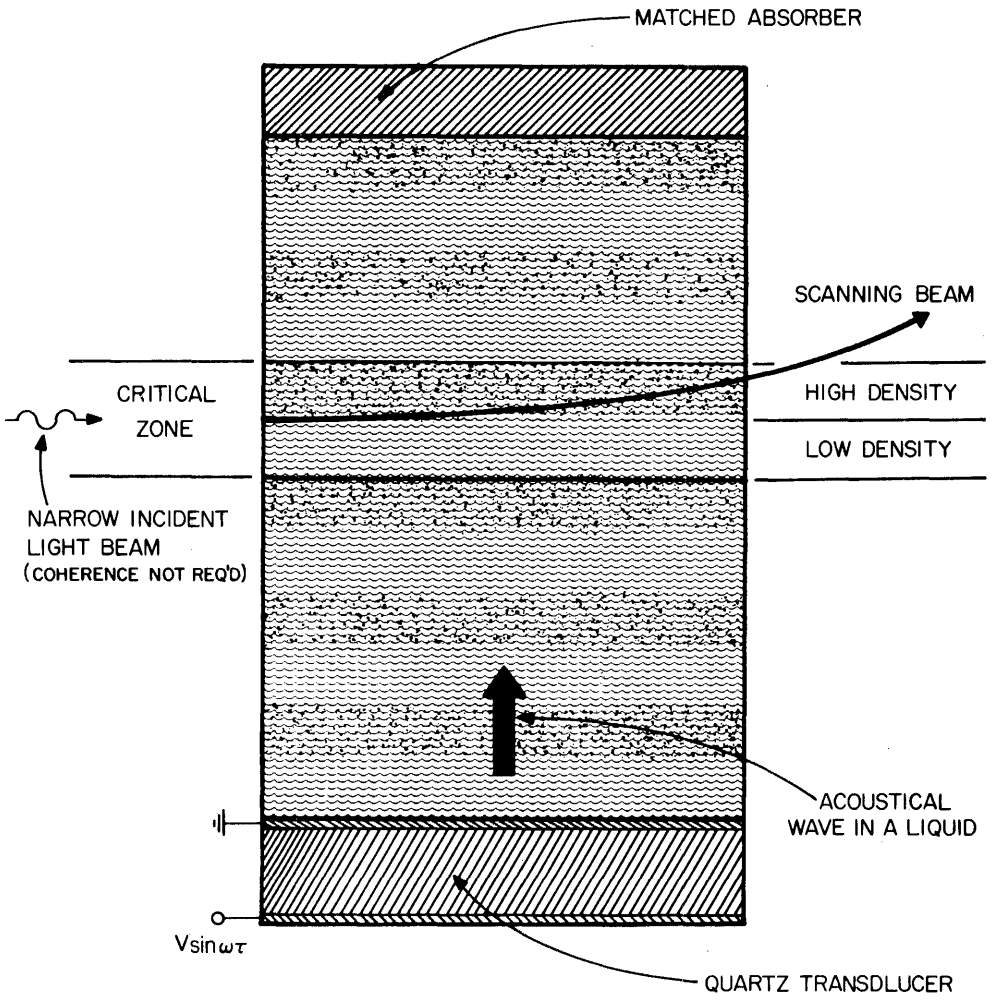


Figure 5. Ultrasonic deflector (Scophony system).

a compressible liquid at a frequency corresponding to the desired scanning action. The wave travels across a critical zone in which a (coherent or non-coherent) light beam is deflected in "Fata Morgana fashion" by the variable densities (and the ensuing varying index of refraction) generated by the wave. Upon reaching the far side of the device a matched absorber eliminates acoustical reflection and standing waves.

This system has been successfully used by Reich at Lockheed in the design of a photochromic semi-random access memory.[16] The liquid used was simply water and the operating frequency one megacycle, the source being a laser providing a beam narrow with respect to the acoustical wavelength. The beauty of the system is that two-dimensional scanning is possible by using a cell containing two acoustical waves traveling at right angles to each other. It is perhaps interesting to note that a quarter of a century ago the Scophony Television receivers in England used precisely the same method of light deflection!

3.2 *Ardenne Tube*

Another device saved from oblivion by Pole of IBM is the Ardenne Tube[17] shown in Fig. 6. Invented in 1934 for the German Postoffice for facsimile transmission, it is essentially a CRT with a special faceplate: a KDP crystal between two electrodes, one reflecting and the other transparent. When an electron beam hits the reflecting electrode, the negative charge (together with the induced positive charge in the transparent electrode) produces a strong electric field in the KDP. Due to the Electric Kerr Effect there will be a phase difference between ordinary and extraordinary components of light,

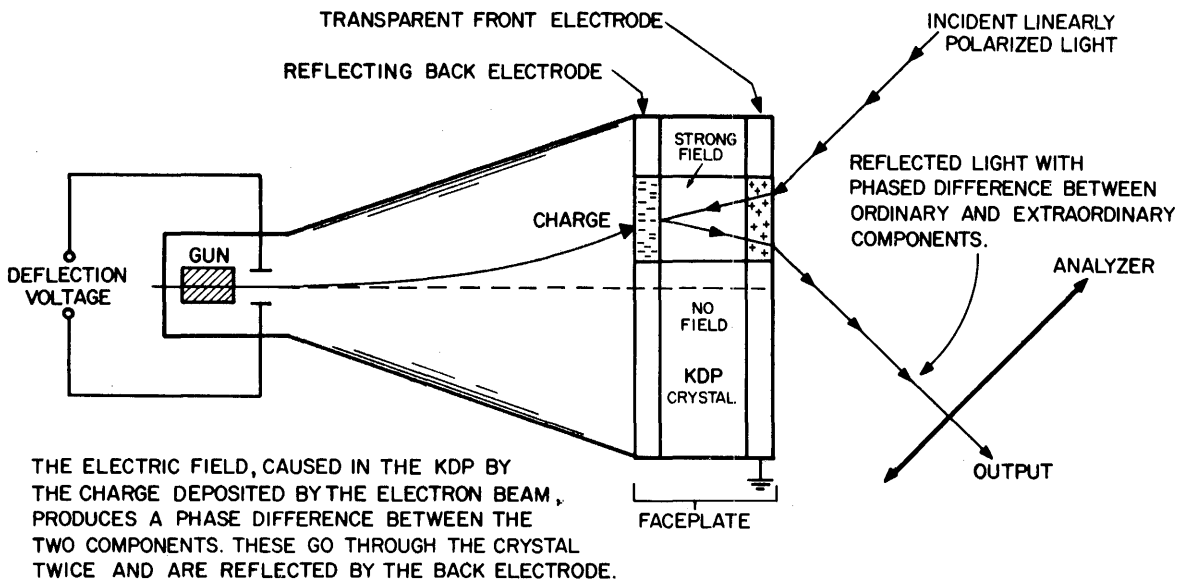


Figure 6. Ardenne tube.

this difference being proportional to the field. Light from a linearly polarized source in front of the faceplate traverses the crystal twice (being reflected by the back electrode) and finally goes through an analyzer. If the analyzer is appropriately positioned, it is possible to make the high field portions of the KDP "light up," the rest of the faceplate remaining dark. Clearly the position of the "light source" is solely controlled by the electron beam: we have a method for positioning a high luminosity source of light by controlling a CRT. Similar designs have been investigated by Pulvari[18] and Lindberg[19] of Motorola.

3.3 *Scanistor*

The basic idea of Horton's Scanistor[20] is to access an array of light-sensitive elements connected between a bus and a set of linearly increasing return voltages by applying to the bus an appropriate time-dependent voltage. The selection of any given element is actually performed by making the voltage across it nearly zero: the reason that this succeeds is that a back-to-back diode pair has always a big resistance as long as one or the other diode is strongly reverse biased. Zero total drop across the pair, however, puts both diodes into a region of relatively low impedance. If light is now received by one of the diodes (which therefore must be not only a diode but a photodiode) it starts acting like a current source, the source intensity being approximately proportional to the incoming light intensity.

Figure 7 shows an array of diode pairs illuminated by a step-function light distribution. The bus is used to sweep through the gamut of return voltages, i.e., $e(t)$ goes from 0 to some voltage E_0 within some scanning period T. For the first two diode pairs nothing happens, even when $e(t)$ reaches 0 and E_0/n. The third pair, however, suddenly passes a current through the upper bus as $e(t)$ reaches $2E_0/n$. Similarly the fourth pair produces a current through the bus as $e(t)$ reaches $3E_0/n$, etc. By using a transformer, each current increment produced during the sweep gives an output pulse to $v(t)$, the height being roughly proportional to the light intensity on the pair being accessed. It is easy to see that an appropriate integration (or even a running together of widened $v(t)$ pulses) produces an envelope which imitates the light intensity after mapping position into time: this is precisely what a scanner has to do.

It is important to note that the voltage divider chain between E_0 and ground can be built into the diode array. Semi-integrated forms of the circuit have had as many as 200 diode pairs per inch and have performed very satisfactorily. One of the main attractions is the very low power requirements of the device.

4. STORAGE AND DISPLAY

Storage devices must typically accept an optical input or the corresponding electric input in the form

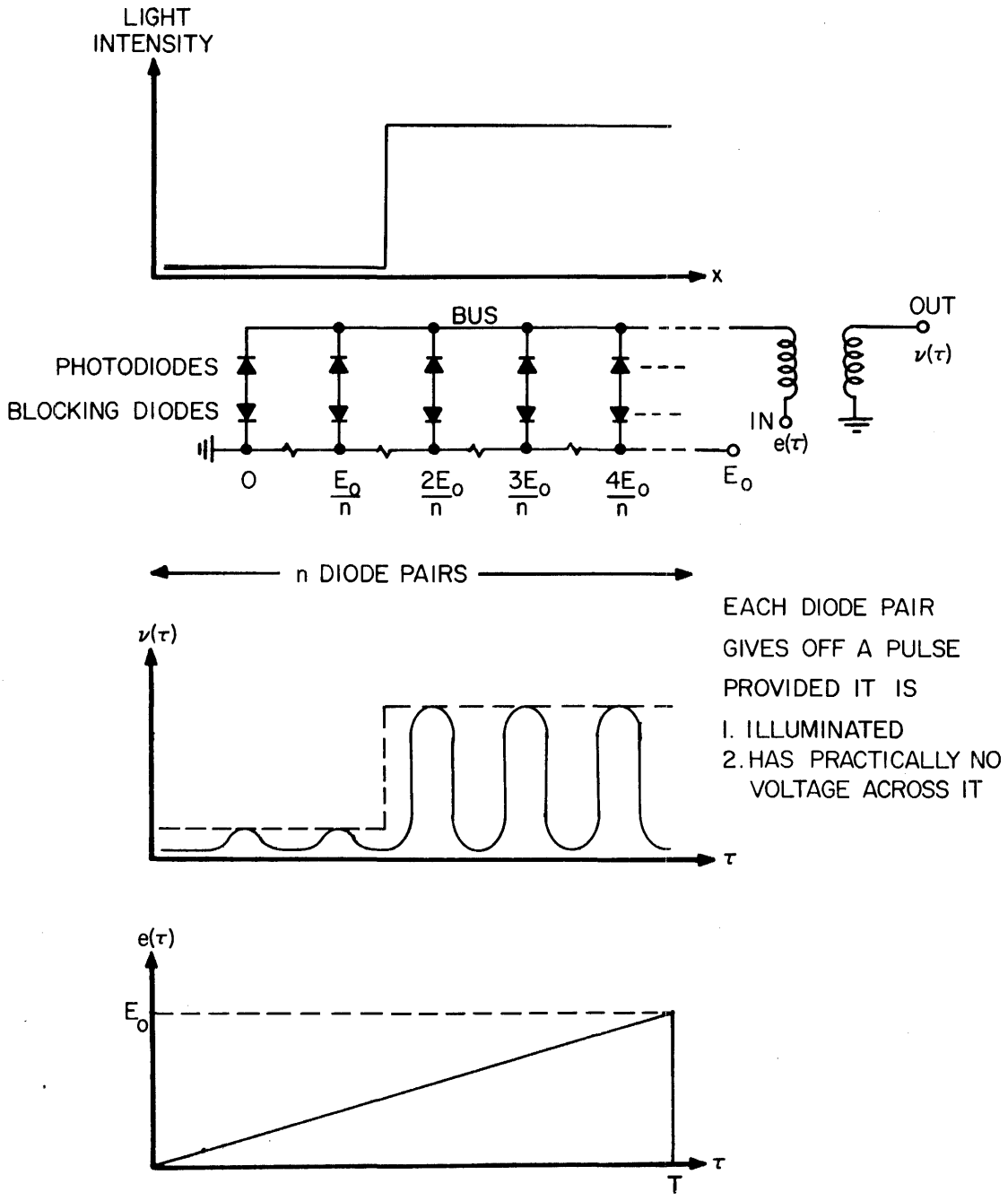


Figure 7. Scanistor principle.

of a time sequence of electrical signals generated by the electronic scanning of the picture. On demand they must repeat this time sequence after an arbitrarily long period. Excepting core storage of a digitized version of the time sequence (which will have to be quantized for this effect by subdividing scanning lines into a sequence of points!), it is interesting to note that all presently used devices perform the storage on a continuous or discontinuous matrix. It is this latter fact which makes such storage devices good candidates for display.

The continuous storage matrix may be as trivial as an old-fashioned memotron (i.e., a CRT with a flood-gun which, by selective secondary emission on the screen, keeps bright positions bright and dark positions dark) coupled to a vidicon to examine the

stored information. It can be made more sophisticated by essentially uniting all elements into a "Tonotron" (Hughes) or Scan Convertor, even if, at present, readout for both is somewhat destructive. It can finally take on the slightly esoteric form of an Oscillating Cloud Tube in which the information is a planar electron-density distribution being bounced back and forth between suitable electrodes (see 4.1). In all these versions the access is provided by an electron beam. It is quite possible to obtain storage on a continuous matrix using a light beam for access. A practical system developed by Lockheed uses a photochromic emulsion for storage and an ultrasonic scanner. A version with a mechanical scanner was designed by NCR. The most elegant continuous light-accessed array is, however, the electroluminescent panel (see 4.2) in which a CdS-film is used to store the information and an overlay of electroluminescent material as a light emitter controlled by the CdS-film.

Discontinuous storage matrices using light beams for access are presently of the thin magnetic film type: here spots are magnetized by a coincident current system in an underlying grid of wires. The state of magnetization can be read out by observing the angle through which the plane of polarization is rotated by the magnetic Faraday effect, using a plane-polarized scanning beam. Such work has led to angles of rotation of several degrees but is yet unpublished. Another, now well-developed, technique is based on the existence of striated domains and their observation by a diffraction grating effect (see 4.3).

There is, finally, a static storage and display design which is a sort of two-dimensional analog of the Scanistor. In this "Paramosaic" system a curve can be written into a matrix by making its grid-points equipotential and storing this fact on appropriate elements (see 4.4).

4.1 *Oscillating Cloud Tube*

In the Oscillating Cloud Tube[21] designed by Berg and Smith of Imperial College and shown in Fig. 8 we have three distinct sections. To the left is an orthicon-like structure in which the electrons given off by a photocathode are accelerated by a weak axial electric field to a rather low velocity. The identity of electron bunches originating from various parts of the photocathode is preserved by applying a strong axial magnetic field. The cloud now enters the storage section and is trapped between gate 1 and gate 2, the electric field applied via $V \sin \omega t$ bouncing them back and forth. When information is required from the store, gate 2 is opened as the cloud approaches it. The selected electrons are then accelerated into section 3 (iconoscope) which

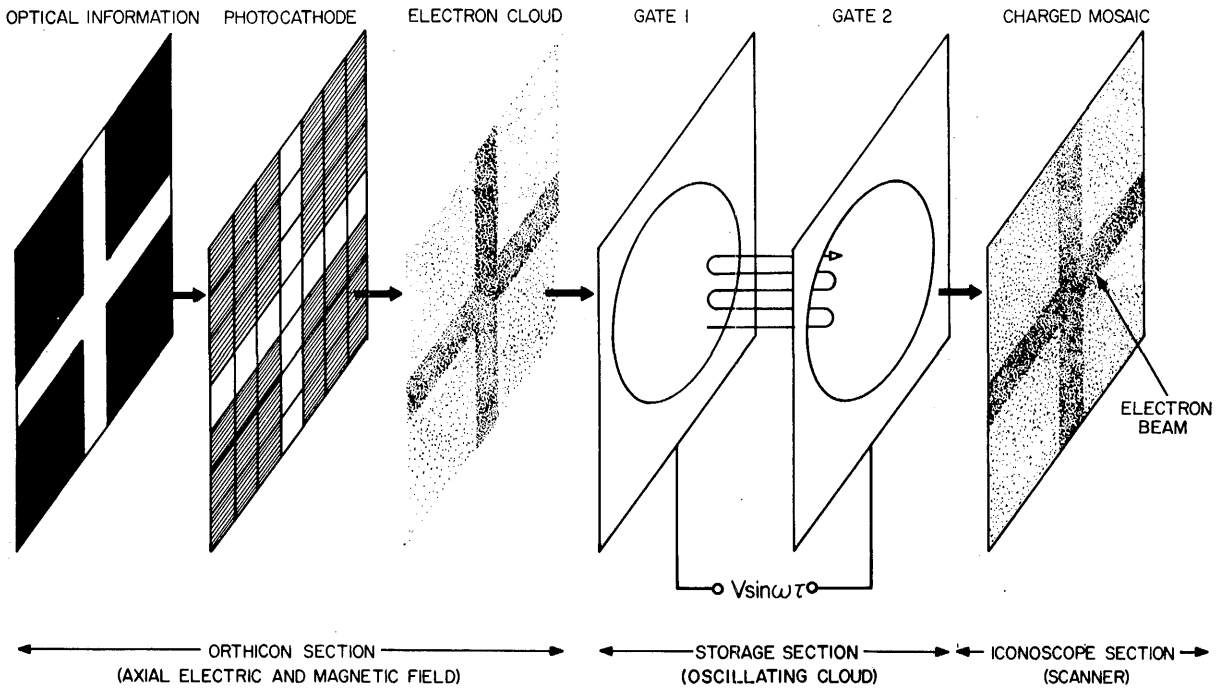


Figure 8. Oscillating cloud storage tube.

contains an intermediate mosaic in which the charges are stored for subsequent scanning by an electron beam.

The present resolution is in the vicinity of 100 lines. It seems entirely possible to store a sequence of distinct electron clouds, each corresponding to one frame of the input information. Selection of the appropriate cloud is then operated by timing the exit through gate 2. The access time to any information in the store is of the order of 200 nanoseconds.

4.2 *Electroluminescent Storage*

Electroluminescent Panels are ideal display devices although they still have a somewhat low light output. Recently panels have been made[22] which incorporate *storage facilities.*

Figure 9 shows the principles of operation. On the left is shown a *pn* diode reverse biassed by $-E$ applied to the *p*-region. Without light, the potential barrier $\Delta\psi$ across the junction is relatively high, its value being approximately given by

$$\Delta\psi = \frac{kT}{q} \ln \frac{np}{n_i^2} + E$$

where n and p are the electron and hole densities and n_i a constant. Let us now shine a light onto the *n*-region, generating hole-electron pairs. Some supplementary electrons neutralize a part of the donors and consequently lower the value of n (assuming space charge neutrality!). The formula above, although strictly valid only for equilibrium, shows that this amounts to a decrease of $\Delta\psi$ and therefore a stronger current flow. Other supplementary electrons are trapped: their slow release after the light disappears continues to neutralize donors, i.e., the current will remain strong. It is assumed that the current itself replenishes the traps, so that the low-impedance state of the diode is indefinitely conserved once light has struck it. Only the removal of E and the emptying of all traps (in darkness) will bring the diode back to its high-impedance state.

The storage and display combination is obtained by essentially putting a continuous sheet of *pn* diodes on top of an electroluminescent substance. Practically the diode action occurs in the barrier between a transparent electrode and an underlying CdS-film. An opaque layer separates the CdS-film and the electroluminescent material and a voltage is applied across the whole sandwich. In those spots

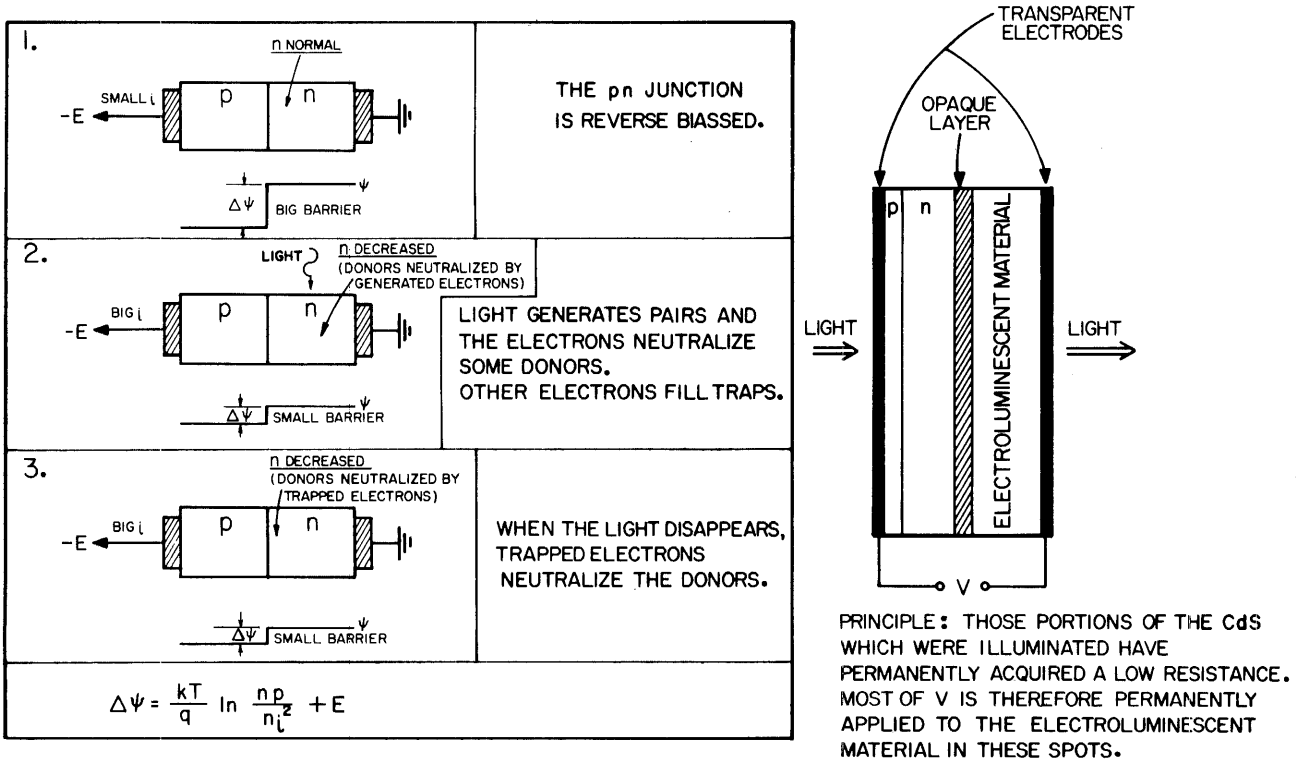


Figure 9. Electroluminescent storage.

where the series diode has been locked into its low impedance state by incident light, we shall have light emission from the highly polarized electroluminescent layer.

4.3 *Diffraction Grating Display*

Fuller of LFE[23] has produced a combined storage and readout system in which a thin magnetic film is magnetized in spots by a coincident current arrangement as shown in Fig. 10. The composition of the magnetic film is such that for one direction of magnetization the domains are long and narrow, the width of each being of the order of that of the wavelength of visible light and constant from one to the next.

A Bitter Solution (essentially iron filings in a suitable viscous liquid) is held between two transparent covers adjoining the magnetic film. As soon as the right magnetization occurs, the Bitter Solution will show a bunching effect and will take on the aspect of a periodic structure having all the properties of a diffraction grating. Incident light will therefore be diffracted into an observer's eye in the correctly magnetized spots, giving direct optical readout of the state of magnetization. The memory action obviously stems from the retention of this state even after the selecting currents have been taken away. Certain difficulties with diffused light and higher order diffractions can be attenuated by covering the device with a supplementary plate, producing total reflection for all but the desired direction of output light.

4.4 *Paramosaic*

A principle similar to that used in the Scanistor (but extended to two dimensions) was used at the Department of Computer Science of the University of Illinois[24] to produce storage and display in a matrix of wires with suitable elements thrown across

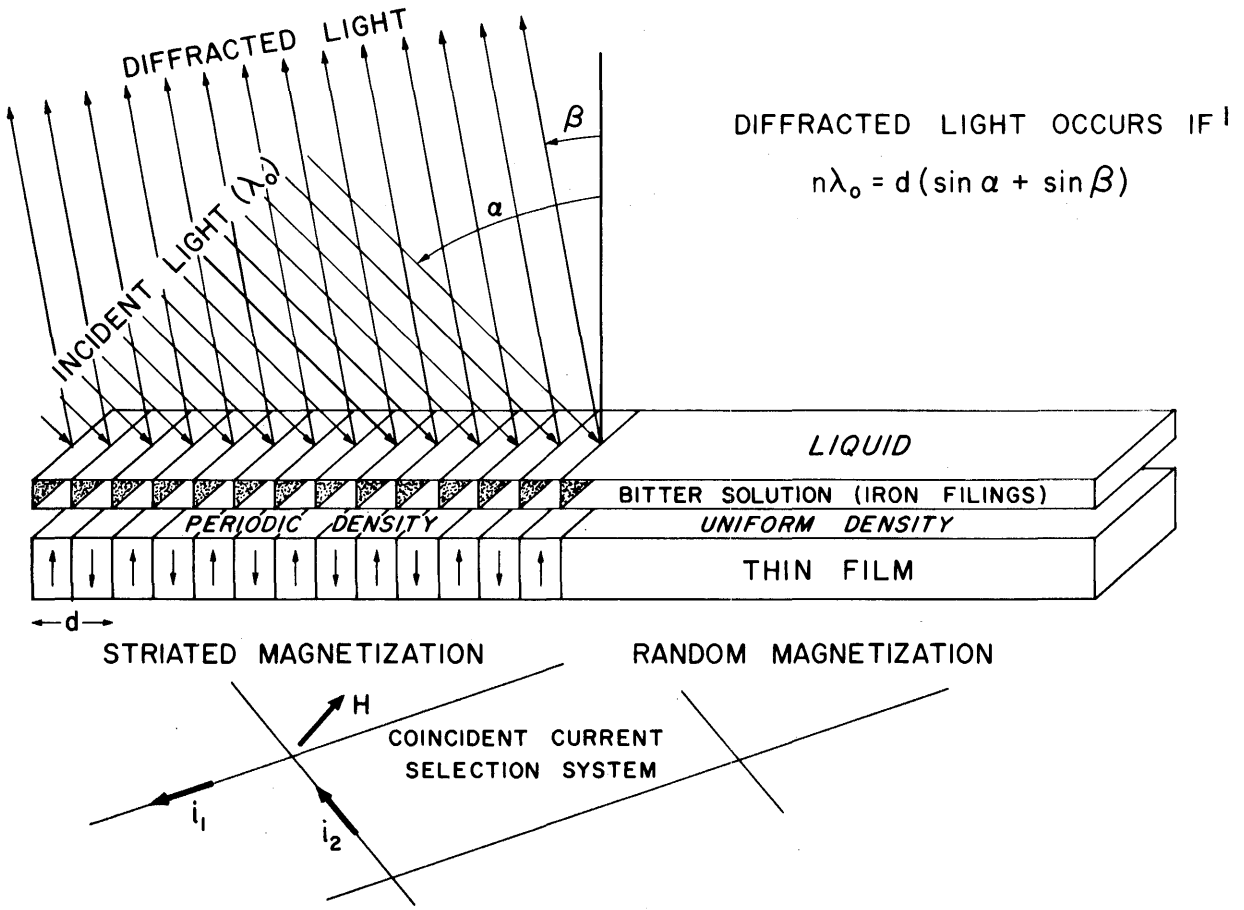


Figure 10. Diffraction grating display.

each grid-point (see Fig. 11). The practicality of such a "Paramosaic" hinges upon the cost of each display element. The one shown in the figure only gives temporary storage due to the inertia of the lamp and the actual model (32 × 32) used a more sophisticated design, consisting of a sensitive comparator and a flip-flop with readout via a tungsten light: if the grid-points were within 0.2v of each other, the light would come on permanently.

The interesting property of the system is that any single-valued function $y = f(x)$ can be displayed by giving a minimum amount of information, namely the voltage distribution $u(x) = kf(x)$ on the vertical wires. Making $v(y) = ky$ for the horizontal wires, only points with $y = f(x)$ (or nearly) will light up. Transmitting a sequence of "profiles" $u(x)$ can display line drawings with arbitrary complications and there is a considerable bandwidth reduction when the profiles are sent as time sequences.

5. GRAPHICAL PROCESSING AND COMPUTERS

Graphical processors always involve input and output devices of the type described above. Usually, however, the scanner is followed by a digitizer, and a general-purpose computer is used for processing and storage. The author feels that in *graphical processing*, many interesting operations can be performed at high speed and low cost by appealing to the latest designs in analog circuitry and by using storage principles of the kind discussed in Section 4. The reasoning is simply that the best scanners give resolutions of the order of 1000 lines

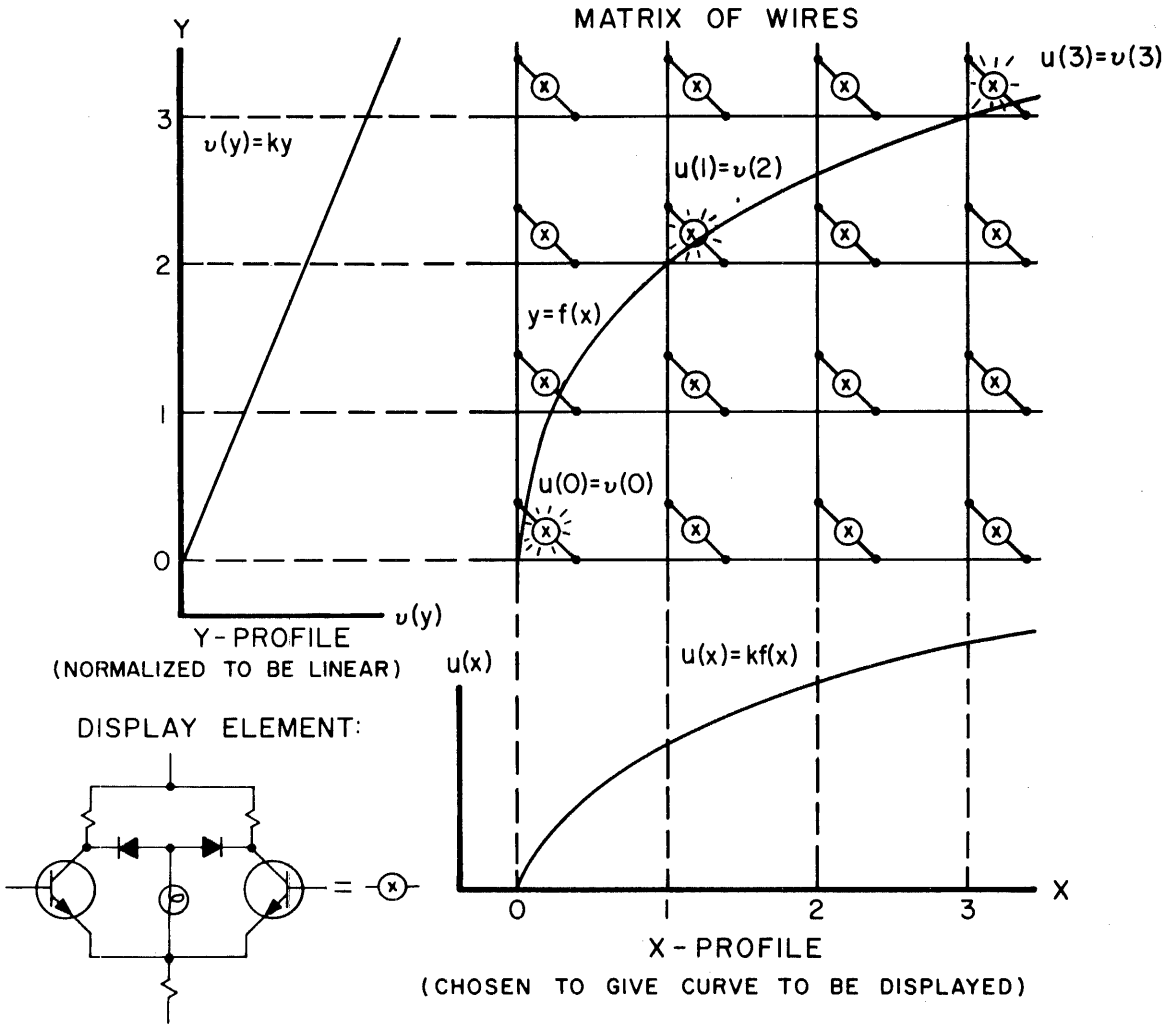


Figure 11. Paramosaic.

(10^6 bits per frame after quantization) and that *as long as the machine interacts with a human* a frame rate of a few frames per second is adequate: the data rate inside the processor is then of the order of a few megacycles per second.

The Department of Computer Science at the University of Illinois is presently engaged in proving these views in a system according to Fig. 12 used in automating the constructions of Euclidian geometry. In this Artrix System[25] use is made of hybrid digital-analog circuits with 0.3% precision from DC to 2 megacycles, the signal swings being from -10v to $+10$v. The three stores (PERMANENT, PAD and ACCUMULATOR) are all formed on memo-

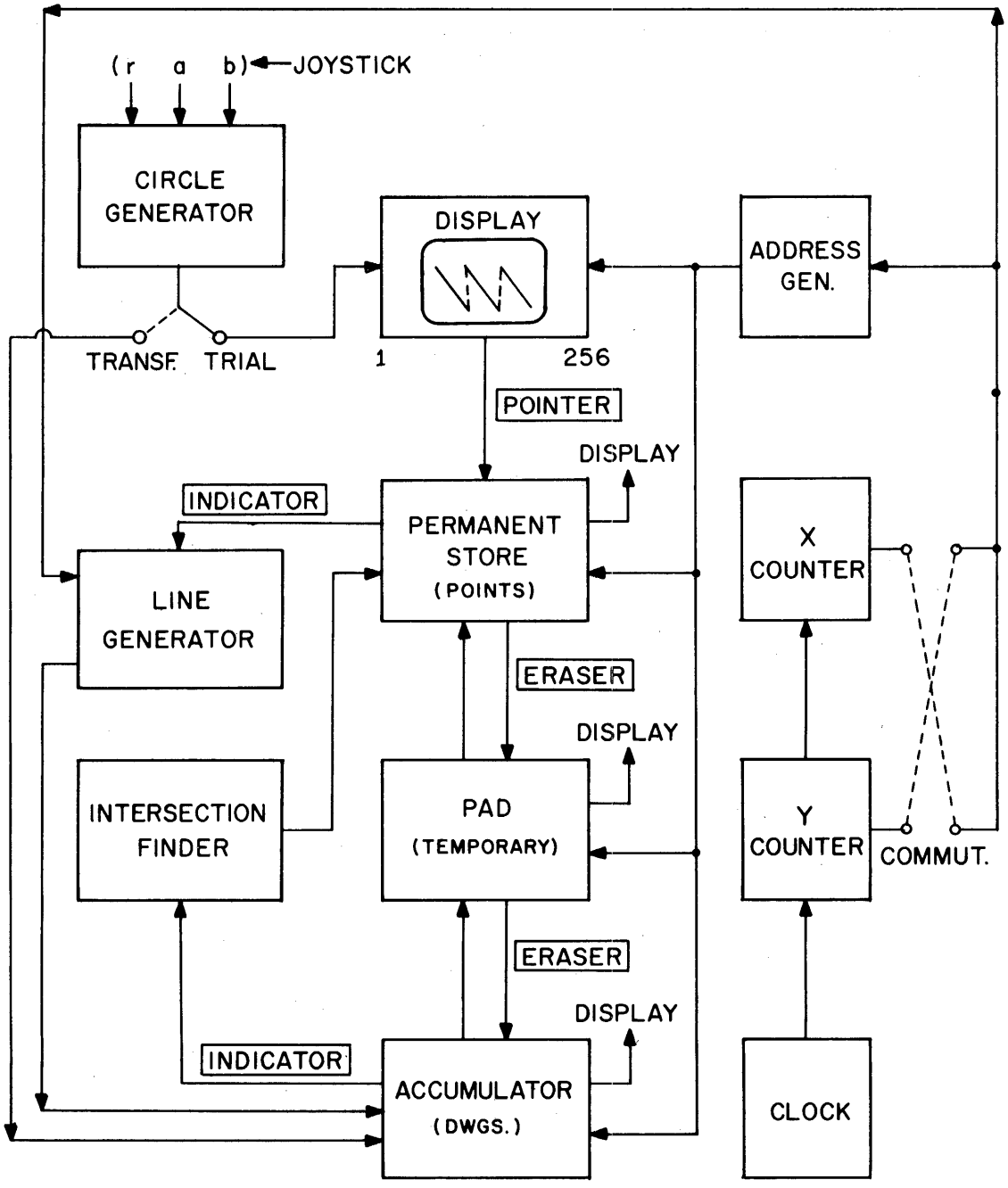


Figure 12. The Artrix system.

tron-vidicon pairs with a 300-line resolution: their contents are simultaneously sent to a display monitor which is also used for light-pen writing. A digital system positions all beams in synchronism and permits the transmission of digitized coordinates to the processing boxes (CIRCLE GENERATOR, LINE GENERATOR and INTERSECTION FINDER), thus obviating the storage of analog signals: the digital versions, after being converted, feed the analog processors.

Erasure is obtained by selective transfer of information from the permanent store or the accumulator to the PAD: the portions not to be transferred are pointed at with a large aperture "ERASER" light-pen. The "POINTER" light-pen is used to set the initial construction points into the permanent store: it has a small aperture. The "INDICATOR" light-pen has again a large aperture and generates a gating signal which allows the system to look up the exact value of coordinates in the permanent store. The two fundamental operations are the automatic drawing of straight lines through two points and the construction of circles with a given center (a,b) and a given radius r.

ACKNOWLEDGMENTS

The author is indebted to Harold Fleisher and Robert Pole of IBM, Charles Koester of American Optical, Dean Anderson of Autonetics, Louis Cutrona of Conductron Corporation, Harrison Fuller of LFE and James Tippett of NSA for helpful discussions and suggestions.

REFERENCES

1. C. J. Koester, "Some Properties of Fiber Optics and Lasers," *Optical Processing of Information,* Pollock, Koester and Tippett, eds., Spartan Books, Baltimore, 1963, Part B.

2. W. F. Kosonocky, "Laser Digital Devices," *Optical and Electro-Optical Information Processing,* J. Tippett et al, eds., MIT Press, Cambridge, Mass., 1965.

3. C. J. Koester and C. H. Swope, "Some Laser Effects Potentially Useful in Optical Logic Functions," ibid.

4. G. J. Lasher and A. B. Fowler, "Mutually Quenched Injection Lasers as Bistable Devices," *IBM Journal of Research and Development,* Sept. 1964.

5. W. J. Poppelbaum, "Electro-Optical Information Processing," *Proceedings of the IFIP Congress,* 1965.

6. R. E. J. Moddes and L. O. Gilstrap, "Research on Optical Modulation and Learning Automata," *Optical and Electro-Optical Information Processing,* J. Tippett et al, eds., MIT Press, Cambridge, Mass., 1965.

7. J. C. Bliss and H. D. Crane, "Relative Motion and Nonlinear Photocells in Optical Image Processing," ibid.

8. E. N. Leith, L. J. Porcello and L. T. Cutrona, "Coherent Optical Data Processing Techniques," *Proceedings of the NEC,* 1959.

9. L. T. Cutrona, "Recent Developments in Coherent Optical Technology," *Optical and Electro-Optical Information Processing,* J. Tippett et al, eds., MIT Press, Cambridge, Mass., 1965.

10. G. Parrent, "Relation Between Bandwidth and Spatial Coherence in Experiments Involving Dispersion," *Journal of the Optical Society,* vol. 55, no. 9 (1965).

11. W. J. Poppelbaum et al, "Film Scanner," Quarterly Technical Progress Report of the Department of Computer Science, University of Illinois, July–Sept. 1965.

12. D. B. Anderson, "Application of Semiconductor Technology to Coherent Optical Transducers and Spatial Filters," *Optical and Electro-Optical Information Processing,* J. Tippett et al., eds., MIT Press, Cambridge, Mass., 1965.

13. G. W. Stroke, "Theoretical and Experimental Foundations of Optical Holography," ibid.

14. R. V. Pole et al, "Laser Deflection and Scanning by Internally Lifting Degeneracy of Multimode Cavities," ibid.

15. H. Fleisher et al, "An Optically Accessed Memory Using the Lippman Process for Information Storage," ibid.

16. A. Reich and G. H. Dorion, "Photochromic, High-Speed, Large Capacity, Semirandom Access Memory," ibid.

17. M. Von Ardenne, *Tabellen der Elektronenphysik, Ionenphysik und Uebermikroskopie,* Deutscher Verlag der Wissenschaften, 1956, vol. 1, p. 202.

18. C. F. Pulvari, letter in *Electronics,* Feb. 28, 1964.

19. E. Lindberg, "Solid Crystal Modulates Light Beam," *Electronics,* Dec. 20, 1963, p. 58.

20. J. W. Horton, R. V. Mazza and H. Dym, "The Scanistor—A Solid-State Image Scanner," *Proceedings of the IEEE,* Dec. 1964, p. 1513.

21. A. D. Berg and R. Smith, "An Electron Image Information Store," *AGARD Symposium on Opto-Electronics,* Paris, 1965.

22. N. H. Lehrer and R. D. Ketchpel, "Thin Film Conductive Memory Effects Applicable to Electron Devices," *Optical and Electro-Optical Information Processing,* J. Tippett et al, eds., MIT Press, Cambridge, Mass., 1965.

23. H. W. Fuller and R. J. Spain, "A Thin Magnetic Film for Wall Panel Display," ibid.

24. W. J. Poppelbaum et al, "Paramatrix System," Technical Progress Report of the Department of Computer Science, University of Illinois, June 1964.

25. ——, "The Artrix System," ibid, July–Sept. 1965.

BASIC THEORY OF PARTIAL COHERENCE

George B. Parrent, Jr.
Technical Operations Research
Burlington, Massachusetts

INTRODUCTION

The structure for a fundamental treatment of image formation problems already exists in the formalism of modern coherence theory as introduced by Wolf.[1] An adequate introduction to the subject is provided by Born and Wolf,[2] (Chap. 10), and a detailed description of most of the results of the theory to date may be found in Beran and Parrent.[3] Therefore it will not be necessary to review the subject extensively here. Rather, we shall limit ourselves to a statement of the pertinent definitions and a summary of the treatment of the imaging problem in coherence theory.

BASIC DEFINITIONS

Mutual Coherence Function

The basic entity in the theory of partial coherence is the mutual coherence function, $\Gamma_{12}(\tau)$, which may be defined by

$$\Gamma_{12}(\tau) \equiv (\underline{x}_1, \underline{x}_2, \tau) = \langle V(\underline{x}_1, t)\, V^*(\underline{x}_2, t+\tau)\rangle \qquad (1)$$

Here the underscore denotes position vector, the asterisk a complex conjugate, and the sharp brackets indicate a long time average,* i.e.,

$$\langle f\rangle \equiv \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} f\,dt \qquad (2)$$

*Equation (2) is equivalent to the definition introduced by Wolf, though in a slightly different form.

In (1) V is the analytic signal associated with the optical disturbance, which we assume to be a single Cartesian component of the electric field vector. In terms of the mutual coherence function, the complex degree of coherence, $\gamma_{12}(\tau)$ is defined as

$$\gamma_{12}(\tau) = \frac{\Gamma_{12}(\tau)}{\sqrt{\Gamma_{11}(0)\,\Gamma_{22}(0)}} \qquad (3)$$

It should be noted that the complex degree of coherence, like the mutual coherence function, is a function of seven variables, six position coordinates, and the time-delay coordinate τ. The physical significance of these parameters is illustrated by the example discussed below. The treatment of problems involving partially coherent light involves the solution of the two wave equations:

$$\nabla_s^2 \Gamma_{12}(\tau) = \frac{1}{c^2} \frac{\partial^2 \Gamma_{12}(\tau)}{\partial \tau^2} \qquad (s = 1, 2) \qquad (4)$$

where ∇_s^2 denotes the Laplacian operator in the coordinates of the point x_s. A typical problem involves determining the mutual coherence in the source or object plane, solving (4) to obtain the mutual coherence on a later surface, such as the image plane, and then recovering the intensity, I, in the plane of interest from the relation

$$I(x_1) = \Gamma(\underline{x}_1, \underline{x}_1, 0) \qquad (5)$$

Equation (5) follows directly from the definition of

the mutual coherence function and the properties of the analytic signal.

For a large class of problems the theory outlined in the preceding paragraph may be greatly simplified. These problems are characterized by the quasi-monochromatic approximations, which are stated as

$$\left\{\begin{matrix}\Delta\nu < \bar{\nu} \\ 1/\Delta\nu \ll |\tau|\end{matrix}\right\}$$

where $\Delta\nu$ is the spectral width. Of these two constraints, the second is obviously the more significant. White light may often be treated as quasi-monochromatic if the path differences, $c|\tau|$, involved in the experiment are suitably small. In those circumstances for which the approximations above are applicable, the mutual coherence function may be replaced by the mutual intensity function, $\Gamma(\underline{x}_1, \underline{x}_2)$,

$$\Gamma(\underline{x}_1, \underline{x}_2) \equiv \Gamma_{12} = \Gamma(\underline{x}_1, \underline{x}_2, 0) \qquad (6)$$

The complex degree of coherence reduces to $\gamma_{12}(0) \equiv \gamma_{12}$ and the wave equations (4) reduce to the two Helmholtz equations

$$\nabla_s^2 \Gamma_{12} + \underline{k}^2 \Gamma_{12} = 0 \qquad (s = 1, 2) \qquad (7)$$

where $\underline{k}$ is the wave number.

Coherent and Incoherent Fields

Equations (1) through (7) provide the basis of the theory of partial coherence as introduced by Wolf. To apply this theory to the imaging problem and recover the familiar limiting forms, several theorems due to Parrent are required. Principal among these are:

1. A field is coherent if and only if the mutual intensity function describing it can be factored in the form

   $$\Gamma_{12} = U(\underline{x}_1) U^*(\underline{x}_2)$$

   where

   $$\nabla^2 U(\underline{x}_1) + k^2 U(\underline{x}_1) = 0 \qquad (8)$$

2. An incoherent field cannot exist in free space; however, an incoherent source consistent with this result may be defined.

(For the proof of these theorems and their extensions to polychromatic fields the reader is referred to Beran and Parrent.[3]) Of particular significance for the problem of image evaluation is the second of these theorems. We shall reserve a discussion of the significance of the incoherent limit for a later point (a comprehensive treatment may be found in Beran and Parrent,[3] Chaps. 2 and 3).

The van Cittert-Zernike Theorem

An additional theorem is required before attacking the treatment of the image formation problem. The van Cittert–Zernike theorem may be stated as follows:

> The mutual intensity of the illumination derived from a distant incoherent source may be expressed in the form

$$\Gamma(\underline{x}_1, \underline{x}_2) = \int I(\underline{\xi}) e^{\frac{2\pi i}{\lambda R} \underline{\xi} \cdot (\underline{x}_1 - \underline{x}_2)} d\underline{\xi} \qquad (9)$$

Here I is the intensity distribution across the source, and R is the distance from the source plane to the observation plane. If the source is placed in the focal plane of a lens and the coherence of the emergent beam examined, it is found to follow the same law with the R replaced by the focal length f.

THE IMAGING PROBLEM

We may now direct our attention to the formulation of the general imaging problem. As will become clear in the following discussion, a basic description of image formation (at least as far as the lenses are concerned) already exists in coherence theory and, in fact, may be found in Refs. 2 and 3. This theory has not however been applied to the significant problems of image evaluation. Indeed, the theory has been applied to very few problems. In the next section the basic theory is outlined and those pertinent problems that have been solved are reviewed and discussed.

Review of Image Theory

In coherence theory an object is described by its mutual intensity* (or mutual coherence) distribution rather than its intensity distribution. Thus the object described by $\Gamma_0(\underline{\xi}_1, \underline{\xi}_2)$ and the relationship between object and image, $\Gamma_i(\underline{x}_1, \underline{x}_2)$, is developed by solving the two Helmholtz equations (7) subject to the appropriate boundary conditions. The gen-

*Our discussion in this section will be limited to quasi-monochromatic radiation. This serves to introduce the concepts, and at the same time keeps the development tractable.

eral solution is (see Ref. 3, Chaps. 7 and 8):

$$\Gamma_i(\underline{x}_1, \underline{x}_2) = \int\int \Gamma_0(\underline{\xi}_1, \underline{\xi}_2) K(\underline{x}_1 - \underline{\xi}_1) \cdot K^*(\underline{x}_2 - \underline{\xi}_2) d\underline{\xi}_1 d\underline{\xi}_2 \qquad (10)$$

Here K denotes the amplitude impulse response of the lens; i.e., denoting the complex transmission of the aperture by $A(\underline{\alpha})$, we may write

$$K(\underline{\xi}) = K\left(\frac{\underline{\beta}}{\lambda f}\right) = \int A(\underline{\alpha}) e^{\frac{2\pi i}{\lambda f} \underline{\alpha} \cdot \underline{\beta}} d\alpha \qquad (11)$$

The two familiar limits may be recovered from (10) by using the theorems of the previous section. Thus, in the coherent limit, $\Gamma_{12} = U_1 U_2^*$, and (10) reduces to

$$\Gamma_i(\underline{x}_1, \underline{x}_2) = \int U_0(\underline{\xi}_1) K(\underline{x}_1 - \underline{\xi}_1) d\underline{\xi}_1 \cdot \int U_0^*(\underline{\xi}_2) K^*(\underline{x}_2 - \underline{\xi}_2) d\underline{\xi}_2 \qquad (12)$$

From (12) and theorem 1 ("Coherent and Incoherent Fields," above), it is clear that the image of a coherently illuminated object is coherent. A somewhat more surprising result (and certainly more interesting in the image evaluation problem) is obtained in the incoherent limit. Thus, we may take* $\Gamma_{12} = I(\underline{\xi}_1)\delta(\underline{\xi}_1 - \underline{\xi}_2)$ to describe the object. The general image, Eq. (10), then reduces to

$$\Gamma_i(\underline{x}_1, \underline{x}_2) = \int I(\underline{\xi}) K(\underline{x}_1 - \underline{\xi}) K^*(\underline{x}_2 - \underline{\xi}) d\underline{\xi} \qquad (13)$$

From (13) it is clear that the image mutual intensity is no longer of the same form as the object mutual intensity; i.e., the image of an incoherent object is not incoherent but is partially coherent. This result will be seen to have rather far-reaching implications in the problems of image formation.

For most applications, the primary exposing radiation may be safely taken as incoherent. For example, sunlight is coherent only over a distance of approximately 1/20 mm. Thus, even a reconnaissance system which resolved an inch on the ground could probably be safely described by the incoherent limit of Eq. (10). In this case, the intensity in the image can be obtained by setting $\underline{x}_1 = \underline{x}_2$ in (13); thus

$$I_i(\underline{x}) = \int I_0(\underline{\xi}) \, | \, K(\underline{x} - \underline{\xi}) \, | \, ^2 d\underline{\xi} \qquad (14)$$

Equation (14) will be recognized as the familiar incoherent imaging equation. The difficulty arises, of course, when the scale of the mutual coherence function becomes comparable with the resolution of the optical instrument. (This point will be discussed at length in a later section.) While this condition is not likely to arise in the original taking system in the near future, it becomes serious in viewing and analyzing equipment such as microscopes, enlargers, and microdensitometers at the present state of the art. If one envisions improvements in taking equipment of a factor of two or more, it will become even more serious. This point will become clear as we analyze transilluminated objects.

While (10) represents the general solution to the partially coherent imaging problem, a more useful form for application to spatial filtering is obtained by considering the object to be a transparency that is transilluminated. This is, of course, the case in almost all viewing of reconnaissance imagery, and certainly in all uses of microscopes and microdensitometers in image evaluation. To describe this class of problems, it is necessary to describe the object in terms of its complex transmission $t(\underline{\xi})$. For transilluminated objects Eq. (10) may be expressed as

$$\Gamma_i(\underline{x}_1, \underline{x}_2) = \int\int \Gamma_0(\underline{\xi}_1, \underline{\xi}_2) t(\underline{\xi}_1) t^*(\underline{\xi}_2) \cdot K(\underline{x}_1 - \underline{\xi}_1) K^*(\underline{x}_2 - \underline{\xi}_2) d\underline{\xi}_1 d\underline{\xi}_2 \qquad (15)$$

In most cases, one is interested in the intensity of the image, which may be obtained from (15) by setting $\underline{x}_1 = \underline{x}_2$. Thus,

$$I_i(\underline{x}) = \int\int \Gamma_0(\underline{\xi}_1, \underline{\xi}_2) t(\underline{\xi}_1) t^*(\underline{\xi}_2) K(\underline{x} - \underline{\xi}_1) \cdot K^*(\underline{x} - \underline{\xi}_2) d\underline{\xi}_1 d\underline{\xi}_2 \qquad (16)$$

In (15) and (16) $\Gamma_0(\underline{\xi}_1, \underline{\xi}_2)$ must be interpreted as the coherence of the illumination incident on the transparency. The illumination in most cases of practical interest will be derived from a primary incoherent source. In this case $\Gamma_0(\underline{\xi}_1, \underline{\xi}_2)$ takes a special form (because of the van Cittert–Zernike theorem):

$$\Gamma_0(\underline{\xi}_1, \underline{\xi}_2) \equiv \Gamma_0(\underline{\xi}_1 - \underline{\xi}_2) \qquad (17)$$

That is, it becomes a function of coordinate differences only. Under these circumstances (16) becomes

$$I_i(\underline{x}) = \int\int \Gamma_0(\underline{\xi}_1 - \underline{\xi}_2) t(\underline{\xi}_1) t^*(\underline{\xi}_2) K(\underline{x} - \underline{\xi}_1) \cdot K^*(\underline{x} - \underline{\xi}_2) d\underline{\xi}_1 d\underline{\xi}_2 \qquad (18)$$

From (18) it is clear that for transilluminated objects the transition from object intensity $| \, t(\underline{\xi}) \, | \, ^2$ to image intensity is nonlinear. The significance of this conclusion is that the customary image evaluation techniques and criteria are not, in general, applicable to such systems. For example, knowing how such a system images sine waves or edges does not permit us to describe how it images other objects. Furthermore, the same optical system could

*Actually this form for the incoherent limit is only an approximation and must be used with care. However, it is sufficiently precise to illustrate the present problem.

be expected to yield different results if the coherence of the illumination varied. At high resolutions a small variation in the scale of the coherence function can produce dramatic changes in the image. This may account, in part, for the difficulty encountered in intercalibrating instruments in different laboratories, or in the cross-checking of microdensitometers that have essentially equivalent optical components but produce different results in edge trace analysis.

Since systems of this type are inherently nonlinear, it is impossible to characterize them by a transfer function. This point is easily established by taking the Fourier transform of both sides of (18). Thus,

$$\tilde{I}(\underline{\mu}) = \int \tilde{t}(\underline{\beta})\tilde{t}^*(\underline{\mu} - \underline{\beta})\{\int \tilde{\Gamma}[\underline{\mu} - (\underline{\sigma} + \underline{\beta})] \tilde{K}(\underline{\mu} - \underline{\sigma})\tilde{K}^*(\underline{\sigma})d\underline{\sigma}\}d\underline{\beta} \quad (19)$$

In (19) the inner integral is characteristic of the instrument only, while the factors $t(\underline{\beta})$ and $t^*(\underline{\mu} - \underline{\beta})$ are determined solely from the object spectrum. However, (19) is not in the form of "object spectrum times transfer function equals image spectrum." The inner integral has been referred to as a generalized transfer function, but that nomenclature is rather misleading since the function is not used as a transfer function at all. A better terminology is the more cumbersome one introduced by Wolf, the "transmission cross coefficient," which emphasizes that it is a function of two frequencies.

With these general reservations in mind, we may direct our attention to the development of the system analysis for spatial filtering systems.

SYSTEM ANALYSIS

In this section the relationships between "object" and "image" for three cases of imaging with coherent radiation are derived. Denoting by z_1 and z_2, respectively, the object and image distances, we define these cases as follows:

1. $\frac{1}{z_1} + \frac{1}{z_2} = \frac{1}{f}$
2. $z_1 = z_2 = f$
3. $z_1 = 0, \quad z_2 = \mathrm{f}$

Condition (1) produces an image in the ordinary sense only if the object is in the near field of the lens. Condition (2) yields an "image" which is the Fourier transform of the object, and condition (3) yields a Fourier transform multiplied by a quadratic phase term.

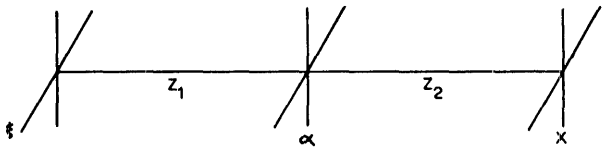


Figure 1. Coordinate system.

The geometry is illustrated in Fig. 1, in which ξ is the coordinate in object space, α is the coordinate in the aperture plane, and x is the coordinate image space. Assuming paraxial optics and ignoring obliquity factors, we may express the relation between object and "image" as follows (Beran and Parrent,[3] Chaps. 3 and 7):

$$\Gamma_i(x_1, x_2) = \int\int\int\int \Gamma_0(\xi_1, \xi_2)R(\alpha_1)R^*(\alpha_2) \cdot e^{ik\left[r(\xi_1,\alpha_1) - r(\xi_2,\alpha_2) - \frac{\alpha_1^2}{2f} + \frac{\alpha_2^2}{2f} + r(\alpha_1,x_1) - r(\alpha_2,x_2)\right]} \cdot d\xi_1 d\xi_2 d\alpha_1 d\alpha_2 \quad (20)$$

Here $R(\alpha)$ describes the transmission of the aperture and all integrals are infinite. The term $\alpha^2/2f$ is, of course, the saggital approximation and the r's remain from the Green's function.

Assuming coherent quasi-monochromatic radiation, we find that

$$\Gamma_0(\xi_1, \xi_2) = U_0(\xi_1)U_0^*(\xi_2) \quad (21)$$

and the image becomes

$$\Gamma_i(x_1, x_2) = U_i(x_1)U_i^*(x_2) \quad (22)$$

where

$$U_i(x) = \int\int U_0(\xi)R(\alpha)e^{ik\left[r(\xi,\alpha) - \frac{\alpha^2}{2f} + r(\alpha,x)\right]} d\xi d\alpha \quad (23)$$

Ignoring terms of order $\frac{K(\xi - \alpha)^4}{z^3}$ we may write

$$r(\xi, \alpha) = z_1 + \frac{(\xi - \alpha)^2}{2z_1} \quad (24)$$

and

$$r(\alpha, x) = z_2 + \frac{(\alpha - x)^2}{2z_2} \quad (25)$$

Hence, omitting constant phase terms we may rewrite (23) as

$$U_i(x) = \int U_0(\xi) \cdot \int R(\alpha)e^{ik\left[\frac{\alpha^2}{2}\left(\frac{1}{z_1} + \frac{1}{z_2} - \frac{1}{f}\right) + \frac{\xi^2}{2z_1} + \frac{x^2}{2z_2} - \alpha\left(\frac{x}{z_2} + \frac{\xi}{z_1}\right)\right]} \cdot d\alpha\, d\xi \quad (26)$$

Case (1): $\frac{1}{z_1} + \frac{1}{z_2} = \frac{1}{f}$

Under these conditions (26) becomes

$$U_i(x) = \int U_0(\xi) \int R(\alpha) e^{ik\left[\frac{\xi^2}{2z_1} - \frac{\alpha\xi}{z_1} + \frac{x^2}{2z_2} - \frac{\alpha x}{z_2}\right]} d\alpha\, d\xi \quad (27)$$

Consider first the integral

$$\int U_0(\xi) e^{\frac{ik[\xi^2 - 2\alpha\xi]}{2z_1}} d\xi = e^{-\frac{ik\alpha^2}{2z_1}} U(\alpha) \quad (28)$$

Here

$$U(\alpha) = \int U_0(\xi) e^{\frac{ik(\xi-\alpha)^2}{2z_1}} d\xi \quad (29)$$

Equations (28) and (29) are obtained by simply completing the square in the exponent. We may now write (27) as

$$U_i(x) = \int R(\alpha) U(\alpha) e^{ik\left[\frac{x^2}{2z_2} - \frac{\alpha x}{z_2} - \frac{\alpha^2}{2z_1}\right]} d\alpha \quad (30)$$

Or completing the square again we have

$$U_i(x) = e^{\frac{ikx^2\left[\frac{z_1}{z_2} + \frac{z_1^2}{z_2^2}\right]}{2z_1}} \int R(\alpha) U(\alpha) e^{-\frac{ik\left(\alpha + \frac{z_1}{z_2}x\right)^2}{2z_1}} d\alpha \quad (31)$$

$$= e^{\frac{ikx^2 z_1}{2fz_2}} \int R(\alpha) U(\alpha) e^{-\frac{ik\left(\alpha + \frac{z_1}{z_2}x\right)^2}{2z_1}} d\alpha \quad (32)$$

If the lens is unapodized and unaberrated, (32) becomes

$$U_i(x) = e^{\frac{ikx^2 z_1}{2fz_2}} \int_{-a}^{a} U(\alpha) e^{-\frac{ik\left(\alpha + \frac{z_1}{z_2}x\right)^2}{2z_1}} d\alpha \quad (33)$$

Under the condition

$$a^2 \gg \frac{2z_1}{k} \quad (34)$$

i.e., object in near field of lens, the limits $-a$ to a (the aperture size) may be regarded as infinite and (33) may be evaluated by the inversion theorem for Fresnel transforms, giving

$$U_i(x) = e^{ikx^2 g(z_1, z_2)} U_0\left(\frac{z_1}{z_2} x\right) \quad (35)$$

that is, an image multiplied by a quadratic phase term.

Case (2): $z_1 = z_2 = f$

Under these conditions, (26) reduces to

$$U_i(x) = \int\int U_0(\xi) R(\alpha) e^{\frac{ik[\alpha^2 + \xi^2 + x^2 - 2\alpha(x+\xi)]}{2f}} d\alpha\, d\xi \quad (36)$$

Completing the square on the exponent in (36) gives

$$U_i(x) = \int U_0(\xi) e^{-\frac{ik\xi x}{\xi - f}} \int_{-a}^{a} e^{\frac{ik[\alpha - (x+\xi)]^2}{2f}} d\alpha\, d\xi \quad (37)$$

Here an ideal lens is assumed again. Provided α, x, and ξ and the condition

$$a^2 \gg \frac{2f}{k}$$

is met, the inner integral yields a constant C and (37) becomes

$$U_i(x) = C \int U_0(\xi) e^{-\frac{ik\xi x}{\xi f}} d\xi \quad (38)$$

or

$$U_i(x) = C\tilde{U}_0\left(\frac{x}{2\lambda f}\right) \quad (39)$$

i.e., a Fourier transform with no quadratic phase term.

Case (3): $z_1 = 0 \qquad z_2 = f$

Under these conditions we have immediately from Eq. (20)

$$U_i(x) = e^{\frac{ikx^2}{2f}} \int U_0(\alpha) R(\alpha) e^{-\frac{ik\alpha x}{2f}} d\alpha \quad (40)$$

or

$$U_i(x) = e^{\frac{ikx^2}{2f}} \tilde{U}_0\left(\frac{x}{2\lambda f}\right) * \tilde{R}\left(\frac{x}{2\lambda f}\right) \quad (41)$$

Spatial filtering systems are properly constructed around the configuration characteristics of case 2. That is, this setup is used to display the Fourier transform of the input transparency. A second transparency is placed in the Fourier transform plane to modify the spectrum. Then the process is repeated and the "filtered" image is displayed in the final Fourier transform plane.

In each spatial filtering experiment the coherence conditions should be adjusted in accordance with the foregoing analysis in order to be sure that the approximations of the theory have been satisfied and at the same time to minimize the degrading effects associated with coherent imaging. These effects arise primarily from the fact that such systems are nonlinear in intensity. Some examples of these effects are shown below.

EXPERIMENTAL EXAMPLE OF COHERENCE EFFECTS

A simple experiment that illustrates the effects of the spatial and temporal coherence is to form two-beam interference fringes by division of a wavefront. Figure 2 shows the result of such an experiment. Figure 2a shows high-contrast fringes formed with a He–Ne gas laser illuminating a pair of small circular

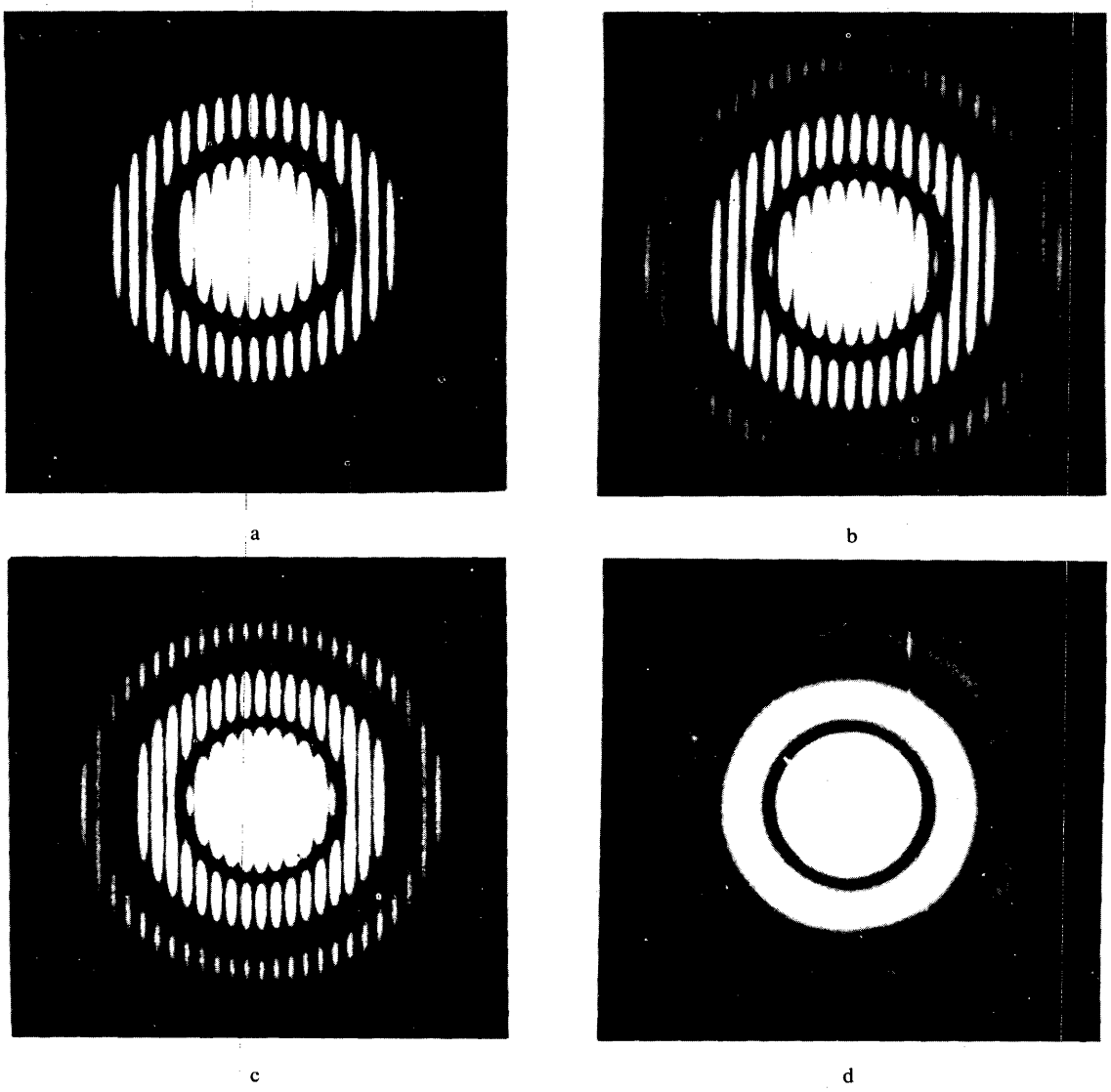


Figure 2. Effect of coherence length. A and B—with gas lasers; C and D—with mercury arc.

apertures. The envelope function is the diffraction pattern of the single aperture. In Fig. 2b, a piece of plane optical quality glass 0.5 mm thick was introduced in front of one of the apertures only, to add an extra optical path. Again illuminating with the He–Ne gas laser, we observe no difference in the fringe contrast. However, when the experiment is repeated with a coherent field produced by a mercury arc lamp without the glass plate, high-contrast fringes are again seen (Fig. 2c), but with the glass in place, the fringes disappear (Fig. 2d). The slight scale change between the two pairs of illustrations results from the different wavelengths (6328 Å for He–Ne and 5461 Å for the Hg green line). This illustration shows that the coherence length of the mercury arc radiation is quite small. Both fields were spatially coherent but the coherence lengths were quite different. Introducing fine ground glass across the pair of pinholes results in the intensity distribution of Fig. 3. The extra paths introduced by the ground glass did not exceed the coherence length; hence, high-contrast fringes are seen over the whole field. A discussion of these types of speckle patterns in terms of their auto-correlation function and their power-spectral density are to be found in a paper by Goldfischer.[4] An attempt to build a coherent projection printer is reported by Milinowski,[5] in which a rotating piece of

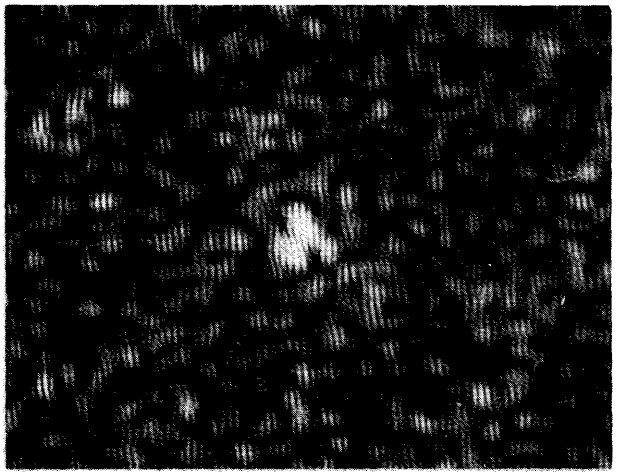


Figure 3. Two-beam interference with diffusing plate.

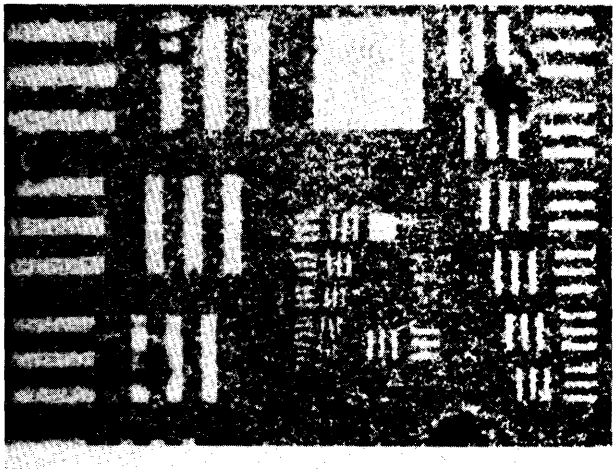
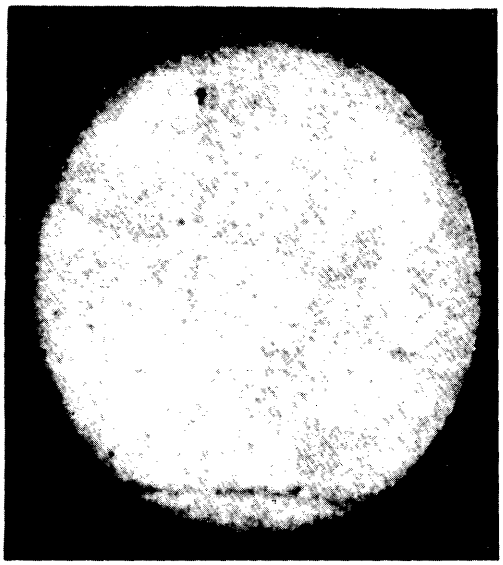


Figure 4. Photograph in reflected coherent light.

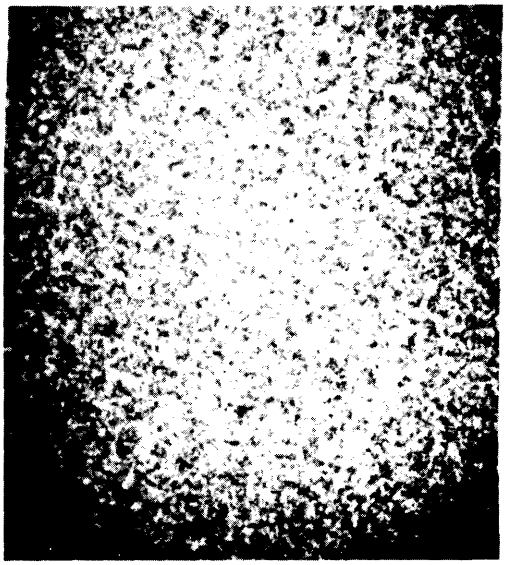
ground glass is used to remove some of the coherence effects.

REFLECTED LIGHT

The different speckle patterns formed when coherent light is reflected from a rough surface have been commented upon a number of times and perhaps form the most objectionable feature of coherent imaging by reflected light (as opposed to transmitted light discussed in the last section). Figure 4 shows a standard bar target that has been printed on a matte photographic paper and then rephotographed in reflected coherent light. The edge-ringing effects are masked by the speckle patterns. The speckling is produced by the interference between the scattered light and is determined by the coherence length of the incident radiation. Figure 5a is a photograph of a portion of a cement-block wall illuminated by a mercury arc so that the light is spatially coherent. In Figure 5a the same portion of the wall is illuminated by a gas laser; the speckling completely obscures any structure of the wall. Both beams had approximately the same spatial coherence, but the gas laser has a considerable longer coherence length.



a



b

Figure 5. Effect of coherence length. A—spatially coherent only; B—spatially and temporally coherent.

REFERENCES

1. E. Wolf, *Proc. Roy. Soc.*, vol. (A) 230, p. 246 (1954).

2. M. Born and E. Wolf, *Principles of Optics*, 2nd ed., Pergamon Press, New York, 1964.

3. M. Beran and G. B. Parrent, *Theory of Partial Coherence*, Prentice-Hall, Englewood Cliffs, N.J., 1963.

4. L. I. Goldfischer, *J. Opt. Soc. Am.*, vol. 55, p. 247 (1965).

5. A. S. Milinowski, ibid, vol. 54, p. 1406 (1964).

THE ROLE OF COHERENT OPTICAL SYSTEMS IN DATA PROCESSING

L. J. Cutrona
Conductron Corporation
Ann Arbor, Michigan

INTRODUCTION

This paper describes a number of signal processing techniques in which coherent optical techniques play an important role. The techniques are powerful and of great versatility.

Examples of both two-dimensional and multichannel one-dimensional signal operations are described. Of particular importance is the fact that the most general linear operation can be mechanized optically. Further examples show how antenna patterns can be simulated optically.

A configuration useful for achieving fine resolution in radar by the generation of a synthetic antenna is described.

Finally, the use of several optical configurations for communication purposes are discussed.

FUNDAMENTAL PRINCIPLES

Much of the capability of optical configurations arises from the ease with which certain one-dimensional and two-dimensional spectral analyses are made.[1-4] Two basic configurations appear repeatedly. These configurations are

1. A configuration using a spherical lens which produces two-dimensional diffraction, and
2. A configuration consisting of a spherical lens in conjunction with a cylindrical lens which produces a multiplicity of one-dimensional diffraction patterns.

The basic configuration for obtaining two-dimensional diffraction patterns is shown in Fig. 1.

In this figure S represents a source of light, L_1 represents a collimating lens, P_1 represents the input plane in which a transparency is placed, and lens L_2 is the spherical lens which is the essential element for producing a two-dimensional diffraction pattern. Plane P_2 is the plane in which the two-dimensional spectrum (of the transparency in plane P_1) is exhibited.

In order that the distribution of light in plane P_2 be the two-dimensional spectrum analysis of the density distribution of the transparency in plane P_1, it is necessary that planes P_1 and P_2 be spaced a focal length on either side of lens L_2. If $f(x, y)$ represents the amplitude of light emerging from plane P_1, then the distribution of light amplitude in plane P_2 is given by Eq. (1):

$$F(\alpha, \beta) = \int\int f(x, y) e^{jk(\alpha x + \beta y)}\, dx\, dy \qquad (1)$$

In Eq. (1), the amplitude of the light in plane P_2

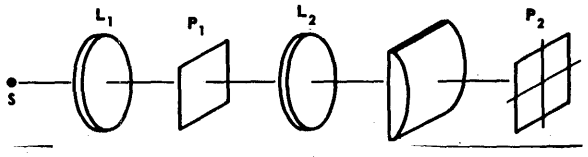


Figure 1. Configuration for two-dimensional spectrum analysis.

is given by $F(\alpha, \beta)$. Here k represents the wave number of the light while α and β represent the direction cosines of the diffracted beam with respect to the x and y axes.

The configuration of Fig. 1 can be converted to a multichannel one-dimensional diffraction equipment by the addition of a cylindrical lens to the configuration of Fig. 1. The cylindrical lens is placed between planes P_1 and P_2 to give the configuration shown in Fig. 2.

In this case the distribution of light in plane P_2 is given by Eq. (2). It will be noted that this expression indicates distribution of light corresponding to a multichannel spectrum analysis. The parameter y is an index referring to a given channel. The other parameters have been previously defined.

$$F(\alpha, y) = \int f(x, y) e^{jk\alpha x} \, dx \qquad (2)$$

Figures 1 and 2 will be seen to appear in a number of configurations in the following sections. A photograph of equipment using the configuration of Fig. 2 is shown in Fig. 3.

LINEAR OPERATIONS

Signal processing operations include a large number of linear operations. Among these operations are spectrum analysis, filtering, auto-correlation, cross-correlation, etc. Each of these operations, including (a) the most general linear operation on a function of a single continuous variable, and (b) the most general linear operations on vectors, can be mechanized optically.

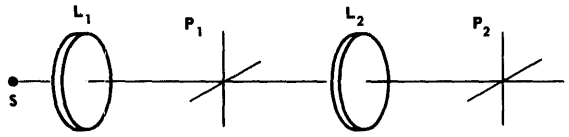


Figure 2. Configuration for one-dimensional multichannel diffraction.

Spectrum Analysis

The configuration of Figs. 1 and 2 are those essential for two-dimensional spectrum analysis or multichannel one-dimensional spectrum analysis. The operation of these configurations formed the content of the above section.

Filtering

It is often desirable to perform filtering operations upon recorded signals. In such cases, it is usually required to view the signals corresponding to these altered spectra. To achieve this alteration of the spectrum and viewing of the result, it is necessary to modify the optical configurations shown in Figs. 1 and 2 to those shown in Figs. 4 and 5, respectively.

The configurations in Figs. 4 and 5 permit operations on the spectra by filtering operations in plane P_2. A number of filtering operations are possible:

In the simplest case, one can achieve bandpass and bandstop filtering in plane P_2. A bandpass is achieved by having a transparent region at the appropriate location in plane P_2. A bandstop is achieved by locating an opaque spot at the appropriate position in plane P_2.

A more complicated filtering operation can be achieved by placing in plane P_2 a transparency having a density varying as a function of position. This corresponds to a filter which changes the relative magnitudes of the spectral components.

A different filter is one in which phase variations are desired. Difficulties in making filters of this kind arise from the short wavelengths of light.

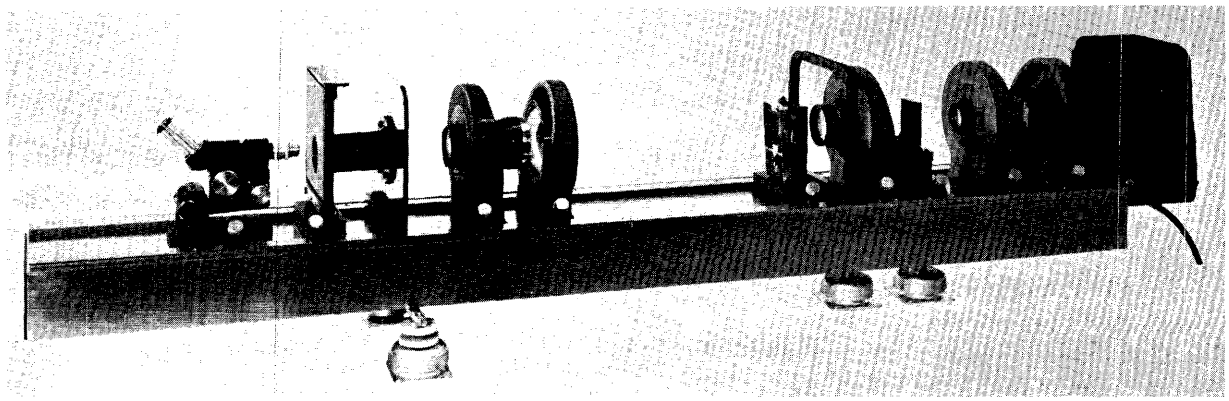


Figure 3. Equipment for one-dimensional multichannel spectrum analysis.

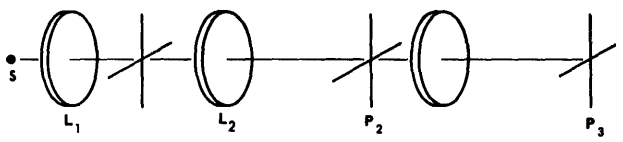


Figure 4. Configuration for two-dimensional spectrum analysis and filtering.

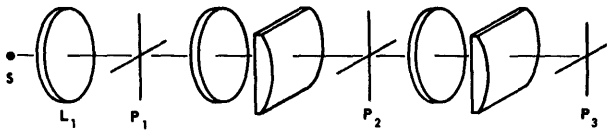


Figure 5. Configuration for one-dimensional multichannel spectrum analysis and filtering.

In the most general case, a filter is desired in which both the magnitude and the phase shift of the spectra can be varied. In this case one can place two transparencies in contact in plane P_2; one of the transparencies has a varying density while the other has a varying thickness or phase shift. This permits the most general filtering operation to be performed optically.

Information theory indicates a number of cases in which a desirable signal operation is that of passing a signal through a matched filter. In general the matched filter will require a variation of both magnitude and phase, hence the general filter consisting of variable magnitude and phase is required.

A scheme for achieving the equivalent of a complex filter (magnitude and phase shift both variable) makes use of a technique recently demonstrated by Leith and Upatnieks,[5] and Vander Lugt.[6]

In this case a recording is made in which both phase and magnitude are preserved but in which this information can be recovered from a transparency having density variations only.

Auto-Correlation and Cross-Correlation

Important linear operations are those of auto-correlation and cross-correlation. These functions will be considered together since the equipment needed to mechanize the operations is identical. In performing a cross-correlation the operations performed are those indicated by Eq. (3) while an auto-correlation is given by Eq. (4).

$$\varphi_{fg}(x_0) = \int f(x)\, g(x - x_0)\, dx \qquad (3)$$

$$\varphi_{ff}(x_0) = \int f(x)\, f(x - x_0)\, dx \qquad (4)$$

It will be noted from Eqs. (3) and (4) that to mechanize these operations, techniques are needed for performing multiplication, translation, and integration. A configuration capable of performing a multiplicity of one-dimensional auto-correlations or cross-correlations is given in Fig. 6. In this figure the source and collimating lens to the left of the plane P_1 causes a plane coherent wave to be incident on the transparency $f(x, y)$.

The optics between planes P_1 and P_2 causes the multichannel spectrum analysis of $f(x, y)$ to appear in the plane P_2. The optics between planes P_2 and P_3 perform a second multichannel spectrum analysis of the signals in plane P_2. Thus, incident upon P_3 is the function $f(x, y)$. If one looks through plane P_3 toward the source, the distribution of light will be the product $f(x, y)\, g(x, y)$.

Let the holder which contains the function $f(x, y)$ have provision for transporting this transparency along the x axes. If this displacement is through a distance x_0, then the distribution of light in plane P_3 looking toward the source will be the product $f(x, y)\, g(x - x_0, y)$.

The combination of spherical and cylindrical optics between planes P_3 and P_4 cause a multichannel spectrum analysis of the light distribution emerging from plane P_3. Hence, the distribution of light in plane P_4 is described by Eq. (5).

$$\varphi(x_0, y; \propto) = \int f(x, y)\, g(x - x_0, y)\, e^{j \propto z}\, dx \qquad (5)$$

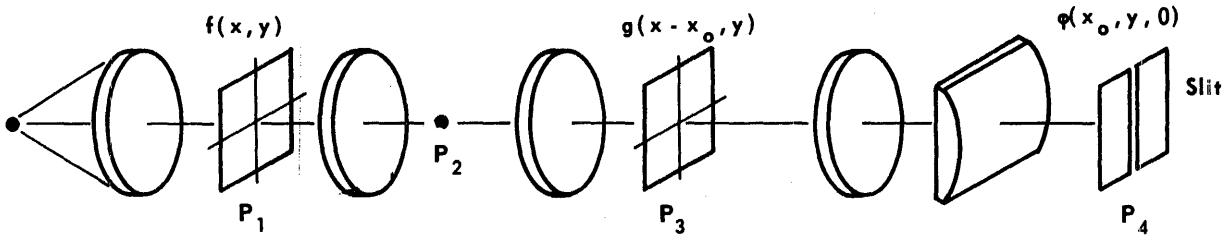
where $\propto = 2\pi/\lambda \sin \theta$.

It will be noted that Eq. (5) resembles Eq. (3) except that a multiplicity of operations is performed (one for each value of y) and that the factor $e^{j \propto x}$ appears as a factor in the integrand. In Eq. (5), $\propto = 0$ corresponds to the light in a slit parallel to the y axis. If only the light in this slit is recorded, the exponential factor in Eq. (5) assumes the value unity. In this case, Eq. (5) becomes identical with Eq. (3) except for its multichannel feature. This result is written as Eq. (6):

$$\varphi(x_0, y, 0) = \varphi_{fg}(x, y) \qquad (6)$$

As the plane P_3 is transported, at a given position in plane P_4, there will appear an amplitude of light corresponding to the value of the cross-correlation function for that value of the displacement x_0. This auto-correlation function can be recorded by transporting a film past the slit. The configuration in Fig. 6 is, thus, capable of performing a multiplicity of simultaneous correlations.

To perform an auto-correlation using the configuration of Fig. 6 one uses a second copy of $f(x, y)$ in plane P_3.



$$\varphi(x_0, y, 0) = \int f(x, y)\ g(x - x_0, y)\ dx$$

Figure 6. Cross-correlator configuration.

In the mechanization shown as Fig. 6 a relatively complicated optical arrangement was shown to image plane P_1 onto plane P_3. It is necessary to use this configuration in order to remove errors arising from bias levels used in recording the signals in planes P_1 and P_3.

Auto-correlations and cross-correlations are important operations and there are many instances for which information theory indicates these as optimum signal detection and/or parameter estimation operations. It will be noted that the configuration of Fig. 6 performs a multiplicity of such auto-correlations or cross-correlations simultaneously. There is no difficulty in recording onto film a density of 50 cycles per mm. Hence 35mm film can be used in configuration of Fig. 7 to perform simultaneously more than 1000 simultaneous auto-correlations or cross-correlations. Equipments having the configuration of Fig. 6 are commercially available. A photograph of such a device is shown as Fig. 7.

General Linear Operation

The most general linear operation [0] on a function $f(t)$ to produce an output $g(t)$ can be written in the form given by Eq. (7):

$$g(t) = 0[f(t)] = \int h(\tau, t) f(\tau)\, d\tau \qquad (7)$$

In this equation, the nature of the operation to be performed determines the kernel function $h(\tau, t)$. The fact that Eq. (7) represents a general linear operation is discussed in texts dealing with functional analysis.[7] Pertinent discussion has also been given in publications by L. A. Zadeh.[8]

In order to mechanize the operation given by Eq. (8), the configuration shown in Fig. 8 may be used.

$$g(t) = \int h(\tau, t) f(\tau)\, d\tau \qquad (8)$$

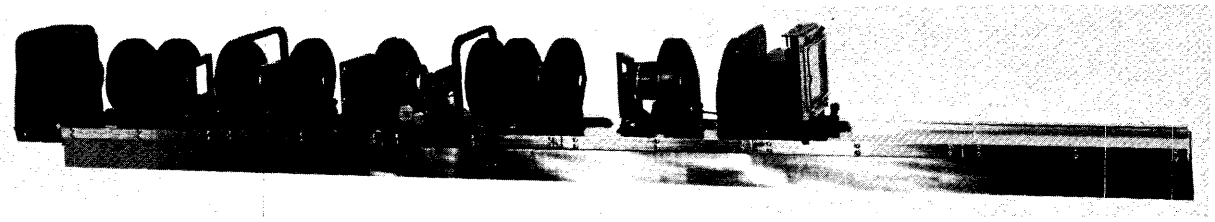
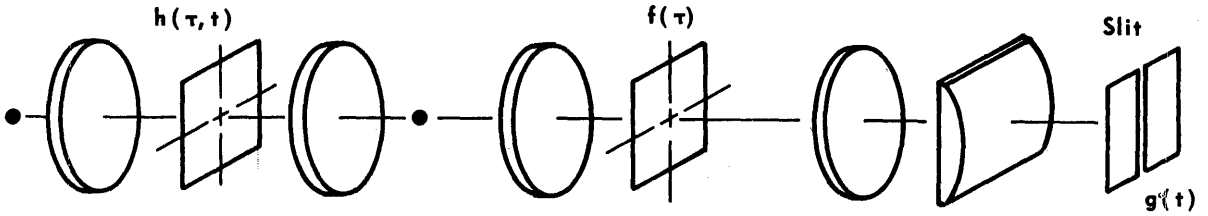


Figure 7. Cross-correlator.



$$g(t) = \int h(\tau, t)\ f(\tau)\ d\tau$$

Figure 8. General linear operation configuration.

If one looks toward the left, the light amplitudes in the plane containing $f(\tau)$ contains the product of $h(\tau, t)$ with $f(\tau)$.

Between the transparency $f(\tau)$ and the output slit in which $g(t)$ is found, is a pair of lenses, one of which is spherical, the other of which is cylindrical. This configuration performs the function of causing the line-by-line spectral analysis of the light in the $f(\tau)$ plane to be displayed in the output plane. The distribution of light in the output plane is described by Eq. (9). It will be noted that Eq. (9) is a somewhat more general operation than that described by Eq. (8), and that it describes a two-dimensional distribution of light in the output plane.

$$I(t,\omega) = \int h(\tau,t)f(\tau)e^{-j\omega\tau}d\tau \qquad (9)$$

It will be further noted that Eq. (9) becomes identical with Eq. (8) if ω is set equal to 0. Thus, performing a linear operation expressed as Eq. (8) is accomplished simply by observing the light which is present in the central slit in the output plane.

Thus, performing a general linear operation optically requires the configuration shown in Fig. 8 together with the ability to record on two transparencies the functions $h(\tau,t)$, which represents the operation to be performed, and the function $f(\tau)$, which represents the function upon which the operation is performed. The result of the operation is present in a centrally located slit in the output plane of the equipment.

It is known that, if a number of linear operations are performed in tandem, one can represent the tandem sequence of operations by a single equivalent operation. Hence, Eq. (8) represents not only a single operation but a sequence of linear operations, if such is desired.

Matrix Multiplication

Eq. (8) and Fig. 8 are pertinent when a general linear operation or a function of one variable is to be performed. It is useful, however, to consider the case in which a linear operation is to be performed in a space of a finite number of dimensions. In this case each input and output quantity of interest will be a vector (n-tuple) and the linear operation will be that of matrix multiplication. If V_1 and V_2 repre-

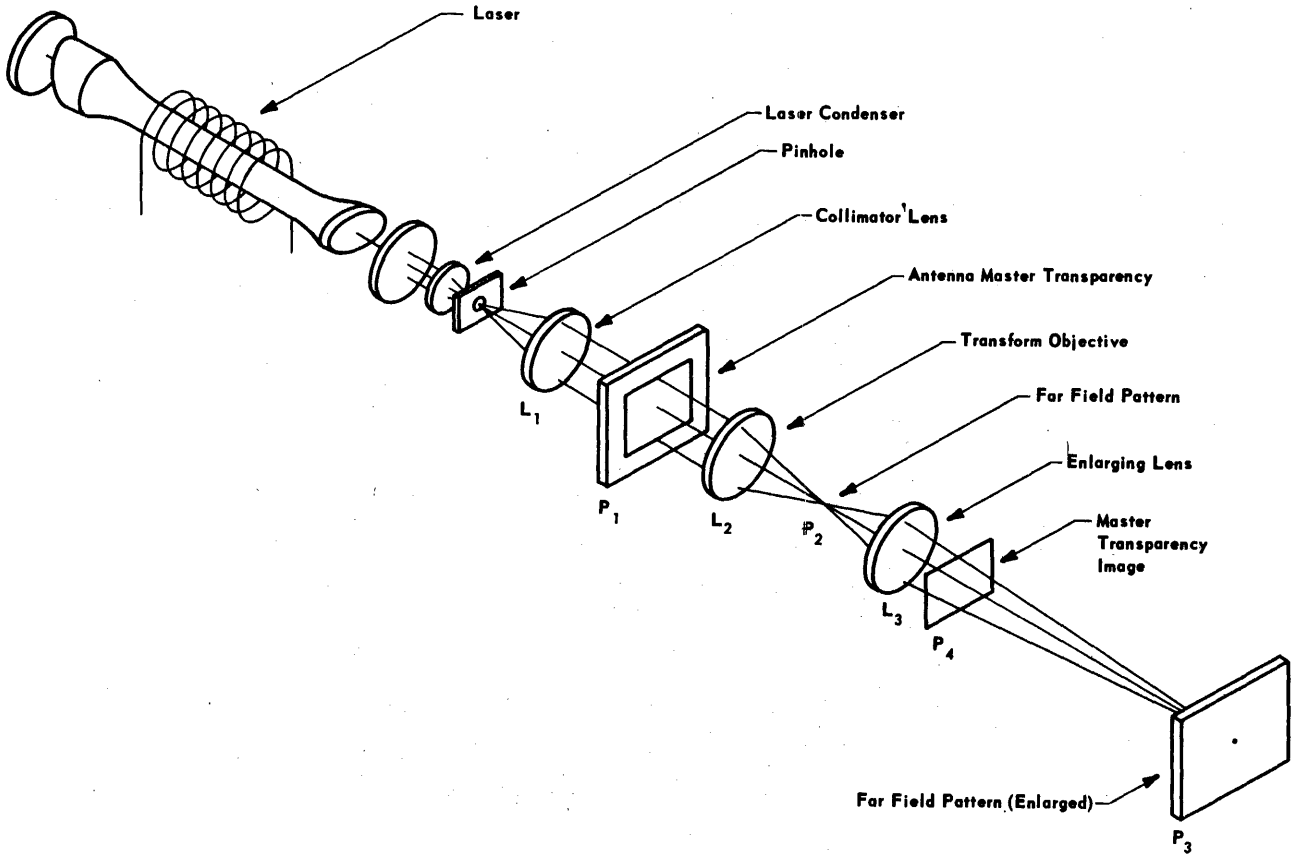


Figure 9. Microwave antenna simulator at optical frequency.

sent two vectors and if M represents a matrix one may write

$$V_2 = MV_1 \tag{10}$$

to indicate that V_2 is derived by a linear operation on V_1.

In Eq. (10) V_1 and V_2 are vectors (column matrices whereas M is a k by n matrix. These matrices may be of the form given by Eqs. (11) and (12).

$$V_1 = \left| \begin{pmatrix} v_{11} \\ \vdots \\ v_{1n} \end{pmatrix} \right| \qquad V_2 = \left| \begin{pmatrix} v_{21} \\ \vdots \\ v_{2n} \end{pmatrix} \right| \tag{11}$$

$$M = \left| \begin{pmatrix} m_{11} \ldots m_{1n} \\ \vdots \quad\quad \vdots \\ m_{n1} \quad m_{nn} \end{pmatrix} \right| \tag{12}$$

The operation indicated by Eq. (10) can be performed by the optical configuration shown in Fig. 8. In this case, the transparency $h(\tau + t)$ is replaced by a rectangular array representing the matrix M, and $f(\tau)$ is replaced by V_1. The output slit now contains the values of V_2 instead of $g(t)$.

Thus, the configuration of Fig. 8 makes possible the general linear operation for t, a continuous variable, as well as for discrete variables.

ANTENNA PATTERN SIMULATION

The far field pattern of an antenna can be computed from its illumination function $f(x,y)$ by the use of the relation

$$F(\alpha,\beta) = \int\int_{\substack{\text{over} \\ \text{aperture}}} f(x,y)e^{-jk(\alpha x+\beta y)}dx\,dy \tag{13}$$

In this equation α and β are direction cosines of the beam, $F(\alpha,\beta)$ is the far field pattern, and k is defined by

$$k = \frac{2\pi}{\lambda} \tag{14}$$

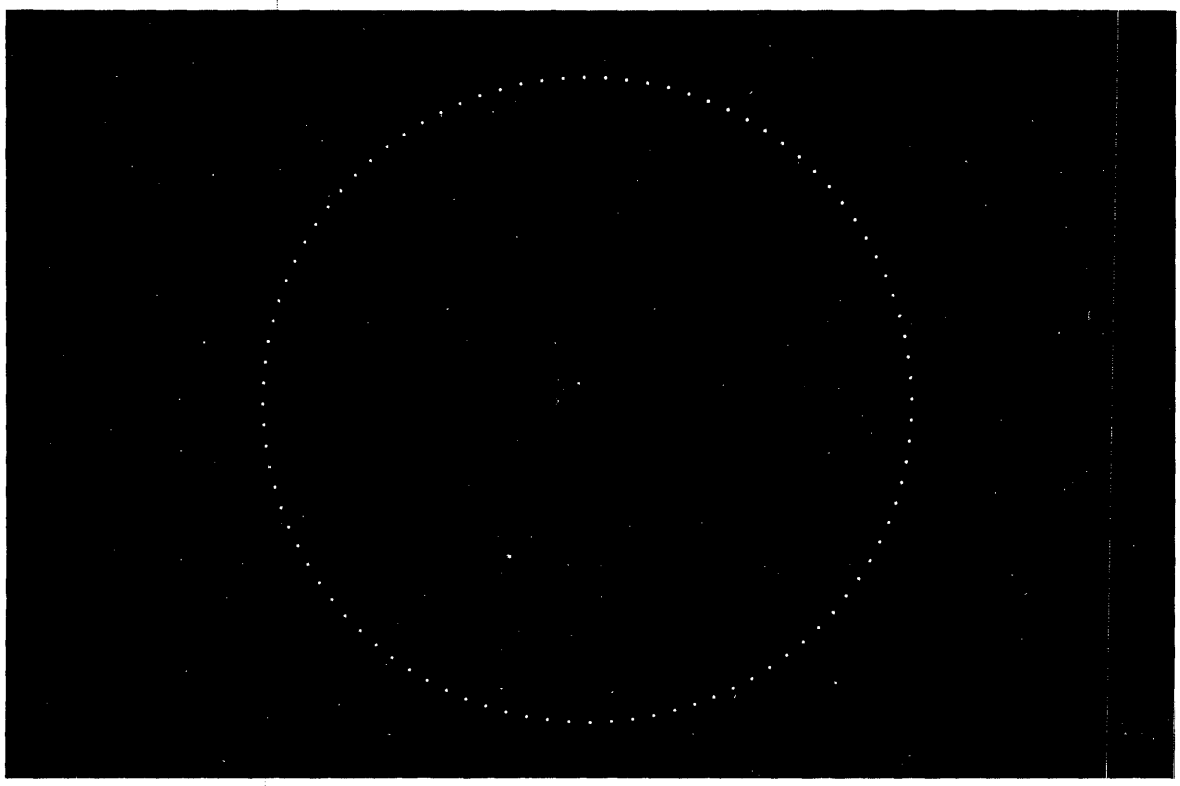


Figure 10. Antenna simulation—Illumination Plane.

This quantity is the wave number and λ represents the wavelength used. If Eq. (13) is compared with Eq. (1) it will be seen to have the same form. Thus, the optical configuration shown in Fig. 9 can be used to display the far field pattern of an antenna.[9] In this case, a transparency containing the aperture function $f(x,y)$ is placed in plane P_1 and the far field pattern is observed in plane P_2.

In some cases, the far field pattern found in plane P_2 may be too small. It is desirable in this case to use another lens to magnify the image found in plane P_2. Such a configuration is shown in Fig. 9 where an enlarged image of the field in plane P_2 is displayed in plane P_3.

In fig. 9, one finds that an image of plane P_1 occurs between lenses L_3 and plane P_3 at plane P_4. Thus, between planes P_4 and P_3 one has an opportunity to observe the pattern as it emerges from the illumination function through the near field until at plane P_3 the far field pattern is obtained. With this configuration, studies of the relationship between near field and far field can be made. In addition, by inserting perturbations into the regions between plane P_4 and P_3, for example, by simulating a nonhomogeneous medium, it is possible to observe the effects of perturbations on the far field pattern.

A series of photographs showing the far field developing from the illumination function for an array antenna[9] is shown in Figs. 10–15. This set of photographs is the optical simulation of an antenna array being built by J. P. Wild in Australia. The array consists of 96 parabolas arranged on a circle about 1½ miles in diameter. Each parabola has a diameter of 45 feet. The frequency of operation is 80 megacycles.

Assuming that the far field pattern begins at a distance $2D^2/\lambda$ the far field pattern begins approximately 2400 miles from earth. This poses real problems for measurement of the far field pattern. The configuration of Fig. 9, however, enables one to obtain not only the far field pattern shown in Figs. 14 and 15 but also views of the near field pattern.

Thus, antenna simulation is another demonstration of the versatility of optical equipments.

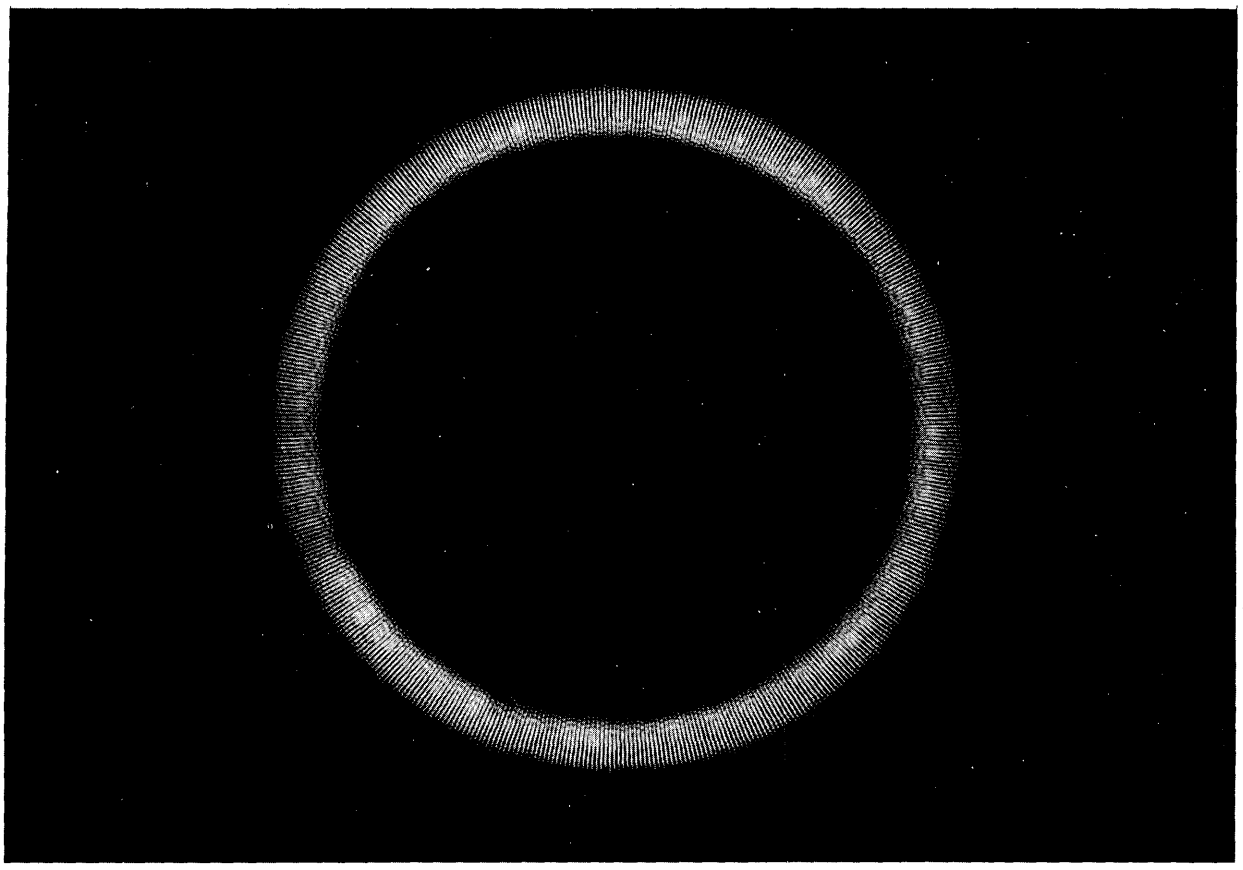


Figure 11. Antenna simulation—near field.

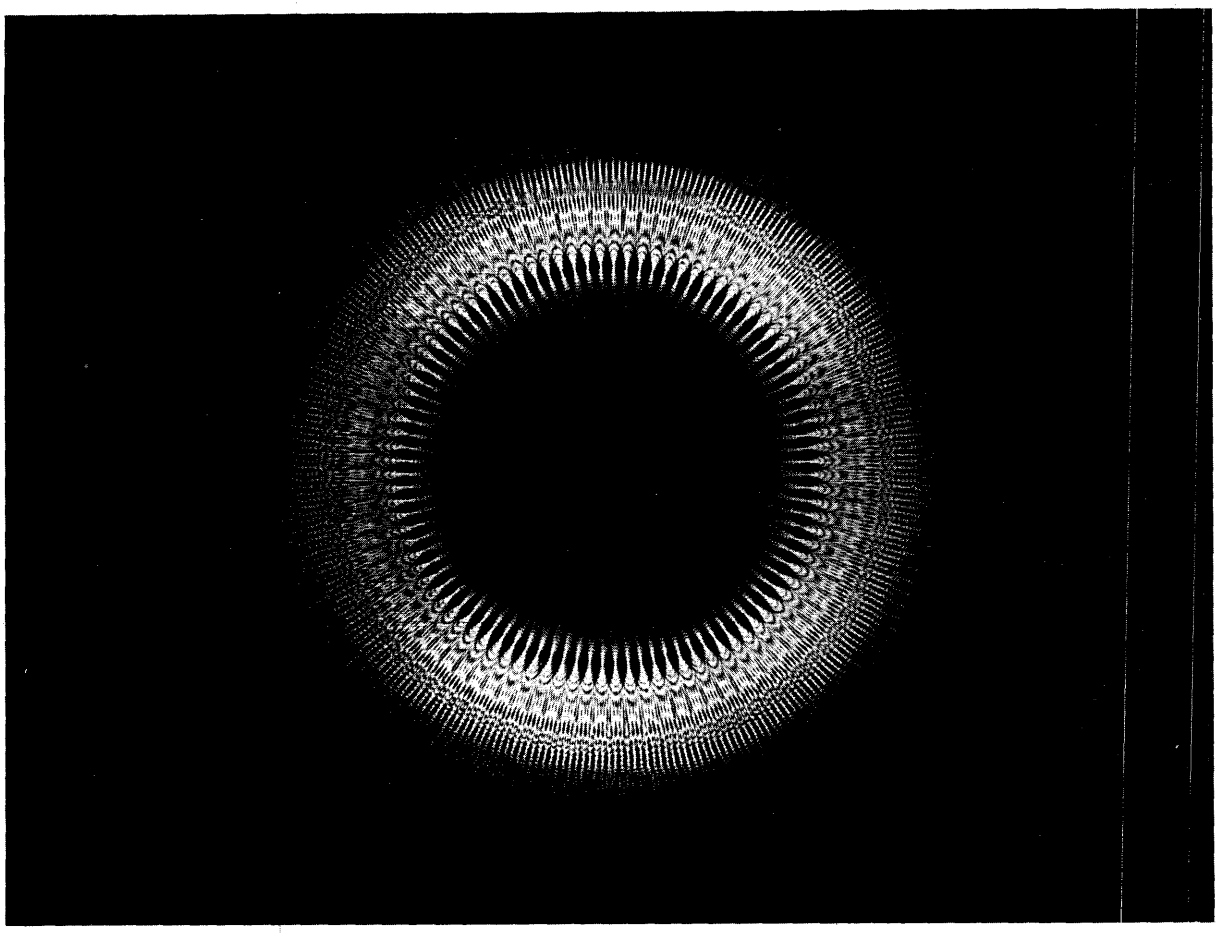


Figure 12. Antenna simulation—near field.

SYNTHETIC ANTENNA GENERATION

One of the most successful and important applications of optical data processing has been that of *synthetic antenna generation* in airborne radar applications. Here, a sequence of signals collected as an airborne radar is translated along a straight line is used to achieve the effect of a long linear array. The term *synthetic antenna* derives from the fact that signal processing generates the long linear array.

The fundamental idea can best be grasped by consideration of Fig. 16. For the physical antenna array shown in Fig. 16a, the individual transmitting and receiving elements are dipoles. While each dipole has a broad radiation pattern, the assemblage of dipoles is made to produce a narrow antenna beam by making the electrical lengths of the individual transmission lines such that signals arriving in phase at the dipoles are added in phase at the main transmission line to the radar. Since transmission and reception at each of the dipoles is simultaneous, this fixed adjustment of phases serves to maintain the desired beam pattern.

Figure 16b depicts the generation of the synthetic array. In this case the individual elements of the array, as indicated by the x's, do not exist simultaneously. Starting with the position at the extreme left, a radar pulse is transmitted and the return signals recorded. A short time later the aircraft has carried the side-looking antenna to the second position where another pulse is transmitted and the received signals recorded. In this way a signal is recorded for each of the positions of the synthetic array. To achieve the effect of a linear array such as Fig. 16a, the return signals must contain phase information which is preserved in the recording. Then by appropriate data processing it should be

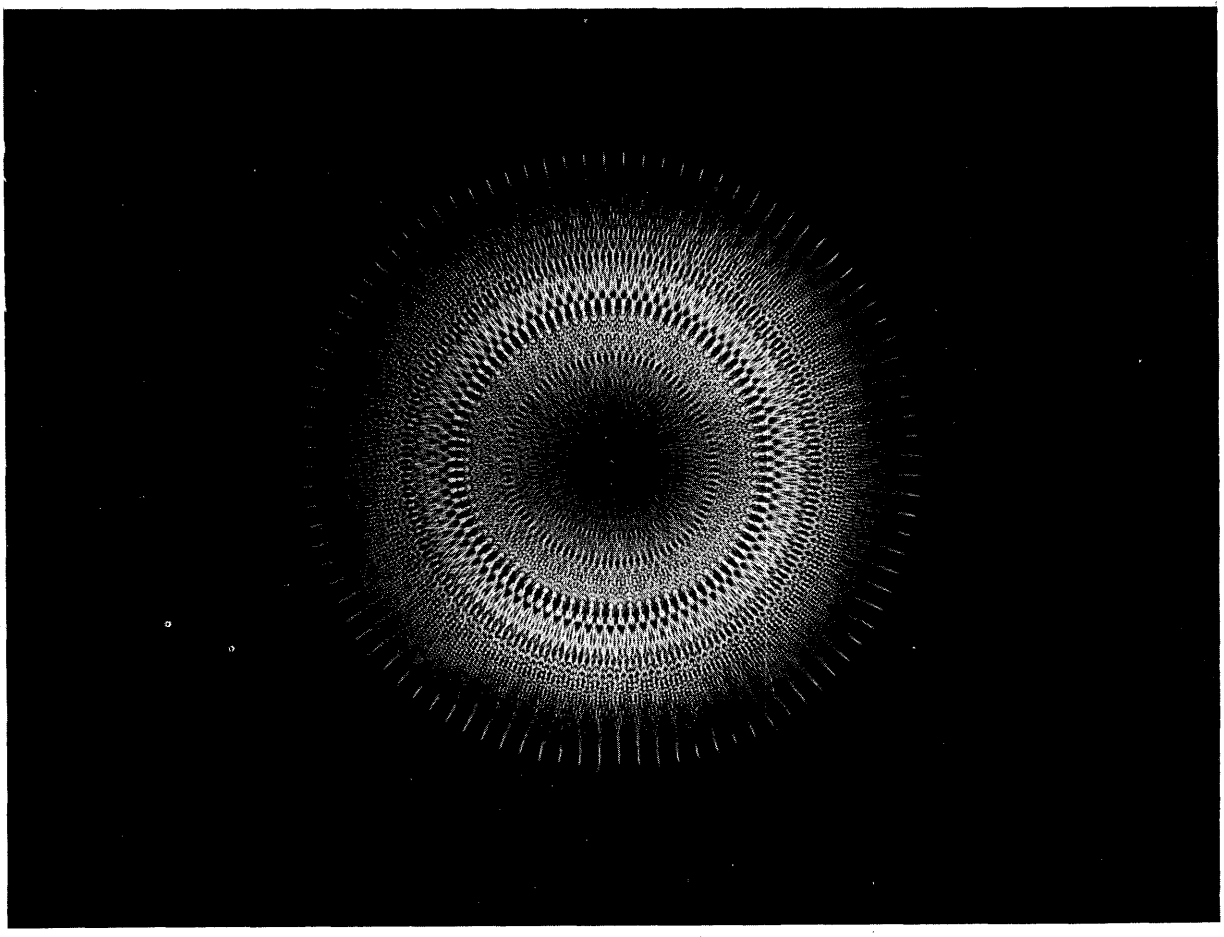


Figure 13. Antenna simulation—near field.

possible to retrieve the stored data and combine it in proper phase to synthesize the desired effect of a narrow-beamed antenna.

It was evident that the radar would have to have excellent transmitter frequency stability and a stable frequency reference for use in comparing the phase of the return signals with the phase of the transmitted pulses. The method of signal storage (recording) had to preserve the range information so that the data processing could be accomplished separately for each element of radar range.

The well-known formula for antenna beamwidth showed that, at least in theory, the synthetic-antenna concept had a great potential for fine angular resolution. The half-power beamwidth β of the physical side-looking antenna (Fig. 17) is

$$\beta = k \frac{\lambda}{D} \text{ radians} \qquad (15)$$

in which

λ is the wavelength of the radar,
D is the length of the physical aperture, and
k is the illumination factor (greater than unity).

The distance across the antenna beam for a radar range r is βr, which represents the amount of data that can be collected between half-power points for this range. If this amount of data is processed at each range, this would represent a synthetic antenna length

$$L_s = \beta r \text{ feet} \qquad (16)$$

if $\wp$ is expressed in feet. The phase information obtained for each of the element positions of the synthesized antenna is based on the round-trip distance between the side-looking antenna and echoing objects; for the physical array the phase depends on the one-way path length. As a consequence of this difference, the phase difference between elements

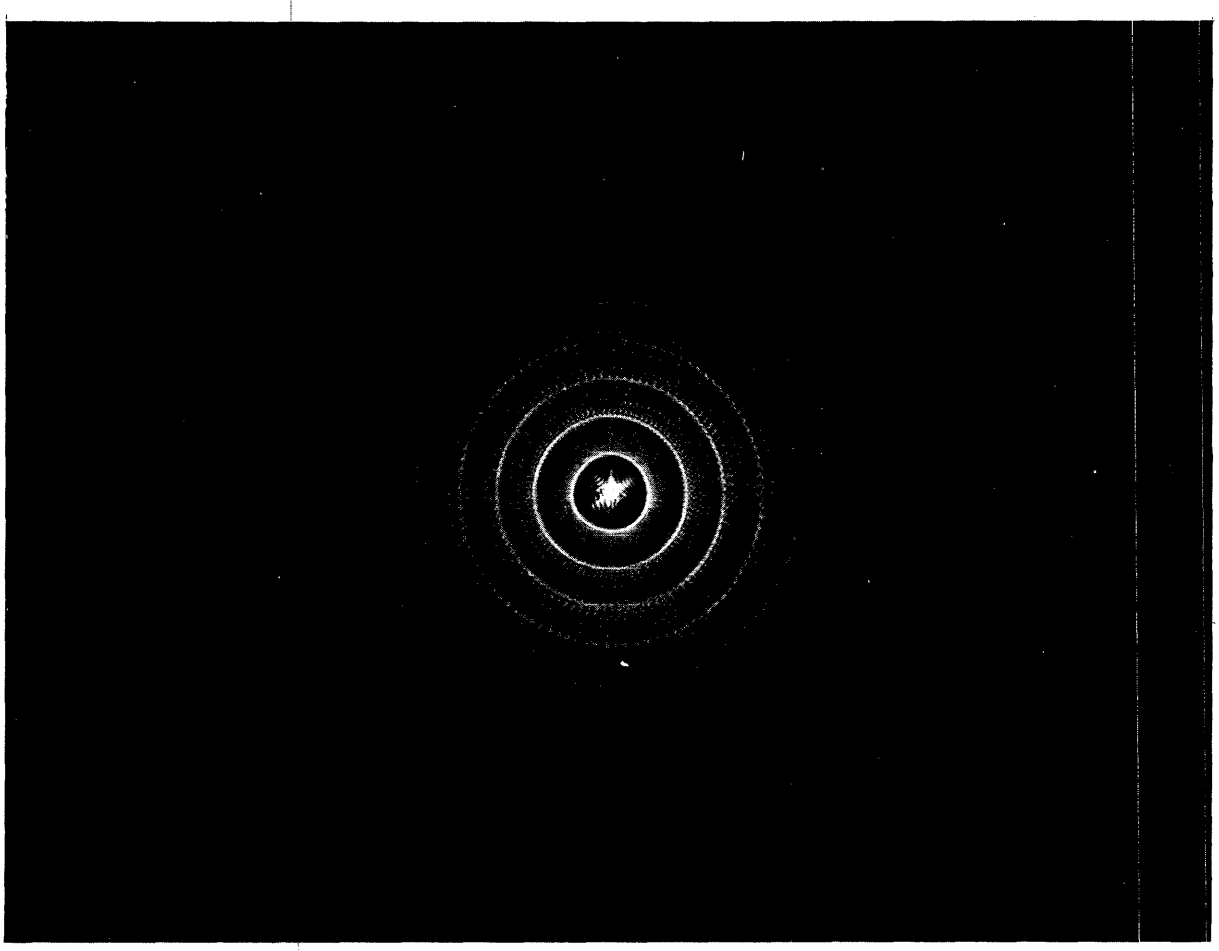


Figure 14. Antenna simulation—far field.

of equally spaced elements of a synthetic array is twice that of a physical array with the same spacing; a synthetic antenna has half the beamwidth of a physical antenna of a given length. That is,

$$\beta_s = k_1 \frac{\lambda}{2L_s} \text{ radians} \tag{17}$$

in which k_1 is the illumination function for the synthetic-sized antenna. Using Eq. (15) this can be expressed in terms of the size of the physical antenna and radar range as

$$\beta_s = \frac{k_1}{k} \frac{D}{2r} \text{ radians} \tag{18}$$

This indicates that the synthetic beamwidth β_s is not only independent of frequency, but also decreases with radar range.

To obtain a measure of resolution, the usual assumption was made that targets could be resolved in angle if they were separated by one antenna beamwidth. For the beamwidth β_s of the synthetic antenna, the distance across the beam at any radar range r is

$$\Delta X = \beta_s r \frac{k_1}{k} \simeq \frac{D}{2} \text{ feet} \tag{19}$$

if the length of the physical antenna aperture D is expressed in feet.

In theory, then, the resolution in the azimuth direction for a synthetic antenna radar is independent of range, independent of radar frequency, and smaller than the physical length of the actual side-looking antenna carried by the aircraft. This assumes the generation of a synthetic antenna of a length equal to the distance across the radar beam at each radar range.

In the analysis represented by Eqs. (15)–(19), the generation of a *focused* synthetic antenna was implicitly assumed. If unfocused synthetic antennas are generated, one obtains a resolution given by Eq. (21).

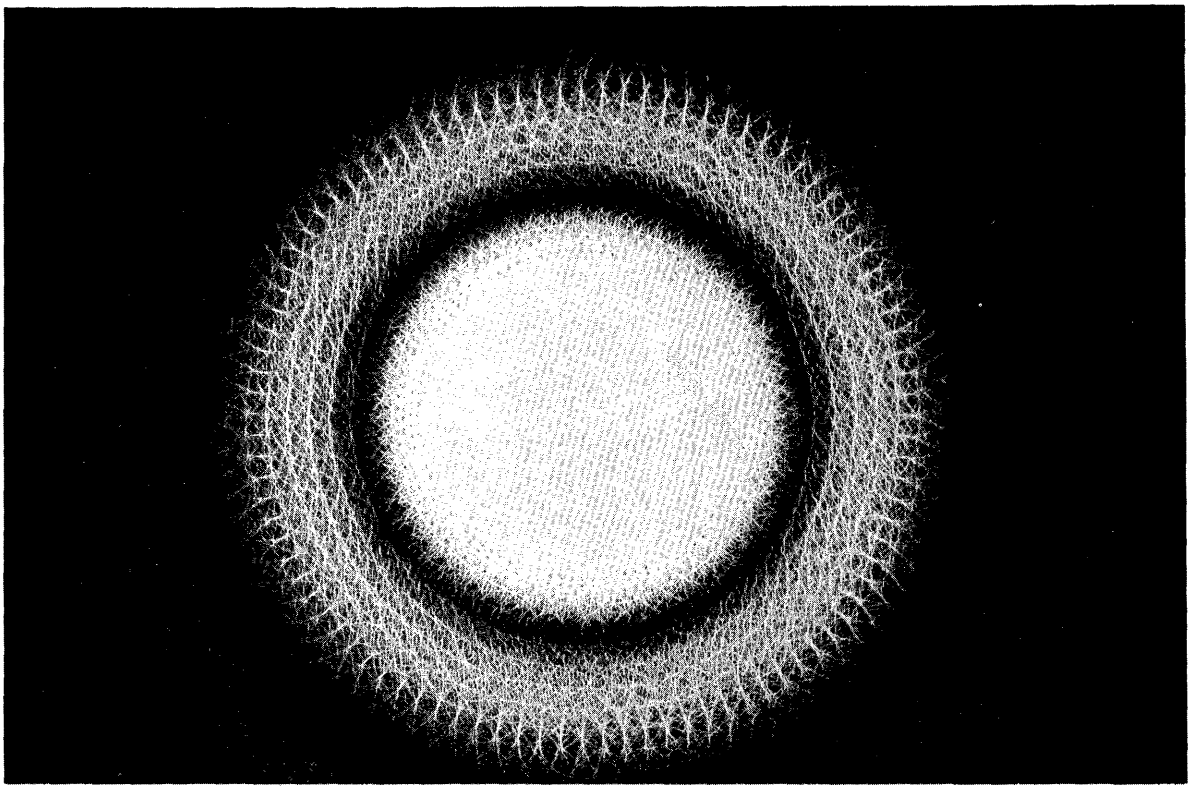


Figure 15. Antenna simulation—far field (longer exposure).

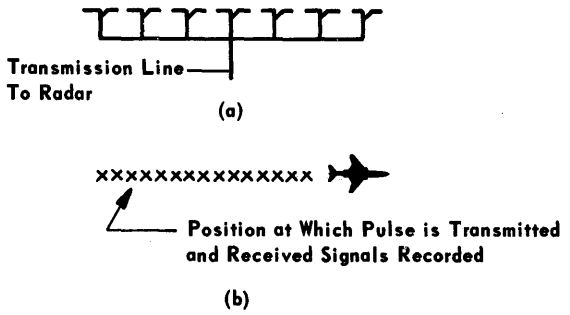


Figure 16. (a) Physical array of dipoles. (b) Synthetic array generation.

It is useful to compare the azimuth resolution capabilities of conventional radars, unfocused synthetic antenna radars, and focused synthetic antenna radars. The theoretical results are given by Eqs. (20), (21), and (22). These results are plotted as Fig. 18.

1. *The conventional technique:* In this technique azimuth resolution depends upon the width of the radiated beam.
2. *The unfocused synthetic antenna technique:* In this case the synthetic antenna length is made as long as the unfocused technique permits.
3. *The focused synthetic antenna technique:* In this case the synthetic antenna length is made equal to the linear width of the radiated beam at each range.

As is shown in the sections which follow, the linear transverse resolution for the *conventional case*

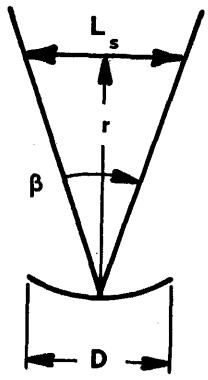


Figure 17. Synthetic antenna geometry.

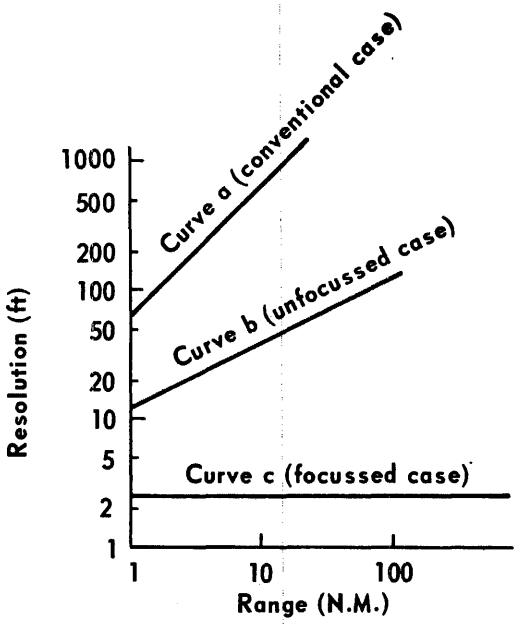


Figure 18. Aximuth resolution for three cases: (a) conventional; (b) unfocused; (c) focused.

is given by

$$\text{Resolution}_{\text{conv}} = \frac{\lambda R}{D} \tag{20}$$

For the *unfocused case*, the resolution is given by

$$\text{Resolution}_{\text{unf}} = \tfrac{1}{2}\sqrt{\lambda R} \tag{21}$$

whereas for the *focused case*, the resolution is given by

$$\text{Resolution}_{\text{foc}} = \frac{D}{2} \tag{22}$$

In the above expressions λ is the wavelength of the radar signal transmitted, D is the horizontal aperture of the antenna, and R is the radar range.

Figure 18 is a plot of the resolution for each of these cases as a function of radar range. This plot is for an antenna aperture of 5 ft and a wavelength of 0.1 ft.

One of the most successful applications of coherent optical data processing is the conversion of raw data obtained from a synthetic antenna type radar system into a fine-resolution radar map. Synthetic aperture radars may well become the most important of the ground-mapping types of radars, since they have a resolution potential at long ranges which is considerably greater than that of other types of mapping radars. Coherent optical processing is eminently well suited to the processing tasks which arise in such radar systems.

The azimuth or angular resolution of a conventional radar is limited by the width of the physical radar beam, which is given by λ/D, where λ is the wavelength and D is the antenna width. Because radar wavelengths are several orders of magnitude larger than optical wavelengths, very large values of D must be used if radars are to achieve angular resolutions comparable with those of photoreconnaissance systems. If one wishes to have fine (linear) azimuth resolution at long ranges, the required antenna length will be of the order of hundreds or thousands of feet; obviously, such an antenna could not be carried by an aircraft.

The synthetic antenna technique[10-12] offers a way around this impasse: the aircraft carries a small, side-looking antenna, producing a beam that is relatively wide in the azimuth direction, which scans the terrain by virtue of the aircraft motion. The antenna is carried by the aircraft to a sequence of positions which can be treated as if they were the positions occupied by the individual elements of a linear antenna array. At each position, the antenna radiates a pulse, then receives and stores the reflected signal. These stored data are then processed in a manner analogous to the coherent weighted summation carried out in a large linear array. The processed radar signals bear a quantitative similarity to those which would be obtained if a large antenna were used; in particular, the resolution and the signal-to-noise ratio are greatly improved by the signal processing.

Array-type antennas add, or integrate, the returns received on each of the array elements. The synthetic antenna falls into the array category. One can readily generate synthetic apertures which are so long that they will only realize their full gain and azimuth-resolution capabilities if they are focused at the range at which the radar is operating. The focusing operation represents a phase adjustment of the signal received on each array so that, in the summation process, the contributions from all array elements are combined in phase.

The Radar Signal

An aircraft carries an antenna which illuminates a ground-swath parallel to the flight path; the radar beam is oriented in azimuth roughly normal to the direction of flight; at range R, the azimuth lineal beamwidth βR is much larger than the desired azimuth resolution at that range. Finally, the radar is *coherent*, that is, the receiver has available a

reference signal from which the transmitted signal was derived.

Synthetic antenna radars derive range information through pulsing, and derive fine azimuth resolution by processing Doppler-shifted radar returns which lie in a spectral band which is adequately sampled by the pulse rate of the radar. For our purposes, we may neglect this intermediate sampling process and consider only the reconstructed azimuth histories which are easily derived from the samples. We will assume that the entire radar receiver and processor behave as a linear system; we can then investigate the response of the radar to a single *point* target, and by superposition extend the results to apply to realistic reflective complexes.

Examples of radar imagery obtained by the use of synthetic antenna techniques are shown in Figs. 19 and 20.

The radar data is recorded onto film as an intermediate step. This film, called a *signal history film*, is then used in an optical configuration such as that shown in Fig. 21.

In this figure, the conical lens is the primary element responsible for *focusing*, while the combination of cylindrical and spherical lenses causes the device to become *multichannel* so that the separate range intervals remain resolved.

COMMUNICATIONS APPLICATIONS

The essential components of a communications system are shown in Fig. 22. In such a system the

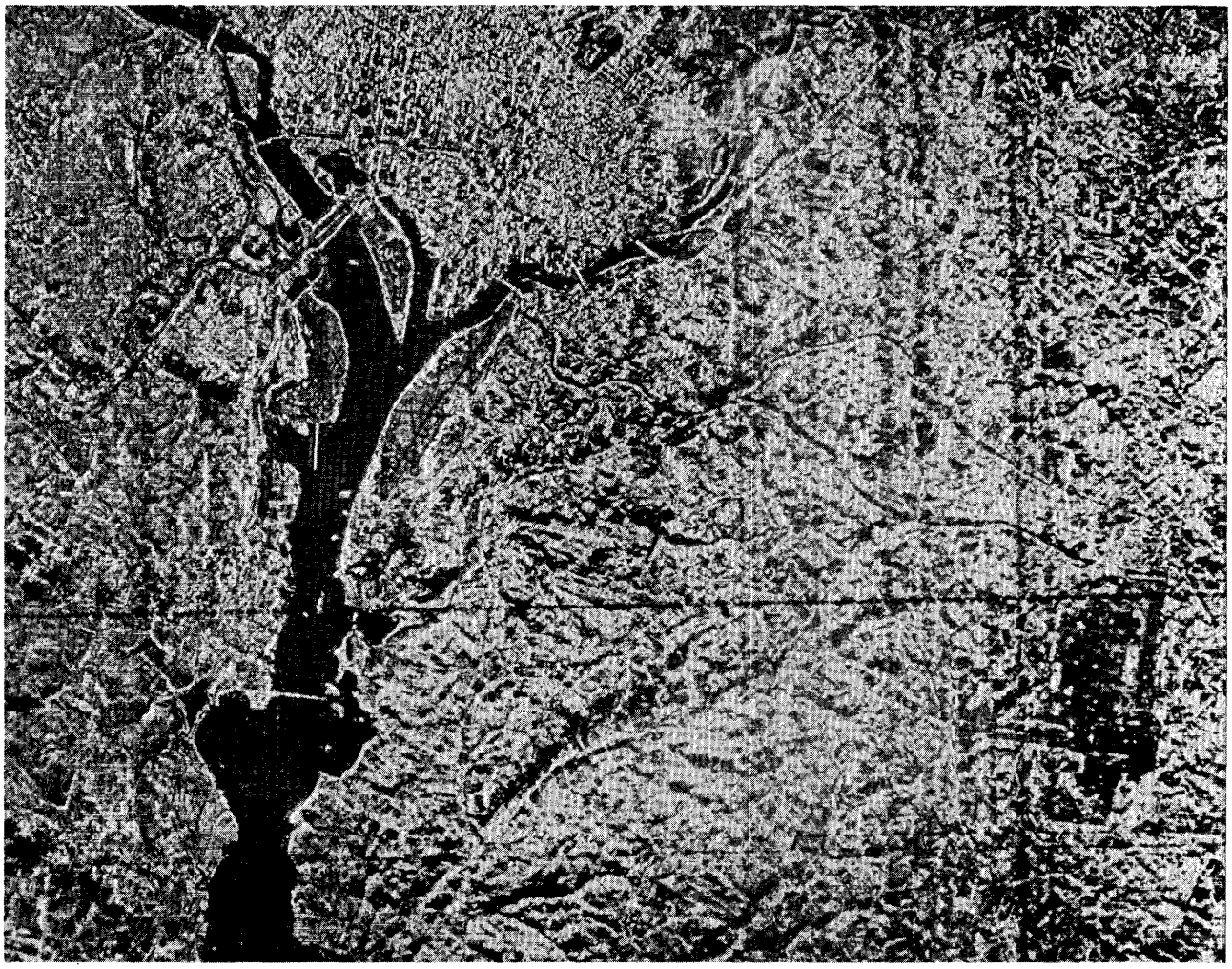


Figure 19. Fine resolution radar image, Washington, D.C.

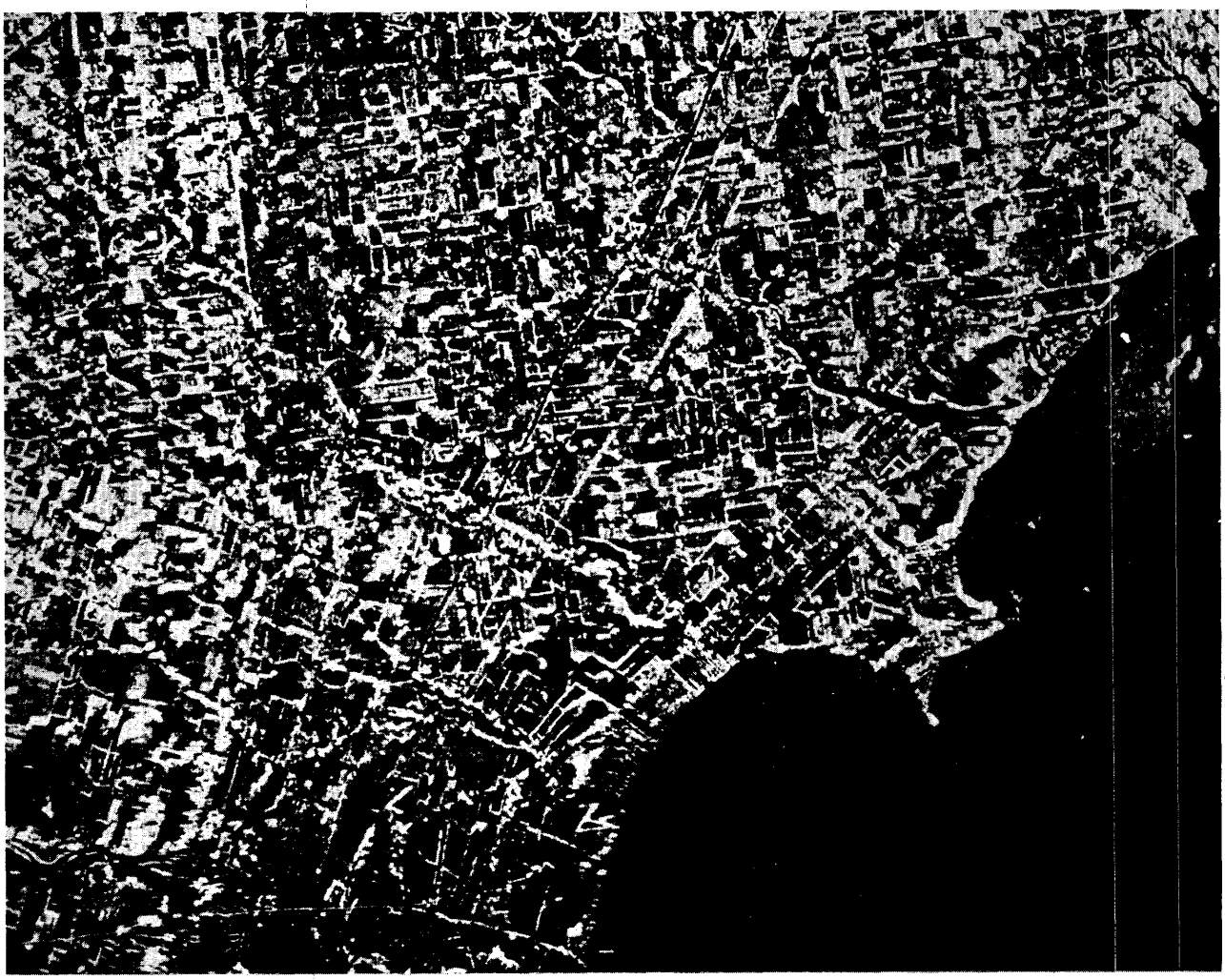


Figure 20. Fine resolution radar image, southeastern Michigan.

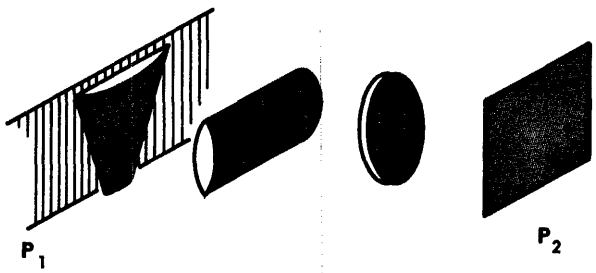


Figure 21. Optical processor using a conical lens.

source generates the information which is eventually delivered to the destination. The source may be any of a number of items. It may be oral as in voice communication; it may be a picture as in TV transmission; it may be a series of digits if digital data is transmitted; it may be an analog voltage.

The output of the source is sent to a coder. It is the object of the coder to perform such operations on the output of the source as will adapt these signals to the transmission channel. Often a transducer of some type is included in the coder such as a microphone in the case of oral sounds, a TV camera for scenes, so that the information is converted into electrical form. Additional operations are also performed. The most common of these is to impress the information onto some electromagnetic carrier. Many types of modulation exist, such as amplitude modulation, frequency modulation, and a number of pulse modulations.

These operations are primarily for the purpose of adapting the signal to the channel. However, in more sophisticated communications systems additional coding is employed for a variety of purposes. For example, such additional coding may be used

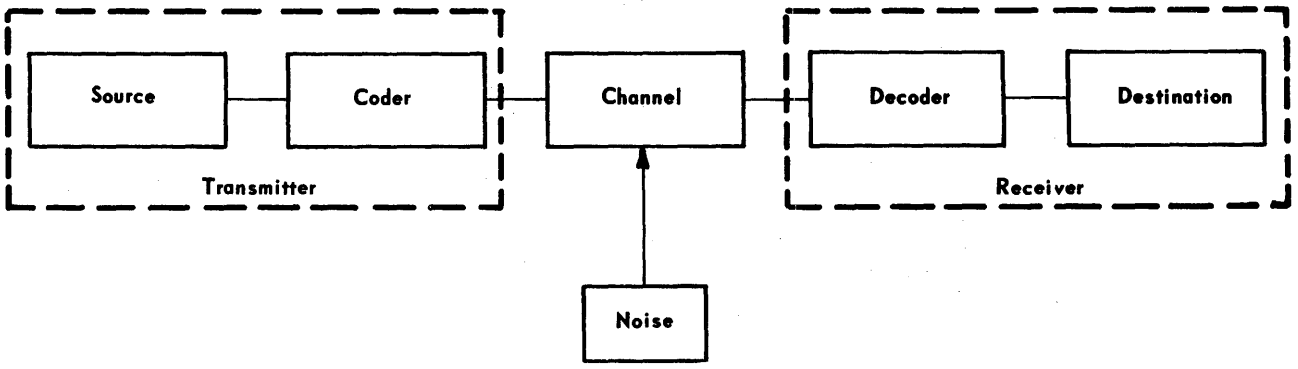


Figure 22. Essential components of a communications system.

for the purpose of signal-to-noise improvement, error correction, etc.

The output of the coder is fed to the communications channel. Two general types of channels are employed, in one case a wire system connects the transmitter to the receiver; in the other case (radio transmission) an electromagnetic wave is launched by way of a transmitting antenna and a receiving antenna associated with the receiver abstracts from this electromagnetic energy. A wide range of frequencies is available for either wire transmission or radio transmission. Recently, this spectrum has been extended to include visual frequencies so that modulation of light sources such as lasers may now be considered.

In either wire transmission or radio transmission, noise is added to the signals. Often, this is additive noise. However, multiplicative noise and multipath transmission are operations (generally undesirable) which may also be performed on signals as they pass through the channel.

The signals from the channel become the input to a receiver. It is the function of the receiver to detect the signal and to convert it into a form suitable for acceptance by the destination. If coding for signal-to-noise improvement or error correction have also been included in the coder, the operations necessary to achieve these improvements are also performed by the decoder.

Not shown in Figure 22 but sometimes used and necessary are a number of administrative equipments. These equipments perform such functions as automatic testing, automatic switching to a standby unit in case of failure, etc.

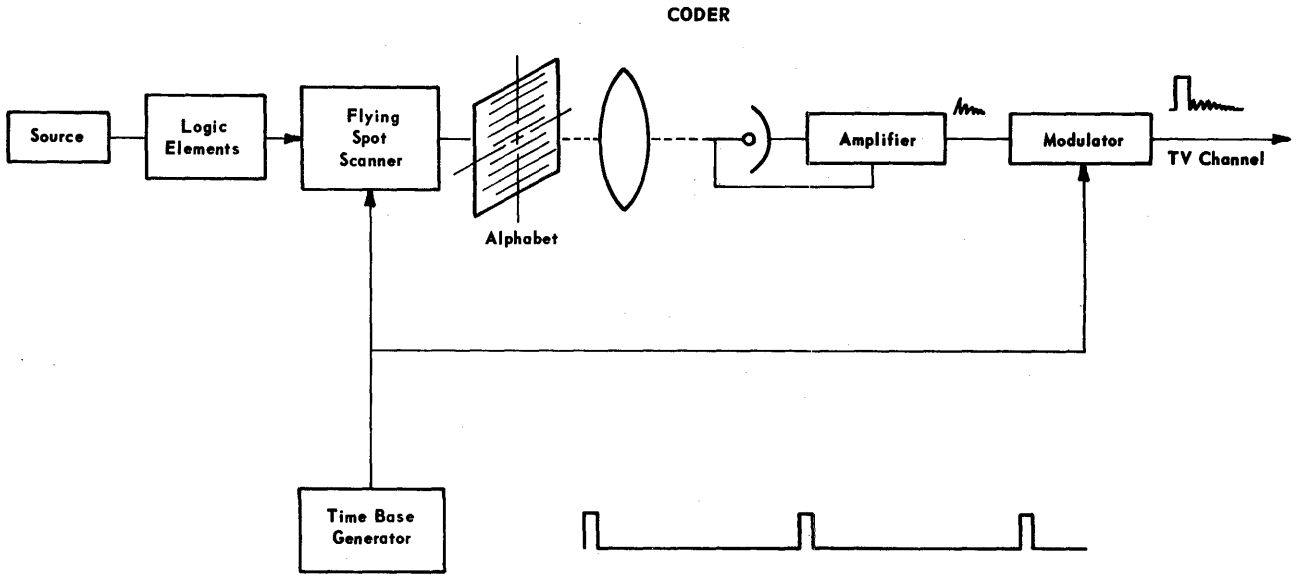


Figure 23. Optical coding.

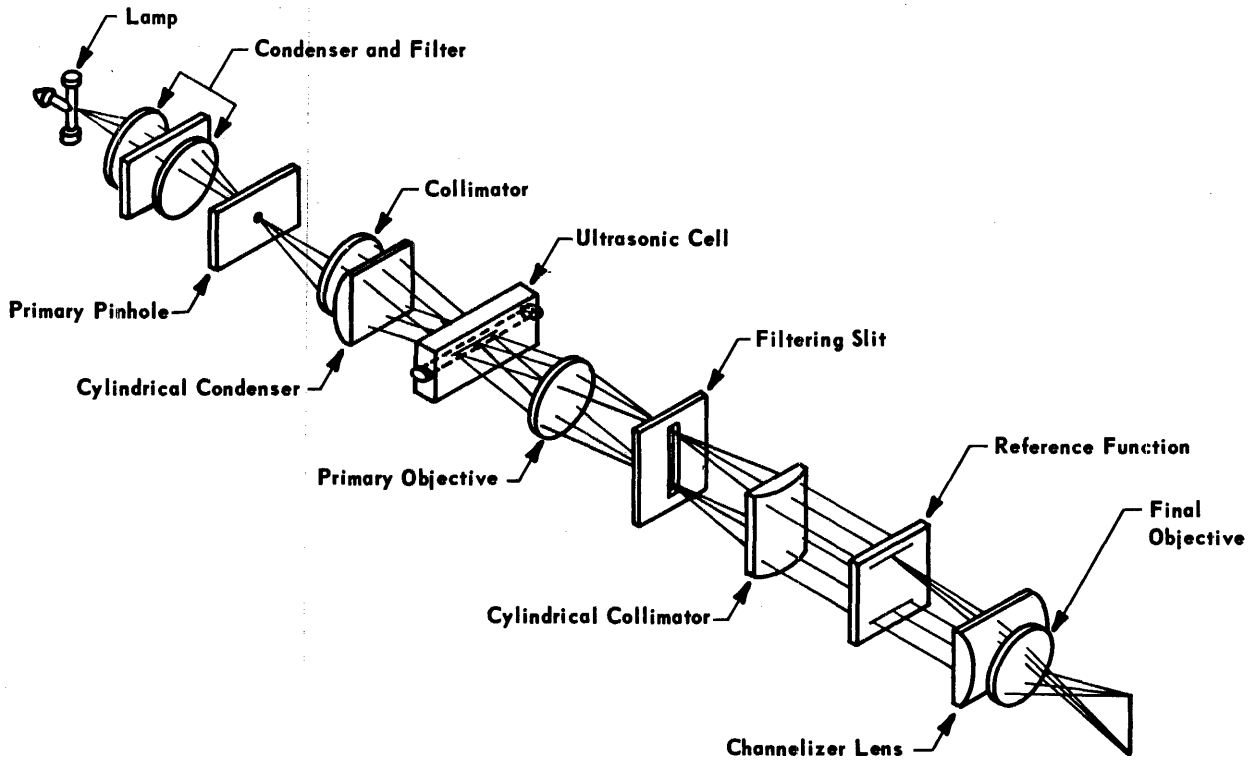


Figure 24. Optical decoding.

Block diagrams of optical coders and decoders are shown in Figs. 23 and 24. In these figures it is assumed that an alphabet (set of symbols) and rules for their selection have been previously determined using information theoretic considerations.

CONCLUDING REMARKS

In this paper optical computing and optical signal processing have been applied to a number of problems. The list of problems is by no means complete.

Optical signal processing must be considered as an extremely powerful and versatile technique.

ACKNOWLEDGMENTS

The author gratefully acknowledges the contributions of many of his colleagues and former colleagues to the content of this paper.

REFERENCES

1. L. J. Cutrona, E. N. Leith and L. J. Porcello, "Filtering Operations Using Coherent Optics," *Proceedings of the National Electronics Conference*, vol. 15, 1959.

2. ——, "Coherent Optical Data Processing," *IRE WESCON Convention Record,* Part 4, 1959, pp. 141–153, and *IRE Transactions on Automatic Control,* vol. AC-4, no. 2, pp. 137–149 (1959).

3. ——, "Data Processing by Optical Techniques," *Proceedings of the Third National Convention on Military Electronics,* 1959.

4. L. J. Cutrona et. al, "Optical Data Processing and Filtering Systems," *IRE Transactions on Information Theory,* June 1960, pp. 386–400.

5. E. N. Leith and J. Upatnieks, "Wavefront Reconstruction with Continuous Tone Objects," *J. of the Optical Society of America,* vol. 53, no. 12, pp. 1377–1381 (Dec. 1963).

6. B. A. Vander Lugt, "Signal Detection for Complex Spatial Filtering," *IEEE Transactions on Information Theory*, vol. IT-10, no. 2, pp. 139–145 (Apr. 1964).

7. B. Friedman, *Principles and Techniques of Applied Mathematics*, John Wiley and Sons, New York, 1956.

8. L. A. Zadeh, "A General Theory of Linear Signal Transmission Systems," *J. of Franklin Institute*, vol. 253, pp. 293–312 (Jan.–June 1952).

9. A. L. Ingalls, "Optical Simulation of Microwave Antennas," *IEEE Professional Technical*

Group on Antennas and Propagation International Symposium Program and Digest, Sept. 1964, pp. 203–208.

10. L. J. Cutrona et al, "A High Resolution Radar Combat-Surveillance System," *IRE Transactions on Military Electronics*, Apr. 1961, pp. 127–131.

11. L. J. Cutrona and G. O. Hall, "A Comparison of Techniques for Achieving Fine Azimuth Resolution," *IRE Transactions on Military Electronics*, vol. MIL-6, no. 2, pp. 119–121 (Apr. 1962).

12. L. J. Cutrona et al, "On The Application of Modern Optical Techniques to Radar Data Processing," presented at 9th Symposium of AGARD Avionics Panel on Opto-Electronic Components and Devices, Paris, Sept. 1965.

BIBLIOGRAPHY

Born, M., and E. Wolf, *Principles of Optics*, Pergamon Press, New York, 1959.

Cheatham, T. P., Jr., and A. Kohlenberg, "Analysis and Synthesis of Optical Processes," Boston University Physics Research Laboratories Technical Note 84, Part I (Mar. 1952).

——, "Optical Filters—Their Equivalence to and Differences from Electrical Networks," *IRE National Convention Record,* 1964, pp. 6–12.

Conductron Corporation Final Technical Report, "Coherent Light Investigation," Conductron No. D-5210-503-T246, Contract AF33(615)-2738 (Dec. 1965).

Cutrona, L. J., "Optical Computing Techniques," *IEEE Spectrum*, Oct. 1964, pp. 101–108.

——, "Recent Developments in Coherent Optical Technology," *Optical and Electro-Optical Information Processing*, MIT Press, Cambridge, Mass., 1965.

Elias, P., "Optics and Communication Theory," *J. of the Optical Society of America*, vol. 43, pp. 229–232 (Apr. 1953).

——, D. Grey and D. Robinson, "Fourier Treatment of Optical Processes," ibid, vol 42, pp. 127–134 (Feb. 1952).

O'Neill, E., "The Analysis and Synthesis of Linear Coherent and Incoherent Optical Systems," Boston University Physics Research Laboratories Technical Note No. 122 (Sept. 1955).

——, "Selected Topics in Optics and Communication Theory," ibid, no. 133 (Oct. 1957).

——, "Spatial Filtering in Optics," *IRE Transactions on Information Theory,* vol. IT-2, pp. 56–65 (June 1956).

Rhodes, J., "Analysis and Synthesis of Optical Images," *American Journal of Physics*, vol. 21, pp. 337–343 (Jan. 1953).

Woodward, P. M., *Probability and Information Theory, with Applications to Radar*, Pergamon Press, New York, 1960.

REQUIREMENTS FOR HOLOGRAM CONSTRUCTION

E. N. Leith, A. Kozma and J. Upatnieks
Institute of Science and Technology
The University of Michigan, Ann Arbor, Michigan

INTRODUCTION

Holography has in the last two years undergone a tremendous resurgence. The laser has contributed immeasurably to this, through the remarkable coherence of its light, which permits previously performed experiments to be carried out with relative ease, and in addition allows the performance of experiments which were hitherto hardly conceivable. Old experiments have been repeated with vastly improved results, and the new interest has given rise to new ideas with exciting promise.

This paper has a twofold purpose: first, to discuss the techniques for making good holograms and to describe how various factors degrade the process; second, to describe other forms of holography, in which phase is preserved by methods other than through phase modulation of an interferometrically produced grating.

ANALYSIS OF SOME STABILITY REQUIREMENTS

Ideal conditions for biographic recording include, among other items, completely stationary components, monochromaticity of the source, optical flatness of the recording surface, and linearity of the recording process. Failure to achieve these conditions causes in general some degradation of the hologram image.

Vibrations

Vibration of the reference beam mirror, the object, or the recording plate results in image degradation that, subject to a few rather plausible constraints, can be readily analyzed.

A hologram is made of the moving point P (Fig. 1). Let the mean position of the point lie at a distance z_0 from the hologram. Also, let the motion be resolved into two components, one along the line from object to recording plate, and the other normal to this direction. For simplicity, consider each of the motions separately.

The former case was treated by Powell and Stet-

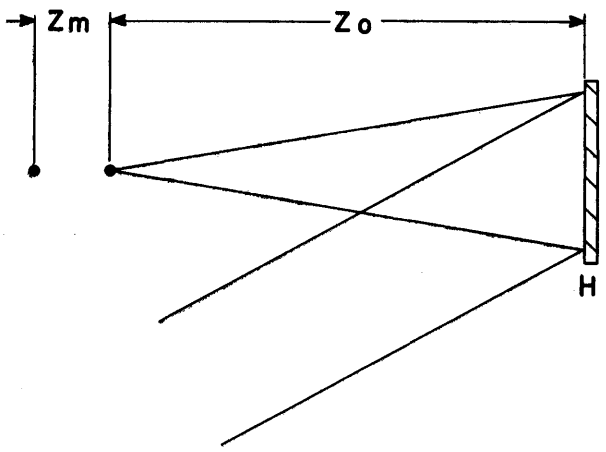


Figure 1. Hologram for a point object vibrating axially.

son[1] for the case of sinusoidal motion. We treat this case more generally by assuming a random motion with a prescribed probability density function for z_m, the deviation of the object point from its mean position.

The hologram of the moving point is a photographic record of the time-averaged light intensity exposing the recording plate, or, to the usual first-order approximation,

$$E(x) = \int_0^T \left| e^{i\alpha x} + k \exp -i \frac{2\pi}{\lambda} \left(\frac{x^2}{2z_0} + z_0 + z'_m(t) \right) \right|^2 dt \tag{1}$$

Here $z'_m = Kz_m$, where K is a scaling factor related to the relative amplitude of vibration ($K = 0$ for negligible motion of the point), $e^{i\alpha x}$ is the reference wave, and k is a constant. The virtual image term is

$$\int_0^T k \exp -i \left(\alpha x + \frac{\pi}{\lambda} \frac{x^2}{z_0 + Kz_m} + \frac{2\pi}{\lambda} Kz_m(t) \right) dt$$

$$= KT \exp -i \left(\alpha x + \frac{\pi}{\lambda} z_0 x^2 + \frac{2\pi}{\lambda} z_0 \right)$$

$$\frac{1}{T} \int_0^T \left[\exp -i \frac{2\pi}{\lambda} K \left(1 - \frac{x^2}{z_0} \right) z_m \right] dt \tag{2}$$

Assuming that the random process z_m is ergodic, the integral becomes

$$M_{zm} \left[\frac{2\pi}{\lambda} K \left(1 - \frac{x^2}{z_0^2} \right) \right] \tag{3}$$

where M_{zm} is the characteristic function of z_m evaluated at $\frac{2\pi}{\lambda} K \left(1 - \frac{x^2}{z_0^2} \right)$. Since x is usually much less than z_0, the argument of M_{zm} can be taken as $\frac{2\pi}{\lambda} K$. The reconstructed image is thus attenuated by the factor M_{zm}; $M_{zm}(0) = 1$ and $M_{zm}(v) < M_{zm}(0)$ for all v. Thus, M_{zm} is always an attenuating factor. In the case of a sinusoidal motion, the attenuating factor is a zero order Bessel function J_0. Since J_0 is oscillatory rather than monotonic, fringes (representing contours of constant vibrational amplitude) form on the reconstructed image.

If the probability density function is Gaussian, the characteristic function similarly is Gaussian; the attenuation factor is monotone decreasing and therefore no fringes are produced on the reconstructed image, only an attenuation proportional to K.

A similar analysis for a random motion x_m in the lateral direction results in a time-averaging factor

$$\frac{1}{T} \int_0^T \exp \left[-i \frac{\pi}{\lambda z_0} (x_m^2 - 2x_m x) \right] dt \tag{4}$$

Let the vibration be sinusoidal; $x_m = A \cos \omega_m t$. The integral becomes

$$\frac{1}{T} \exp \left[-i \frac{2\pi A^2}{4\lambda z_0} \right]$$

$$\int_0^T \exp \left\{ -i \frac{2\pi}{\lambda z_0} \left[\frac{A^2}{2} \cos^2 \omega_m t - Ax \cos \omega_m t \right] \right\} dt \tag{5}$$

A Bessel function expansion, after neglect of various small terms, leads to the hologram signal term being multiplied by $J_0 \left(\frac{2\pi Ax}{\lambda z_0} \right)$. This is similar to the case for axial vibration, except that the presence of the variable x within the argument is a complication which no longer makes this factor merely attenuative.

Another interpretation is applicable, however. The diffraction field of the point source, as formed at the hologram plate, has the phase distribution $\exp -i \left(\frac{\pi}{\lambda z_0} x^2 \right)$. The spatial frequency of the diffracted field is

$$\omega_x = \frac{d}{dx} \left(\frac{\pi}{\lambda z_0} x^2 \right) = \frac{2\pi x}{\lambda z_0} \tag{6}$$

Substitution into the Bessel function argument yields

$$J_0(A\omega_x) \tag{7}$$

The lateral motion thus has, approximately, the effect of a low pass filter which degrades the resolution of the hologram and also reduces the angular field over which the reconstructed image can be observed. Note that the filter operates on the object spectrum *before* its modulation onto the spatial carrier α.

Finally, consider the effect of lateral motion of the recording plate. In this case, a time-averaging factor

$$\frac{1}{T} \int_0^T \exp -i \frac{\pi}{\lambda z_0} \left[x_m^2 - \left(2x - \frac{\lambda z_0 \alpha}{\pi} \right) x_m \right] dt \tag{8}$$

is produced. An analysis similar to the one just given for the case of sinusoidal lateral vibration of the object shows that the process acts, as in the previous case, like a low pass filter except that this filter acts on the signal *after* its modulation onto a spatial carrier; the attenuation is thus more severe.

The Propagation Medium

The propagation medium in general is indifferent to whether the imagery is holographic or conventional; an important exception must be noted, however. If the medium is time-invariant and is available for the reconstruction process, then the medium serves, in the reconstruction process, as a compensator for the errors introduced in the hologram-making process. Thereby, high-quality imagery is supported in a medium which otherwise is incapable of supporting good imagery. This is accomplished by using the real image term in the reconstruction process: the real image term is conjugate to the original object; thus, phase irregularities are canceled when the real image term of the hologram is formed through the irregular medium.

This process has been demonstrated experimentally in two instances. In the first, a lens with severe spherical aberration was part of the medium,[2] in the second, several pieces of ground glass were inserted between object and hologram.[3] In each case, imagery was produced in which the deleterious effects of the medium were eliminated.

Coherence Requirements

The basic coherence requirements of the source can be stated quite simply, although the coherence problem when examined in depth becomes indeed abstruse and could itself well be the subject of a paper. For example, holograms can be made in completely incoherent light, as proposed originally by Mertz and Young, and discussed more recently by others.

In the case of a transparency, the object can be assumed to have no depth, and the coherency requirement is determined solely by the number of fringes required across the hologram, which is, in fact, just twice the number of resolution elements required across the object transparency. Indeed, techniques exist whereby holographic signals can be modulated onto diffraction grating images, and the source need not have the coherence required to produce such a raster of fringes. The minimum coherence length is that needed to produce the required dispersion of the object signal. For example, if a resolution cell on the object transparency is dispersed into a Fresnel zone plate image for impulse response having M fringes, coherence necessary for producing only M fringes is needed. The space-spatial bandwidth product of the signal (the spatial analog of the time-bandwidth or TW product) is M. Thus, the required coherence is related to the TW product of the system impulse response.

For the case of three-dimensional objects, the situation is more severe, since superimposed on the previous requirements is the requirement that the coherence length encompass the object depth. For an object of depth L, the second requirement is $\frac{\lambda^2}{\Delta\lambda} \geq L$, where $\Delta\lambda$ is the wavelength spread of the source.[4] If the source has spectrum $S(\omega)$, the reconstruction will have implanted on it, in the form of intensity variations as a function of depth, the autocorrelation function of the spectrum. This suggests the use of holographic methods for interferometric spectroscopy. Alternatively, proper selection of $S(\omega)$ offers a basis for giving precise information about the depth dimension of the object. For example, in the case of a two-frequency source, fringes that approximately represent the contours of constant depth are present in the reconstructed object.

FILM TRANSFER CHARACTERISTICS

In holography, photographic film plays the role of both a square-law detector and a spatial storage device or recorder. In an ideal recording, the intensity of the sum of two spatially modulated coherent waves is recorded so that a linear relation is obtained between the intensity and the specular amplitude transmission of the resulting recording. On subsequent readout using a coherent interrogating beam, the original incident wave amplitudes can be faithfully recreated from the stored data.

It is well known that if one uses photographic film in the linear region of the D-log E curve, a two-step recording process with a gamma product of two gives a linear relation.[5] However, the efficiency of this type of recording, in terms of the brightness of the reconstruction, is low and the process is difficult to accomplish with good fidelity at high spatial frequencies.

A more straightforward practice is to use a one-step process with exposure versus amplitude trans-

mission as the photographic transfer curve, where exposure is the product of the light intensity and the time of exposure. The amplitude transmission is defined as

$$T_a = \left(\frac{I'}{I_0'}\right)^{1/2} \tag{9}$$

where I' is the light intensity transmitted by the recording while I_0' is the total intensity of the coherent interrogating beam. Here we assume that the amplitude transmission is a real function which requires that the thickness variations of the film be negligible or that the film be immersed in an index matching fluid during readout.

Figure 2 shows the exposure-transmittance (E-T_a) curve for a high-constrast film used in wavefront

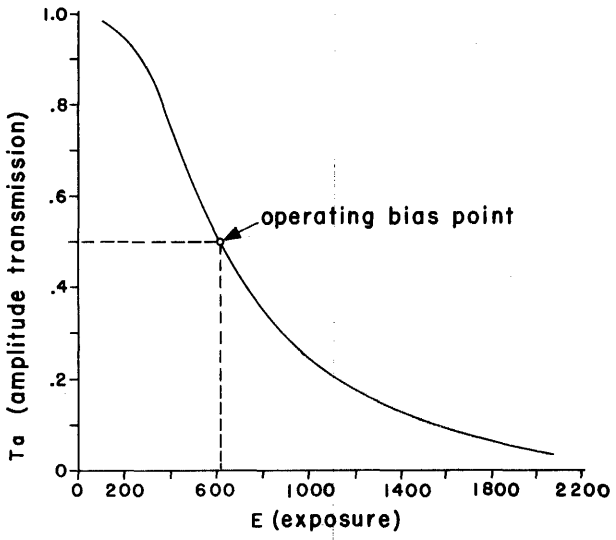


Figure 2. Log exposure vs density curve. Kodak 649 plate, exposed with HeNe laser light (6328), developed in D19 for 12 minutes at 68°F.

reconstruction while Fig. 3 shows the D-log E curve for the same film. One observes from these figures that there is a considerable region on the E-T_a curve over which approximate linearity is achieved. Also, this linear region does not coincide with the linear region of the D-log E curve, but corresponds with the toe and the lower part of the linear region of the D-log E curve.

From these observations it is clear that the D-log E curve is of little interest in wavefront reconstruction. Generally, a film which is optimally recorded for wavefront reconstruction will be underexposed by conventional standards. Also, it is apparent that attempts to achieve a specific γ of

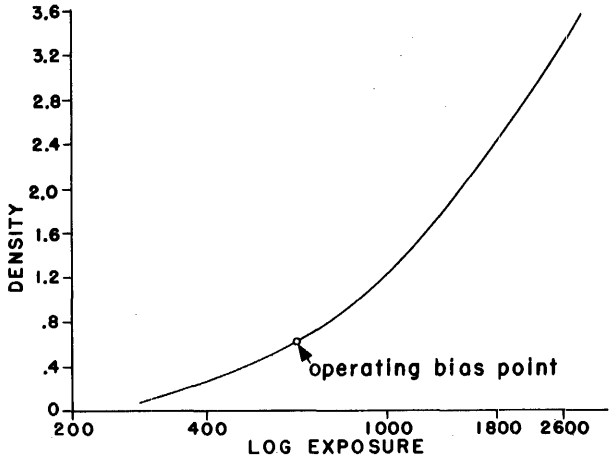


Figure 3. Exposure vs amplitude transmission curve. Kodak 649F plate, exposed with HeNe laser light (6328), developed in D19 for 12 minutes at 68°F.

the D-log E curve is of interest only as a means of relating conventional sensitometry to this application.

The linearity of the E-T_a curve mentioned above is the same type of inverse linearity achieved with a vacuum tube, that is, linearity about an operating bias point. This poses no problem since the intensity functions of interest contain a bias term which can be scaled to coincide with the operating point of the E-T_a curve. That the linearity is inverse is of little consequence for the type of wavefront reconstruction considered here since the spatial fluctuations of the intensity are modulated by a spatial carrier frequency. The net effect is that the fluctuating part of the recorded signal is changed in phase by 180° and in the readout the amplitude of the recovered signal is similarly reversed in phase. However, the recovered signal is ultimately sensed by an energy detector such as the eye or a photocell and the 180° phase change cannot be detected.

The reference wave U_0 and the signal wave U combine at the recording surface to produce the intensity

$$I = |U_0 + U|^2 \tag{10}$$

Letting $U_0 = k e^{i\alpha x}$, and $U = a e^{-i\phi}$, we have

$$I = k^2 + a^2 + ka\cos(\alpha x + \phi) \tag{11}$$

The exposure time is chosen so that the product of the constant part of Eq. (11) and the time is equal to E_0, the operating bias point on the T_a-E curve. The amplitude transmission of the film is then made up of a constant part, T_0, and a spatially fluctuating

part $g(x,y)$ and is given by

$$T_a(x,y) = T_0 + g(x,y) \tag{12}$$

If k is large compared to $a(x,y)$ then the exposure will be confined to the linear portion of the T_a-E curve and the transmission T_a is

$$T_a(x,y) = T_0 - \beta t\{a^2(x,y) + 2ka(x,y)\cos[\alpha x + \phi(x,y)]\} \tag{13}$$

where t is the time exposure and β is the slope of the T_a-E curve at the operating bias point E_0. Thus, we have achieved a linear transfer in the sense described above.

Effects of Film Resolution

To study the effects of the film resolution we assume that the amplitude of the reference wave is sufficiently larger than the signal so that we achieve a linear transfer. Under this condition, k is large compared to $a(x,y)$, $a^2(x,y)$ is small compared to $2ka(x,y)$, and Eq. (13) can be rewritten as

$$T_a(x,y) = T_0 - \beta t\{2ka(x,y)\cos[\alpha x + \phi(x,y)]\} \tag{14}$$

Since we are assuming linearity, we can take into account the film resolution by assigning an impulse response to the film and treating the film as a device in a linear system. The ideal transmission of the film, T_a, is modified by the response of the film and the result is given by

$$T_a(x,y)*H(x,y) \tag{15}$$

where $H(x,y)$ is the appropriately defined impulse response of the film and * denotes the convolution operation.

The physical effect of the frequency transfer characteristics of the film is most easily illustrated by considering film resolution as it affects Fraunhofer diffraction holograms. In this type of hologram the complex wave U is the Fourier transform or the Fraunhofer diffraction pattern of the object. Thus, $a(x,y)e^{-i\phi(x,y)} = \mathcal{J}[\tilde{a}(u,v)]$ where $\tilde{a}(u,v)$ is the object which is to be stored as a hologram and later reconstructed. The reconstruction, in this case, is performed by placing the photographic record in a collimated beam of coherent light and, with a lens, taking the Fourier transform of the record. The result, the light amplitude distribution at the back focal plane of the lens, is given by

$$R(u,v) = \mathcal{J}\{T_a(x,y)*H(x,y)\} \tag{16}$$

Using (14) this can also be written as

$$R(u,v) = \mathcal{J}\{[T_0 - Ke^{i[\alpha x+\phi(x,y)]} - Ke^{-i[\alpha x+\phi(x,y)]}]*H(x,y)\} \tag{17}$$

where $K = 2\beta tk$.

Performing the Fourier transform operation we obtain

$$R(u,v) = T_0\delta(u,v) - K\tilde{a}(u-\alpha,v)H(u,v) - K\tilde{a}(-u-\alpha,-v)\tilde{H}(u,v) \tag{18}$$

The second and the third terms of this expression are usual images, displaced by $\pm\alpha$, from the lens axis, which one expects; however, the images are attenuated by the modulation transfer function of the film.

This situation is quite different from the usual effect of the film transfer function, which is to cause a loss of resolution through attenuation of the higher spatial frequency components. Here, the resolution is unaltered by the transfer function curve; instead, the image is attenuated by an amount proportional to its angular displacement from the reference beam axis. This effect, of the film transfer function, is to cause a narrowing of the field over which the reconstruction can be viewed.

Effects of Nonlinear Recording

In order to increase the brightness of the reconstruction it is desirable to make the amplitude k, of the reference beam about the same magnitude as the amplitude of the signal, $a(x,y)$. However, if this is done, the recording is no longer linear. The effect of this nonlinear recording can be analyzed by using the characteristic function technique for handling nonlinear circuits.[6] Let the input to the nonlinearity be $E_1(x,y)$, the original exposure, $E(x,y)$, with the bias removed. Then the output from the nonlinearity is some nonlinear function of this input given by $-g(x,y)$. Then by expressing $-g(x,y)$ as an inverse Fourier transform we have

$$-g(x,y) = G[E_1(x,y)] = \frac{1}{2\pi}\int_{-\infty}^{\infty}\tilde{G}(\omega)e^{i\omega E_1(x,y)}d\omega \tag{19}$$

where G is the Fourier transform of the nonlinearity, G, and $E_1(x) = ta^2(x,y) + 2tka(x,y)\cos[\alpha x + \phi(x)]$. After substituting E_1 in Eq. (19) we have

$$-g(x,y) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\tilde{G}(\omega)e^{i\omega[ta^2(x,y)+2tka(x,y)\cos[\alpha x+\phi(x,y)]]}d\omega \tag{20}$$

Expanding part of the exponential term using the Jacobi-Anger formula[7] we have

$$-g(x,y) = \sum_{j=0}^{\infty} H_j[a(x,y)]\cos j[\alpha x + \phi(x,y)] \quad (21)$$

where

$$H_j[a(x,y)] = \frac{\epsilon_j}{2\pi}(i)^j \int_{-\infty}^{\infty} G(\omega)e^{i[ta^2]\omega}J_j[2tka\omega]d\omega \quad (22)$$

and where J_j is the Bessel function and $\epsilon_0 = 1$, $\epsilon_j = 2$ $(j = 1,2,\ldots)$. The resulting transmission of the record is

$$T_a(x,y) = T_0 - \left\{H_0[a(x,y)] + \sum_{j=1}^{\infty} H_j[a(x,y)]\cos j[\alpha x + \phi(x,y)]\right\} \quad (23)$$

Some general observations can be made about the recording process from Eqs. (13) and (23). For $j = 1$, we have preserved the phase of the original signal while distorting the amplitude. If the spatial carrier frequency is large compared with the spatial bandwidth of the signal, the various terms in the series (23) will not contain overlapping spatial frequencies even though the amplitude distortion tends to widen the bandwidth of each term. In cases where $a(x,y)$ is a slowly varying function or a constant, most of the information is contained in the phase part and the nonlinearity is of no importance.

As an example, consider the case where the object consists of many randomly distributed points of amplitude a_m and with random phase θ_m. For simplicity, we take these points along a line parallel to the hologram so that we can reduce the problem to one-dimensional notation. The total amplitude, due to the points at the hologram plane is

$$a(x)e^{-i\phi(x)} = \sum_m a_m e^{i[A(x-\beta m)^2+\theta_m]} \quad (24)$$

Here we have approximated the phase of each point at the hologram plane by a quadratic phase term. Then

$$a(x) = \left[\sum_m a_m^2 + 2\sum_{m,n} a_m a_n \cos[A(x-\beta_m)^2 - A(x-\beta_n)^2 + \theta_m - \theta_n]\right]^{1/2} \quad (25)$$

where $m = 1,2,\ldots, N-1$ and $n = (m+1),\ldots, N$ in the second sum. We can also write

$$\phi(x) = \tan^{-1}\frac{\sum_m a_m \sin[A(x-\beta_m)^2 + \theta_m]}{\sum_m a_m \cos[A(x-\beta_m)^2 + \theta_m]} \quad (26)$$

The intensity at the hologram using a plane reference wave, $ke^{i\alpha x}$, is given by

$$I(x) = k^2 + a^2(x) + 2ka(x)\cos[\alpha x + \phi(x)] \quad (27)$$

where

$$\cos[\alpha x + \phi(x)] = \frac{\sum_m a_m \sin(\alpha x + [A(x-\beta_m)^2 + \theta_m])}{a(x)} \quad (28)$$

Suppose we take for the T_a-E curve of the film a hard-limiter. This is an extreme case since film is never this nonlinear; however, this nonlinearity can be handled simply and it illustrates the effects.* For a hard-limiter $\widetilde{G}(\omega) = \frac{2L}{i\omega}$ where L defines the transmission limits of the film. Then substituting for $\widetilde{G}(\omega)$ in (22) we have for H_j

$$H_j[a(x)] = \frac{\epsilon_j L}{\pi}(i)^{j-1}\int_{-\infty}^{\infty}\frac{e^{i[ta^2]\omega}}{\omega}J_j[2tka\omega]d\omega \quad (29)$$

If we expand the integrand using a power-series expansion we have for H_j

$$H_j[a(x)] = \frac{\epsilon_j L}{\pi}\sum_{\mu=0}^{\infty}\frac{(i)^{\mu+j-1}[ta^2]^\mu}{\mu!}\cdot\int_{-\infty}^{\infty}\omega^{\mu-1}J_j[2tka\omega]\,d\omega \quad (30)$$

We assume that ta^2 is small enough when compared to $2tka$ so that only the first two terms of the power-series expansion need be used. We also assume that the carrier frequency, α, is sufficiently high that the coefficients H_j, for $j = 0$ and $j > 1$, can be dropped since the spatial frequencies will not overlap with the terms H_1. Then the coefficient of interest H_1 (that coefficient associated with the reconstructed images) is

$$H_1[a(x) = \frac{2L}{\pi} \quad (31)$$

Then, the part of the recording needed for reconstructing the points is

$$T_a'(x) = \frac{2L}{\pi}\cos[\alpha x + \phi(x)] \quad (32)$$

*Strictly speaking the hard-limiter or any other odd symmetric limiter cannot be a true model for the film since film does not have odd symmetry about the bias point. However, the even contributions from the film nonlinearity are usually much smaller than the odd ones. Other models which can be used are the error function limiter (see A. Kozma, "Photographic Recording of Spatially Modulated Coherent Light," *J. Opt. Soc. Am.* (in press) or a νth law device[6]).

or, using (25) and (28) we can write

$$T'_a(x) = -\frac{2L}{\pi} \cdot \frac{\sum_m a_m \sin(\alpha x + [A(x-\beta_m)^2 + \theta_m])}{\left[\sum_m a_m^2 + 2\sum_{m,n} [a_m a_n \cos[A(x-\beta_m)^2 - A(x-\beta_n)^2 + \theta_m - \theta_n]]\right]^{1/2}} \tag{33}$$

Since $\sin a = -\cos(a + \pi/2)$ we see that the numerator of Eq. (33) is the correct transmission function needed to accurately reconstruct the points. If the amplitudes of the points are all of about the same magnitude (the object is of moderate dynamic range), then the first sum of the denominator is significantly larger than the double sum. To see why this is true we can write the denominator of (33) as

$$\left[\sum a_m^2\right]^{1/2} \left[1 + \frac{2\sum_{m,n} a_m a_n \cos[A[2(\beta_m - \beta_n)x - \beta_m^2 - \beta_n^2] + \theta_m - \theta_n]}{\sum_m a_m^2}\right]^{1/2}$$

The second term in the brackets is a large sum of sinusoids with small amplitude and random phase. In the limit as the number of points considered become large this term will go as the square root of $\frac{2a_m a_n}{\Sigma a_m^2}$ which is $<< 1$. In this case, the denominator reduces to approximately a constant and the object points are reconstructed without distortion.

If one of the points, say a_1, in the object is very much greater than the other points combined then this is not the case. In this case the denominator can be expanded as

$$\frac{1}{a_1}\left[1 - \frac{1}{2}\sum_2 \frac{a_m^2}{a_1^2} - \sum_{m,n} \frac{a_m a_n}{a_1^2} \cos[A(x-\beta_m)^2 - A(x-\beta_n)^2 + \theta_m - \theta_n]\right] \tag{34}$$

A straightforward but tedious calculation shows that, in this case, the weaker object points will be suppressed relative to the point with amplitude a_1 and that false points will appear in the reconstructed image.

OTHER FORMS OF HOLOGRAPHY

The holographic process, ever since its invention by Gabor, has been beset with the problem of eliminating the twin image that arises because of incomplete recording of the phase relations in the wave field. The off-axis reference or spatial carrier method has proved an effective solution to the problem. With this method now convincingly demonstrated, we turn our attention to alternative methods. There are indeed various other possible approaches, several of which we will proceed to describe. All the ones described here involve lens systems and most involve various forms of spatial filtering.

The twin image term arises from the attempt to record a complex function using a phase-discarding method. The traditional objects used in holography have generally been transparencies, which normally are real functions; however, the Fresnel diffraction patterns are complex. The generation of a complex function from a real one is explained by considering that the space between the object and hologram planes treats the signal as an all-pass dispersive filter; the phase relations in the signal are thereby altered.

Accordingly, we might consider producing holograms through a process having a real impulse response, which would therefore retain the real nature of the original signal. Figure 4 shows such a hologram-producing system. The signal (a transparency) is introduced at P_1, and is imaged at P_3. A spatial filter having the property $F(\omega) = F^*(-\omega)$

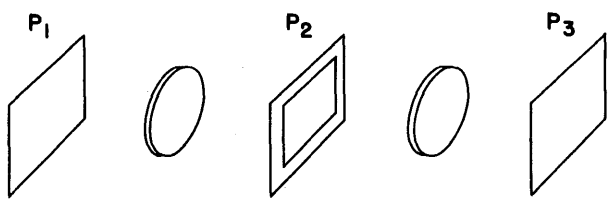


Figure 4. Optical system for generalized holograms.

is placed at P_2; this ensures a real signal at P_3. The signal is thus recorded with completeness without the need for a carrier-frequency reference beam or other phase-preserving technique. We record linearly in amplitude, producing

$$u_0 + u = u_0 + s^* f \tag{35}$$

where f is the impulse response of the spatial filter F, and s is the signal transparency, which is converted into the function $u = s^* f$.

In the reconstruction process, we image the hologram signal through a second spatial filter with impulse response $g(x)$, or transfer function $G(\omega)$. We require that, to within a constant,

$$SFG = S \tag{36}$$

or

$$F(\omega)G(\omega) = 1 \tag{37}$$

For convenience, we can let F and G be pure phase filters,

$$F(\omega) = e^{i\phi_1(\omega)} \tag{38}$$

$$G(\omega) = e^{i\phi_2(\omega)} \tag{39}$$

thus we require

$$e^{i(\phi_1+\phi_2)} = 1 \tag{40}$$

or

$$\phi_1(\omega) = -\phi_2(\omega) \tag{41}$$

It is often convenient to use the same filter both for making the hologram and reconstructing it. Two cases arise: The signal may be passed through the filter in the same manner in both cases, or the signal (or the filter) may be reversed. In the former case, we require for reconstruction that

$$|F|^2 = 1 \tag{42}$$

from which we obtain

$$F = e^{i\epsilon(\omega)\pi} \tag{43}$$

where $\epsilon(\omega)$ is a function that assumes the value 0 or 1 and satisfies the condition $\epsilon(-\omega) = -\epsilon(\omega)$.

Alternatively, if the reconstruction is to be made with the filter of the signal reversed, the reconstruction condition becomes

$$F(\omega)F(-\omega) = 1 \tag{44}$$

or since $f(\omega) = F^*(-\omega)$, we have

$$F(\omega)F^*(\omega) = |F|^2 = 1 \tag{45}$$

from which we derive

$$F(\omega) = e^{i\phi(\omega)} \tag{46}$$

with

$$\phi(-\omega) = -\phi(\omega) \tag{47}$$

Thus, if we want a filter that produces a real output when the input is real, and if we want the function $1/F$ to be the filter reverse $F(-\omega)$, and if the filter is to be purely phase, then we require the phase factor to be antisymmetric, $\phi(-\omega) = \phi(\omega)$. An example of this case is a lens, one half of which is concave and the other convex, as shown in Fig. 5. To a

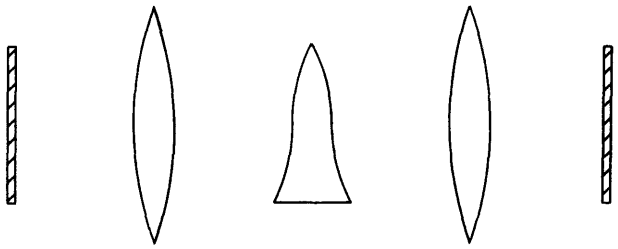


Figure 5. The antisymmetrical lens used in an optical system for generalized holograms.

first-order approximation, we have $F(\omega) = e^{iK\omega^2}$ for $\omega > 0$, and $F(\omega) = e^{-iK\omega^2}$ for $\omega > 0$. Thus we may write

$$F(\omega) = e^{iK|\omega|\omega} \tag{48}$$

Clearly, the fabrication of such a lens is not practicable, since both halves must be matched to within a fraction of a fringe. One might consider using a symmetrical Fresnel zone plate; it obviously satisfies the condition $F(\omega) = F^*(-\omega)$. Unfortunately, it fails to satisfy the requirement $|F|$ = constant. There is a feasible solution, however. With a carrier-frequency method used in making holograms, we may substitute for the required function $e^{iK|\omega|\omega}$ its real function equivalent $1 + \cos(\alpha\omega + K|\omega|\omega)$; the first-order diffracted wave of this function, written on a transparency, yields the desired filter function. A straightforward analysis shows that the required function can be produced, for a cylindrical Fresnel zone plate (focal power in one-dimension only) by removing a portion from the center of the zone plate and joining the two outer portions.

Another method of producing a real function diffraction pattern involves the use of two identical objects, positioned so that their summation produces a diffraction pattern with constant phase (i.e.,

the diffraction pattern is real). A well-known method of accomplishing this is to use a lens, which takes the Fourier transform of the object. By making the signal symmetrical, its Fourier transform will be real. Placing a strong point source on axis in the object plane then causes the Fourier transform to be positive as well, and thus recordable without the loss of phase. This technique is similar to methods used in crystal analyses by X-ray diffraction. A positive real Fresnel diffraction pattern can be generated by displacing the object transparencies in the axial dimension, but maintaining symmetry about the point source which supplies the bias.

REFERENCES

1. R. Powell and K. Stetson, *J. Opt. Soc. Am.*, vol. 55, p. 1593 (1965).

2. E. Leith, J. Upatnieks, and A. Vander Lugt, ibid, p. 595.

3. —— and ——, *SPIE Journal,* vol. 4, p. 3 (1965).

4. M. Born and E. Wolf, *Principles of Optics,* Pergamon Press, Bath, England, 1959.

5. D. Gabor, *Nature,* vol. 161, p. 777 (1948); *Proc. Roy. Soc.* (London), vol. A197, p. 454 (1949); ibid, vol. B64, p. 449 (1951).

6. W. B. Davenport and W. L. Root, *An Introduction to the Theory of Random Signals and Noise,* McGraw-Hill, New York, 1958, Chap. 13.

7. W. Magus and F. Oberhettinger, *Functions of Mathematical Physics,* Chelsea, New York, 1949, p. 18.

BIBLIOGRAPHY

Baez, A. V., *J. Opt. Soc. Am.*, vol. 42, p. 756 (1952).

Duffieux, P. M., *L'Intégrale de Fourier et ses Application à l'Optique,* chez l'Auteur, Université de Besançon, Besançon, France (1946).

El-Sum, H. M. A., "Reconstructed Wavefront Microscopy," PhD thesis, Stanford University, Nov. 1952 (available from University Micro-

El-Sum, H. M. A., and A. V. Baez, *Phys. Rev.*, vol. 99, p. 624 (1955).

Haine, M. E., and J. Dyson, *Nature,* vol. 166, p. 315 (1950).

Haine, M. E., and T. Mulvey, *J. Opt. Soc. Am.*, vol. 42, p. 763 (1952).

Kirkpatrick, P., and H. M. A. El-Sum, ibid, vol. 46, p. 825 (1956).

Leith, E., and J. Upatnieks, ibid, vol. 53, p. 1377 (1963); vol. 54, p. 1295 (1964).

Lohmann, A., *Opt. Acta* (Paris), vol. 3, p. 19 (1956).

Rogers, G. L., *Proc. Roy. Soc.* (Edinburgh) vol. A63, p. 193 and p. 313 (1952); vol. A64, p. 209 (1956).

Papers presented at Washington Conference on Electron Microscopy, National Bureau of Standards, Nov. 1951.

APPLICATION OF COHERENT OPTICAL TRANSDUCERS TO OPTICAL REAL-TIME INFORMATION PROCESSING*

Dean B. Anderson
Autonetics, A Division of North American Aviation, Inc.
Anaheim, California

INTRODUCTION

Interest in optical information processing stems from the never-ending effort to increase data handling capacity and to improve the interface with human senses. The manipulation of two-dimensional data in an image format and processing in a parallel organized integral transform distinguishes the optical analog computer from its counterpart—the electronic digital computer processing one-dimensional data in a sequential manner.[1] The optical analog computer is admirably suited to performing linear operations such as matrix products, Fourier transform integration, and related correlations and convolutions. The photographic plate is usually employed as the input, memory, control function, and output to demonstrate optical information processing concepts.

The high signal sensitivity, large data storage, and wide spatial bandwidth are the attractive characteristics of a photographic emulsion. However, the time required to develop the latent image to obtain access to data is both dismally slow and cumbersome in comparison to electronic means. The chemical amplification encumbrance has led to consideration of thermoplastic and ultrasonic delay line recording as an alternative with the attendant compromise of serial data input. If optical analog computers are to compete with the developing electronic art, it is essential that the access time to current data be reduced to a small fraction of a second without compromise of data capacity. Furthermore, as real-time optical processing is achieved, a requirement for an optical adaptive capability will also arise.

Optical information processing systems require the functions of amplification, modulation, and detection to be performed throughout the signal spatial field. These functions can be effectively synthesized by an array of coherent optical transducers extending across the signal spatial field provided that the spacing between the individual transducers and their size are comparable to the radiation wavelength. A quasi-microwave approach to coherent infrared transducers and their arrays using microphotolithographic techniques is delineated.

OPTICAL OPERATIONS

The application of communication theory to physical optics has provided the foundations of optical information processing.[2,3] The basic image transformation operation is illustrated in Fig. 1. The spatial signal is usually introduced at the object plane by coherent plane wave illumination of a photographic transparency. The resulting diffraction pattern in the Fraunhofer region is the Fourier transform of the spatial signal. The use of a lens (focal length F) permits the scaling of the Fraun-

*Work supported in part by the Office of Naval Research

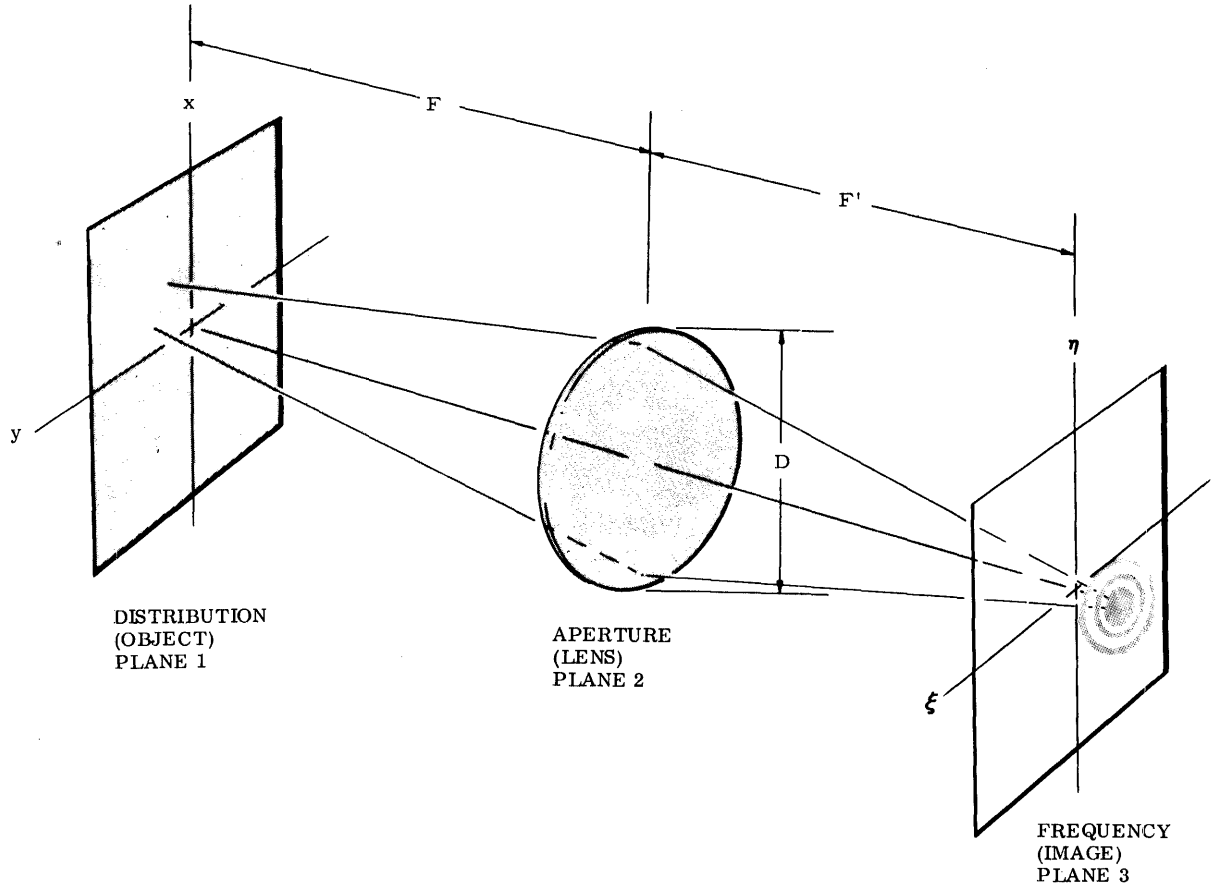


Figure 1. Basic image transformation operation.

hofer diffraction region to a more convenient location at the focus of the lens. The Fourier transform of the spatial signal is formed in the lens image plane as a spatial frequency spectrum. For some applications, this spectral analysis is the desired output.

If various shaped aperture stops or blocks are inserted into the frequency plane, it follows that the output signal distribution from a second transform lens system will be altered by the filter impulse response. In fact, a lens acts as a low-pass filter. Matched filter enhancement of the signal-to-noise ratio can be demonstrated by inserting the complex conjugate of the signal transform into the frequency plane. The realization of a complex-valued spatial function in a photographic transparency is extremely difficult due to the required phase response.

An interferometric approach has been introduced to record complex functions on photographic plates.[4,5] The addition of a reference wavefront as a carrier to the spatial signal produces in a photographic plate one component resembling a diffraction grating. The grating frequency is proportional to the angular separation between signal and reference beams. An interferometric recorded spatial filter of the symbol † is shown in Fig. 2.

The operations of convolution and correlation can be realized by further compounding the system with additional image transformations. The results

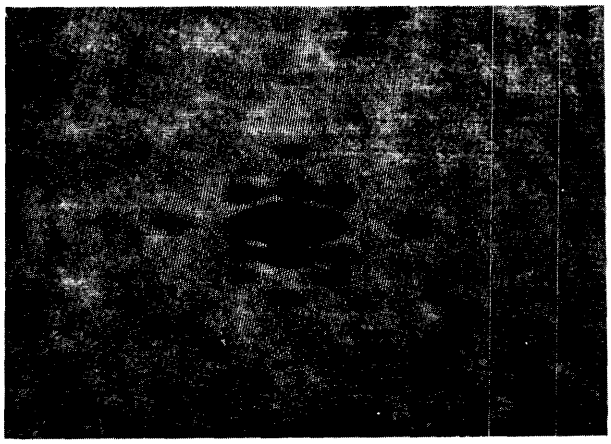


Figure 2. Interferometric spatial filter.

from use of the spatial filter, Fig. 2, for pattern recognition is shown in Fig. 3. The diffraction grating in the spatial filter has provided a convenient means to separate, by orders, the various components of the matched filter process. Using the symbol † as the input, three components are observed in the output: 1) the input convolved with the filter impulse response, 2) a crude image reconstruction, and 3) the input cross-correlation with the filter conjugate impulse response—indicative of recognition.

Gabor's[6] holography has been rekindled anew with introduction of interoferometric methods because of the vivid three-dimension reconstruction of the scene with parallax.[7] A hologram is a spatial filter for a particular scene when coupled with the transform in a human eye. The hologram, when illuminated by a delta-function distribution (point source), which has a uniform spectrum, allows passage of only those spatial frequencies which will form the reconstructed wavefront and thus the desired image. For optical pattern recognition, the holography process is inverted so that the unknown object serving as a source illuminates a spatial filter producing a single point image (correlation point). Figure 3 also shows the symbol reconstruction using the spatial filter in Fig. 2 as a hologram. Both the real and virtual images of the symbol are apparent.

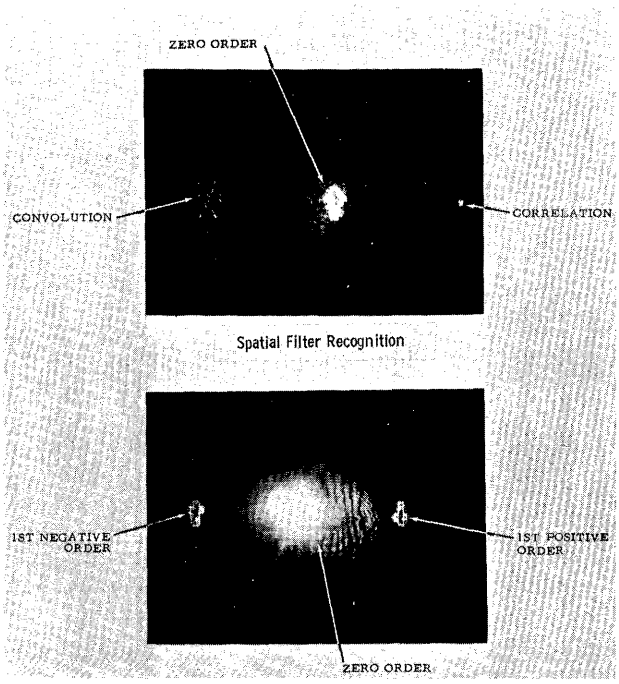


Figure 3. Spatial filter operations.

Current optical pattern recognition techniques require a rigorous matching of the spatial filter and the input. Factors such as position, scale factor, orientation aspect, and inversion severely compromise the recognition fidelity. Some problems can be solved by restricting the list of symbols to a particular style.

Consider the problems of recognition and tracking of hurricanes in cloud patterns recorded by satellites. Pattern characteristics peculiar to the hurricane spiral and invariant with observation conditions are obviously obscured by noise. Therefore, during the learning phase it will be necessary to assemble a large catalog of spatial filters from successive cross-correlations of known hurricane spectra. From the catalog of spatial filters, a characteristic set are selected and the decision criteria established. The correlation of unknown cloud patterns with the stored filters provides a basis of comparison in the recognition process. Of the utmost importance is the optimization of the recognition process by including an adaptive feature; that is, a measurement of the correlation point amplitude ratio with respect to the side lobe skirt level and distribution to alter the filters within the classification set. This means that techniques must be developed to alter the spatial filters dynamically so that they can change their functional properties with time in accordance with new data.

SPATIAL MODULATOR

The requirements of a spatial modulator to dynamically implement optical spatial filters are briefly outlined. A spatial modulator is schematically illustrated as an array in Fig. 4. The functions of amplification, modulation, and detection must be performed throughout the aperture in a linear fashion with respect to the control signal, while the quantized elements of the array must operate independently without crosstalk. A spatial modulator may operate in either a transmission or reflection mode and may alter either the amplitude or phase characteristic.

Factors which influence the quantization of the spatial modulator into an array are available from antenna theory. Some of these are:

1. The half period factor (T) of the highest spatial frequency component must be greater than the array period which must be greater than one-half the wavelength. ($T/2 > s > \lambda/2$)

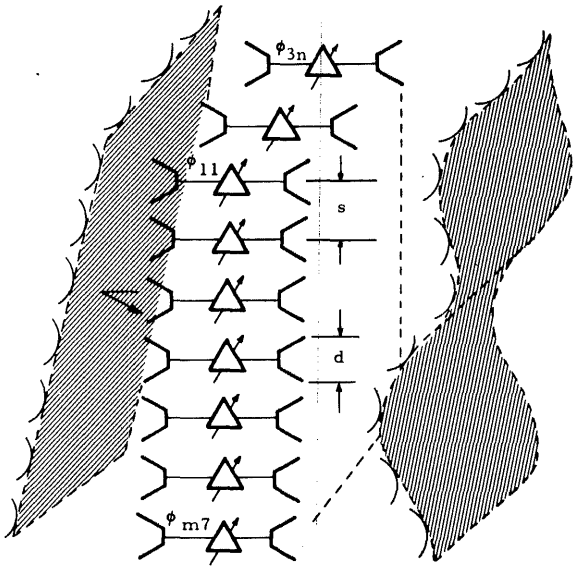


Figure 4. Spatial modulator.

2. The array factor giving rise to a grating lobe period must exceed the angular diameter of the lens (see Fig. 1) for an unambiguous operation. ($\lambda/s \geq D/F$)
3. The beamwidth of a single element must exceed the angular diameter of the lens to be included within the integration. ($2\lambda/d \geq D/F$)
4. Random phase and amplitude errors should be minimized to preserve the dynamic range in the optical system.

A wide variety of bulk interaction phenomena exist to alter the optical properties of materials by application of electric and magnetic fields or by mechanical stress. Most of these interaction phenomena are weak—even in high fields. It is also difficult to induce from the outside the spatial perturbations. Therefore, a bulk spatial modulator without quantization and with a significant spatial bandwidth is impractical. However, heteroepitaxial semiconductor, ferroelectric and ferromagnetic[8] materials imbedded with conductor arrays are a promising approach.

Current integrated circuit technology provides an approach to quantize the array. Moss[9] has analyzed various methods of modulating infrared radiation using semiconductor materials which will respond rapidly. A quasi-microwave approach using semiconductor diode junctions which may be assembled into arrays for modulation and amplification will be discussed in the following sections. The depletion layer in diodes is considered as an optical transmission line where its length is controlled by the applied voltage. Current integrated circuit technology is now becoming interconnection-limited. The interconnection limitation also prevails in an optical spatial modulator. However, there are several means to circumvent the problem. A variety of photoeffects occur in semiconductor material and junctions when the illumination radiation wavelength is shorter than the band edge. Of particular interest here is the photovoltaic effect in a junction. The electron-hole pair created by absorption near the junction is separated by the internal field at the junction and thus alters the optical properties of the depletion layer for radiation longer than the band edge. Through this process, it should be possible to control the optical spatial modulator diode array by illumination of the array with a second beam containing the spatial control signal. A similar control of a parametric amplifier array is also possible through pump excitation. The photodetector array art is currently well established and need not be discussed.

PASSIVE INFRARED WAVEGUIDE

Implementation of an optical spatial transducer requires interconnection by a waveguide which preserves the state of polarization and mode of propagation. Optical dielectric waveguides of circular and planar cross section have been demonstrated and reported in the literature.[10-12] Control of a specific mode of propagation is best accomplished in transmission line with dimensions comparable to the wavelength. The binding of the electromagnetic field to the dielectric structure depends upon the relative index of refraction of the transmission medium being greater than the surrounding environment. An active and a passive infrared waveguide structure is illustrated in Fig. 5. The rectangular dielectric image line supported on a reflecting surface is ideally suited for coupling and integration with optically active devices such as semiconductor junctions.[13]

As an example of dielectric image line waveguide, Fig. 6 shows a "rat race" hybrid junction formed photolithographically. The dielectric is thermally grown silicon dioxide on a highly doped, polished silicon substrate. The dielectric cross section is 0.6 × 2.0 microns and thus is useful in the near infrared region.

The results of an experiment demonstrating the propagation of a single, lowest-order mode through

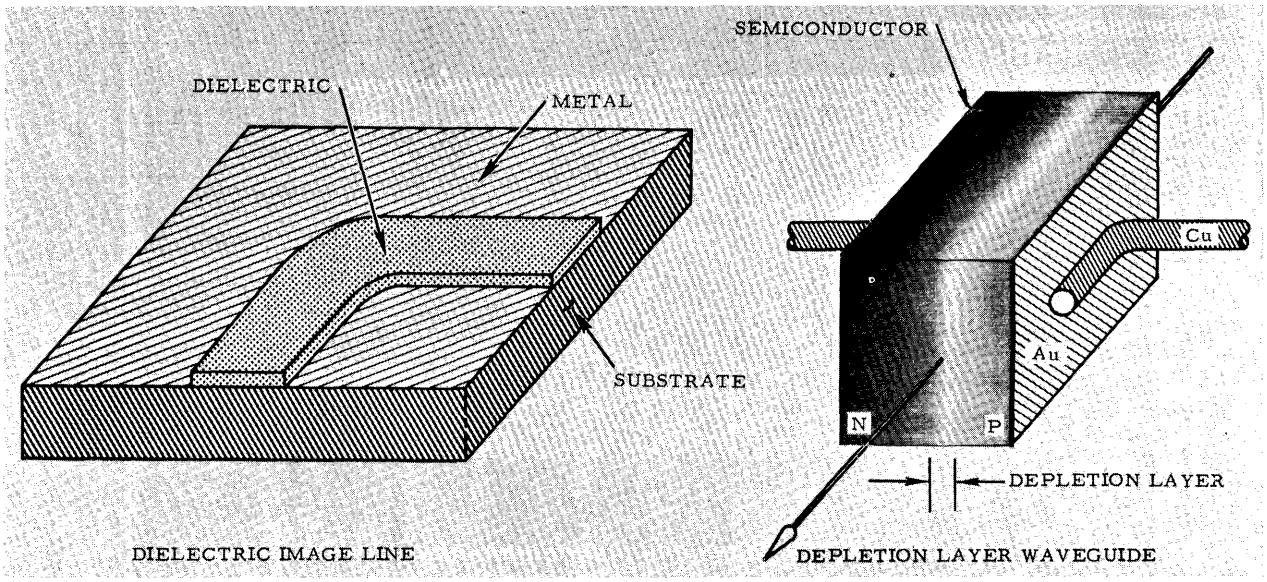


Figure 5. Passive and active infrared waveguide structures.

a dielectric image line waveguide are shown in Fig. 7. A cleaved cross section of dielectric image line (0.3 × 1.2 micron cross section) and substrate was used for the photomicrographs. A Lloyd's mirror[14] demonstration of interference fringes from the image line waveguide substrate is shown in the upper figure. The black region is the silicon substrate. The visibility of the fringes immediately adjacent to the substrate surface disappears because the light source is polychromatic and temporally incoherent. The presence of the waveguide is clearly apparent. The discontinuity in the fringes immediately adjacent to the substrate is due to the reflection from the outer surface of the dielectric image line waveguide. Note the subdued interference normal to the substrate resembling a Fraunhofer pattern. This is due to reflection from the dielectric image line waveguide at greater depth. The Lloyd's mirror illumination has been removed and replaced by a laser beam focused on the dielectric image waveguide in the lower figure. The transmission of a single mode through the waveguide is apparent. The circular radiation distribution is due to the dipolar field and the offset is due to the image phenomena. The substrate surface in the lower figure is indicated by some scattering at the focused input coupling.

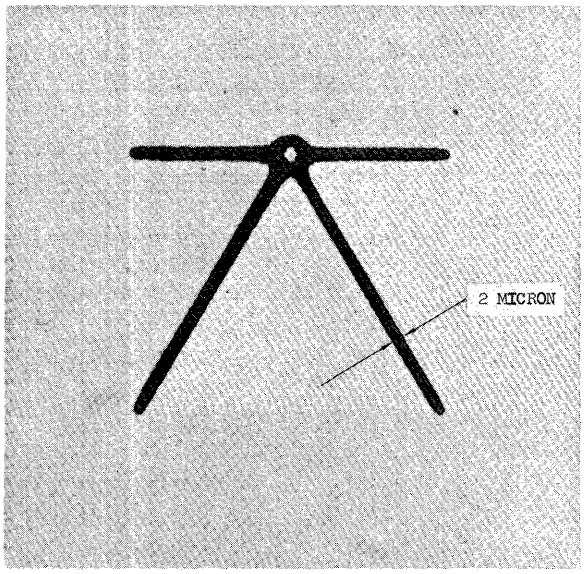


Figure 6. Dielectric image line waveguide—"rat race" hybrid junction.

ACTIVE INFRARED WAVEGUIDE

The depletion layer in a reverse-biased semiconductor junction diode will also guide optical waves.[15-19] The free carriers in both n and p give rise to a solid state plasma having a slightly reduced index of refraction which confines light propagation in the depletion layer. In the back-biased planar diode, the thickness of the depletion layer sandwiched between n and p regions is controlled by the applied bias field and may be comparable to infrared wavelengths. A depletion layer waveguide is a useful active device because it provides an electronic means to control an optical wave phase velocity.

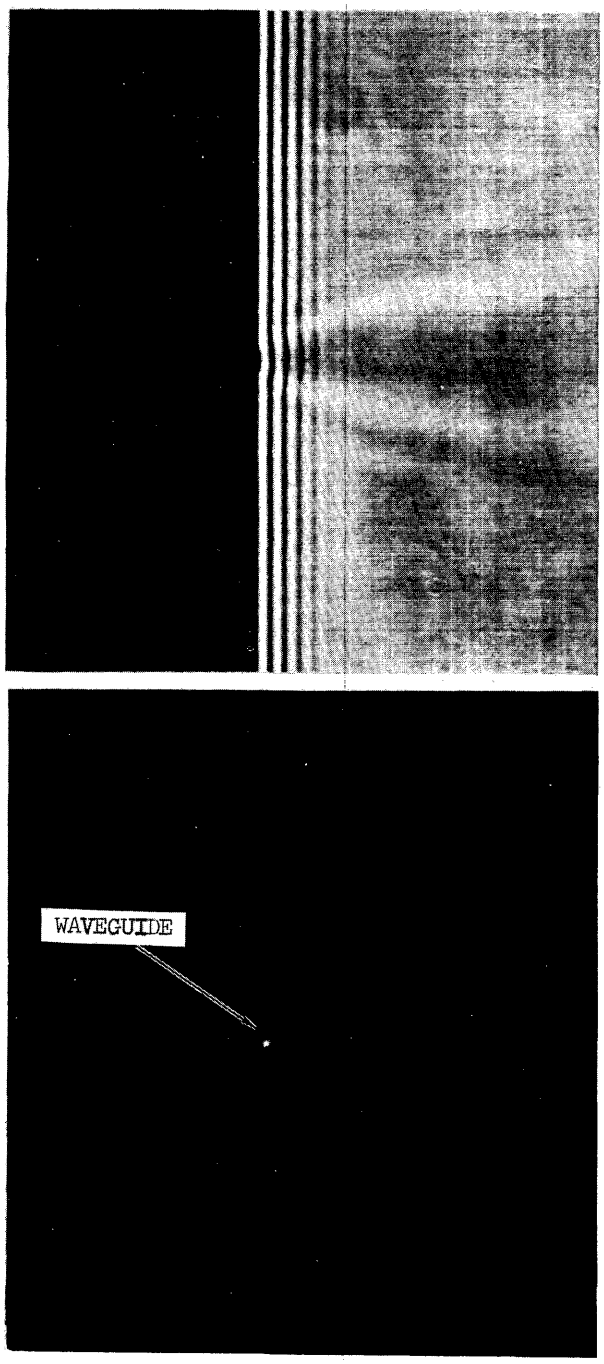


Figure 7. Results of Lloyd's mirror experiment showing transmission through dielectric image line waveguide.

Obviously, a depletion layer used as a dielectric waveguide or resonator must be used in the spectral region where it is transparent. Most semiconductors are opaque in the visible region but are transparent in an adjacent infrared region. The intrinsic absorption is due to excitation of electrons across the forbidden energy gap. Lattice absorption and Reststrahlen bands exist in the far infrared region. The intervening region is comparatively transparent except for impurity and free carrier absorption. The differential index of refraction between the various layers in a depletion layer is comparatively small (10^{-2} to 10^{-4}) so that a wave field is weakly bound to the junction. Careful attention must be given to the selection of doping elements, their concentration and gradient, and the host lattice defects because fields surrounding the junction will cause absorption, diffraction, and scattering losses.

The work of Nelson and Reinhart[20] is illustrated in Fig. 8 which shows the transmission through a GaP diode junction. Although their work exploited the linear electro-optic effect for modulation, this photograph vividly illustrates the changing dimensions of the depletion layer waveguide as a function of the applied bias field.

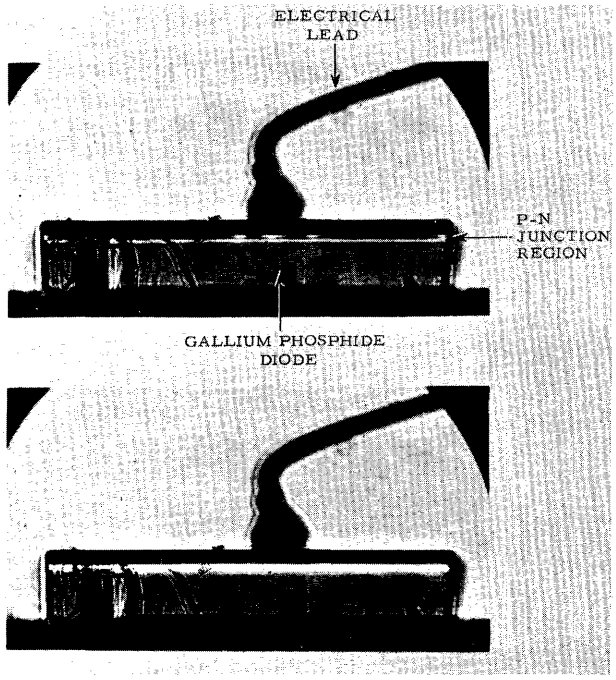


Figure 8. Transmission through GaP diode junction depletion layer waveguide. (By permission of Bell Telephone Laboratories. See Ref. 20.)

There are various other mechanisms to control the phase velocity in a depletion layer waveguide besides that of geometry. One is the electro-optic effect as above which leads to birefringence in the index of refraction. Another is the change of the index of refraction below the band edge of a semiconductor due to the dispersion associated with the

Franz-Keldysh effect. The Franz-Kelkysh effect is a shift of the band edge to a longer wavelength because of the application of an intense electric field. The various considerations which enter into the use of depletion layer waveguide as an active optical circuit element suggest a III-V semiconductor material and a wavelength just short of the band edge.

A depletion layer boundary may also be used in the reflection mode as a movable mirror. A photo field effect transistor shown in the photomicrograph of Fig. 9 has been used for a demonstration experiment. An alloy gate contact and an etched channel

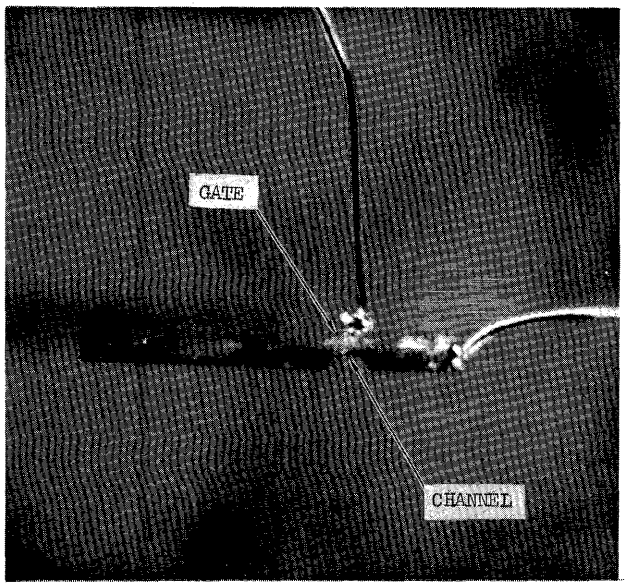


Figure 9. Photo field effect transistor.

were formed in the silicon slice between source and drain. The use of a preferential etch, the alloy junction technique, and oriented silicon has led to a high degree of parallelism between the etched channel surface and the depletion layer boundary. The pinch-off characteristic curve is indicative of the optical quality in the boundary planes and shows that the depletion layer can be driven into planar contact with the channel surface. Experiments show that the position of the reflected beam from the etched channel front surface and the depletion layer is controllable by the applied gate bias. Illumination of the channel also provided a control of the beam position.

PARAMETRIC INTERACTION

To provide optical amplification and a means to control an operation by an optical logic field, the characteristics and feasibility of infrared parametric interactions are discussed. Parametric interactions involve the mixing of one or more signal frequencies with an intense source called a "pump" producing sum and difference combinations. The Manley and Rowe[21] energy relations show that there are two basic amplification mechanisms distinguished by the manner that the signal and pump frequencies are allowed to combine. Further discussion will be restricted to the difference combination which creates an effective negative absorption and results in a signal spectrum inversion.

Parametric interactions depend upon a reactive nonlinear phenomena. The nonlinear capacity of a varactor diode is due to the change of the depletion layer thickness. In the infrared region the carrier inertia prevents a similar response; however, the index of refraction of some materials have a nonlinear behavior in intense light. The polarization nonlinear susceptibility of III-V semiconductor compounds is several orders of magnitude larger than the piezoelectric crystals which have been used for optical parametric amplification.

Recently, positive gain has been realized in difference frequency parametric interactions by Wang and Racette[22] using $NH_4H_2PO_4$ and by Giordmaine and Miller[23] using $LiNbO_3$. They used a Q-spoiled solid state laser to pump a nonlinear cyrstal under index matching to efficiently produce a second harmonic. After filtering and collimating the beam, the harmonic was applied as a pump to a second nonlinear crystal to obtain quasi-degenerate operation. Birefringence in the crystal was used to balance the index of refraction dispersion to obtain matched phase velocities in their traveling wave circuit.

A quasi-microwave approach[24] which differs significantly from the above investigations using piezoelectric crystals is currently under way. A planar section of depletion layer waveguide is used as a multimode resonator tuned to the signal idler and pump frequencies. The dimensions have been selected sufficiently small to prevent other undesirable resonances. The requirements for index matching in a traveling wave structure are removed by the use of low-order modes within the resonator wherein the fields and geometry defining the boundary conditions establish resonances which are algebraically related to the pump frequency. Within the constraints of known material technology, the available power and emission lines from continuous wave gas lasers, and for an acceptably high nonlinear susceptibility, gallium arsenide is preferred

for diode fabrication. The current state of microphotolithography extends the quasi-microwave approach into the one-micron region and thus matches with gallium arsenide. Signal coupling to the amplifier will be provided by image line dielectric waveguide. An optical circulator is desirable to separate the input and output ports and provide a degree of stability. The development of heteroepitaxial ferrite and the associated photolithography completes the requirements so that a circulator may be integrated with the optical parametric amplifier.

CONCLUSION

It is hoped that a future report will verify that the quasi-microwave approach to optical information processing is feasible by demonstration of operation. At this point in the development, interest may be kindled for the extension into adaptive image processing by an array where spatial logic is achieved optically.

ACKNOWLEDGMENTS

Although the conclusions are those of the author, he is indebted to many colleagues at the Autonetics Research Center. In particular, sincere appreciation is due to Mr. R. R. August for the many stimulating discussions of optical semiconductor technology, to Dr. W. T. Cathey and Mr. J. E. Rau for use of the spatial filters, and to Mr. D. Medellin for the photomicrography.

REFERENCES

1. J. T. Tippett et al, eds., *Optical and Electro-Optical Information Processing*, MIT Press, Cambridge, Mass., 1965.

2. E. L. O'Neill, "Spatial Filtering in Optics," *IRE Trans. Inform., T.*, IT-2, p. 56 (1956).

3. L. J. Cutrona et al, "Optical Data Processing and Filtering Systems," ibid, IT-6, p. 386 (1960).

4. A. Vander Lugt, "Signal Detection by Complex Spatial Filtering," ibid, IT-10, p. 139 (1964).

5. E. N. Leith and J. Upatnieks, "Reconstructed Wavefronts and Communication Theory," *J. Opt. Soc. Am.*, vol. 52, no. 10, p. 1123 (1963).

6. D. Gabor, "Microscopy by Reconstructed Wave-Fronts," *Proc. Roy. Soc.* (London) Ser. A, vol. 197, p. 454 (1949).

7. E. N. Leith and J. Upatnieks, "Wavefront Reconstruction with Continuous-Tone Objects," *J. Opt. Soc. Am.*, vol. 53, no. 12, p. 1377 (1963).

8. G. R. Pulliam et al, "Epitaxial Ferrite Memory Planes," *1965 NAECON*, Dayton, Ohio, pp. 241–245 (1965).

9. T. S. Moss, "Methods of Modulating Infrared Beams," *Infrared Physics*, vol. 2, p. 129 (1962).

10. E. Snitzer, "Cylindrical Dielectric Waveguide Modes," *J. Opt. Soc. Am.*, vol. 51, p. 491 (1961).

11. N. S. Kapany and J. J. Burke, "Dielectric Waveguides at Optical Frequencies," *solid/state/design*, vol. 3, p. 35 (1962).

12. J. Kane and H. Osterberg, "Optical Characteristics of Planar Guided Modes," *J. Opt. Soc. Am.*, vol. 54, p. 347 (1964).

13. D. B. Anderson and R. R. August, "Applications of Microphotolithography to Millimeter and Infrared Devices," *Proc. IEEE*, vol. 54, (Apr. 1966).

14. M. Born and E. Wolf, *Principles of Optics*, Pergamon Press, New York, 1959, p. 261.

15. A. Yariv and R. C. C. Leite, "Dielectric-Waveguide Mode of Light Propagation in p-n Junctions," *Appl. Phys. Letters*, vol. 2, p. 55 (1963).

16. W. L. Bond et al, "Observation of the Dielectric Waveguide Mode of Light Propagation in p-n Junctions," ibid, p. 57.

17. A. Ashkin and M. Gershenzon, "Reflection and Guiding of Light on p-n Junctions," *J. Appl. Phys.*, vol. 34, p. 2116 (1963).

18. R. C. C. Leite and A. Yariv, "On Mode Confinement in p-n Junctions," *Proc. IEEE*, vol. 51, p. 1035 (1963).

19. W. W. Anderson, "Mode Confinement and Gain in Junction Lasers," *IEEE Journal of Quantum Electronics*, vol. QE-1, no. 6, p. 228 (1965).

20. D. F. Nelson and F. K. Reinhart, "Light Modulation by the Electro-Optic Effect in Reverse-Biased GaP p-n Junctions," *Appl. Phys. Letters*, vol. 5, no. 7, p. 148 (1964).

21. J. M. Manley and H. E. Rowe, "Some General Properties of Nonlinear Elements—Part I General Energy Relations," *Proc. IRE*, vol. 44, p. 9041 (1956).

22. C. C. Wang and G. W. Racette, "Measurement of Parametric Gain Accompanying Optical Difference Frequency Generation," *Appl. Phys. Letters*, vol. 6, no. 8, p. 169 (1965).

23. J. A. Giordmaine and R. C. Miller, "Tunable Coherent Parametric Oscillation in $LiNbO_3$ at Optical Frequencies," *Phys. Rev. Letters*, vol. 14, no. 24, p. 973 (1965).

24. D. B. Anderson, "Application of Semiconductor Technology to Coherent Optical Transducers and Spatial Filters," *Optical and Electro-Optical Information Processing*, J. Tippett et al, eds., MIT Press, Cambridge, Mass., 1965, pp. 221–234.

TIME-SHARING IN THE IBM SYSTEM/360: MODEL 67

Charles T. Gibson
International Business Machines Corporation
Cambridge, Massachusetts

INTRODUCTION

The basic architecture of the IBM System/360 makes it well suited to processing in a multiprogramming and multiprocessing environment. The Model 67 extends this basic architecture to provide the additional capabilities of an advanced time-sharing system.

The Model 67 incorporates multiprogramming, multiprocessing, and multiaccess capabilities. Multiaccess allows several users at remote consoles to communicate directly with the system and to present a number of applications ranging from conversational compiling to desk calculator functions. Multiprogramming is defined as the ability to have several active programs reside in core simultaneously. As soon as one job is finished, or is held up by an I/O request, or has depleted its time allowance, the next task can begin immediately.

The dynamic relocation feature built into the hardware facilitates multiprogramming; peripheral operations will now be just like any other tasks in the memory. Even without the multiaccess capability, multiprogramming provides much more efficient utilization of the computer's resources than in a stacked job operation. For the first time, a central processing unit is a resource that can be allocated. With multiaccessing, where some of the jobs in core belong to remote terminals, the multiprogramming capability is further enhanced as this enables the rapid switching between jobs, or "time-slicing."

The Model 67 enables each processor of a multiprocessor system to operate as a single processor with its own I/O subsystem, or, jointly with other processors in a symmetric multiprocessing configuration.

To achieve time-sharing and multiprogramming, certain modifications to the product line were made. It is the object of this paper to discuss a typical configuration of equipment and how it will be used. Certain areas where the programming systems relate to the hardware will also be discussed.

CONFIGURATION

Bus Structure

A sample configuration is shown in Fig. 1. The most significant part of the equipment is the shared memories. Each memory module has four "tails," or buses, with one tail connected to each CPU and one to each channel controller. A system having two processors and two channel controllers has four buses. Each memory module of 262,144 bytes of 750 nanosecond storage is independent and three memory accesses can occur simultaneously. Conflicts that occur among the several buses connected to each storage unit are resolved at the storage unit. This conflict resolution adds 150 nanoseconds to

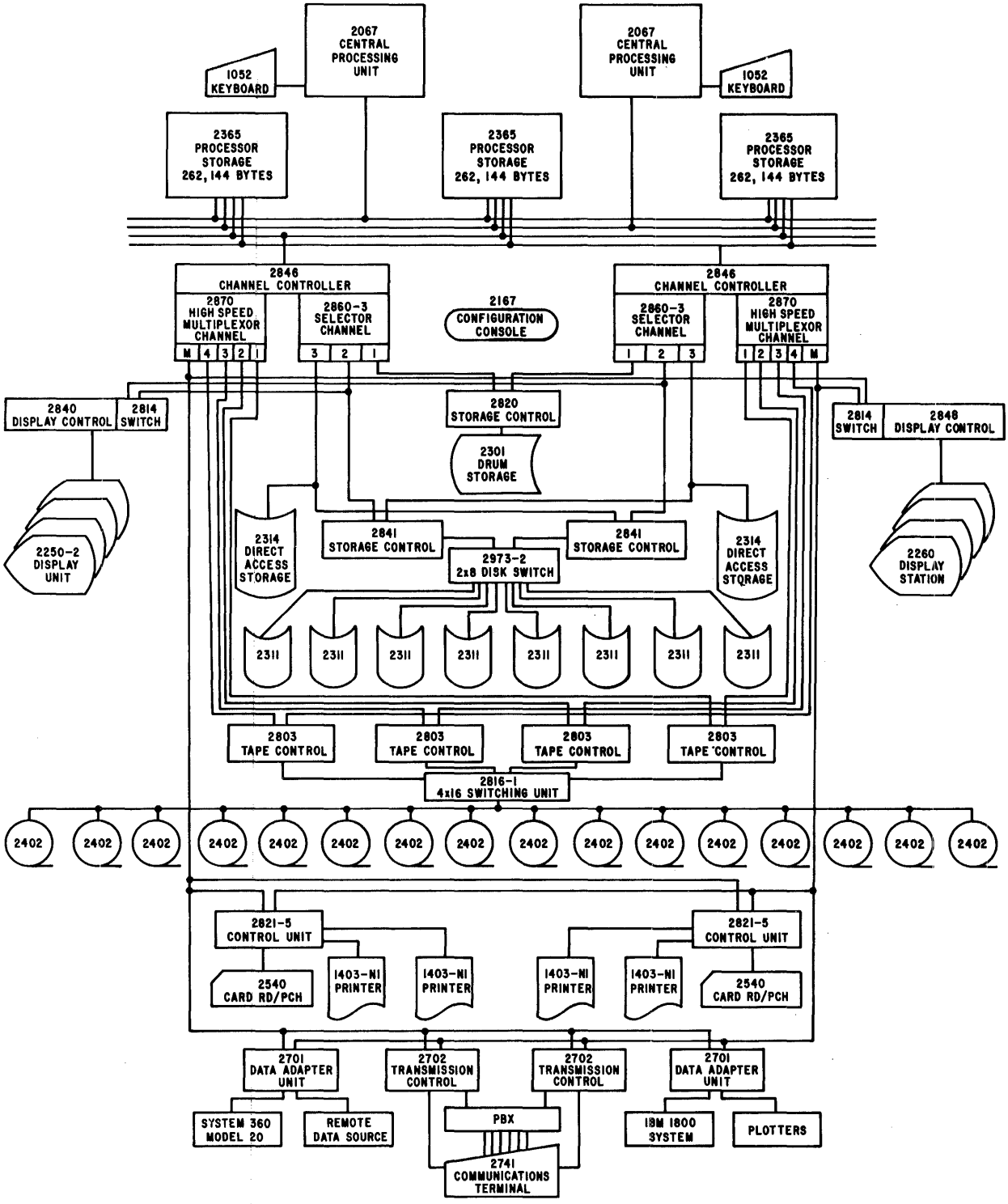


Figure 1. Configuration of typical Model 67.

storage access time. Channel controller requests for storage cycles are given priority over processor requests. No longer is the CPU tied to a particular memory in the classical configuration. Now, both channels may be loading two memories simultaneously at the maximum data rates, and there will be no interference with the CPU's if they operate from the third module. Likewise, a CPU and channel could associate with one module in the classical simplex method. Besides permitting higher data rates to and from core, and with less interference, the Module 67 frees the processors from the usual role of working on a single job until it is done. One CPU can work on a job in one core until it requires a routine from the disk, for example. While the data transfer is being made, that CPU could process another job in another module.

Common Routines

With shared storage, both CPU's could access the same compiler and monitor and could utilize the same job queues. In order for more than one CPU to use the same copy of a routine, the routine must be in reenterable coding. That is, there are no address modification or common storage locations that would be affected by a second program which happened to start the routine before the first ended it. In fact, one job, partly through a routine, might be interrupted to return after other jobs in the *same* CPU had passed through. Reentrancy, therefore, is a requirement of multiprogramming as well as multiprocessing.

A parallel reenterable routine is one which can be executed simultaneously by more than one task. In Time-Sharing System/360 there is only one copy of the supervisor in core regardless of the number of CPU's attached. All of the supervisor is in reenterable coding; some of it parallel reenterable.

In many cases, however, it is impossible or undesirable to have more than one task at a time in a routine; for example, one that sets or detects bits in a common table. These routines are consequently written in a serially reenterable fashion.

To aid in this, a new instruction, Test and Set, is implemented. Its operation uses the left-most bit of the specified byte to set the condition code. Simultaneously, the byte is set to all ones before another access to the same storage is permitted. Test and Set allows a second processor to check whether the first has started a serially reenterable routine. In this way, the first processor job can finish with the routine before allowing its further use. Also, Test and Set is a means of breaking ties between two CPU's which become available for a new task at the same moment.

Signaling

For communications through common storage elements, a processor must be alerted when a message has been prepared for it by another processor. The extended direct-control feature and external interrupt lines of the Model 67 perform this function.

Associated with the direct-control instructions is an interface at which eight signals are made available. A signal from one processor is connected to one of the external interruption lines of another processor. By means of the Read Direct or Write Direct instructions, the program in one processor causes an external interruption in another processor.

If a CPU recognizes a hardware error as indicated by a machine check interrupt, it alerts another CPU by the direct-control feature and causes a malfunction alert interrupt. The troubled CPU then goes into the wait state while the second begins the recovery procedure.

Channel Controller

It is difficult to break away from traditional thinking of a memory-CPU channel system configuration. With multiple tails on the memory and independent channels, the CPU's really behave as pumps, with one or more processing units working on many jobs located in multiple memories. The 2846 Channel Controller is essentially the same bus control unit as in the CPU. It enables the 2860 Selector and 2870 High Speed Multiplexor Channels to communicate directly with the memories without interfering with the CPU's bus control units. Channels are addressed by, and can return their interruptions to, either CPU, providing a flexible and general organization. I/O terminations, therefore, are not unique to a CPU, but may cause an interruption in whatever CPU, by masking, is conditioned to accept them.

The 2870 High Speed Multiplexor has the ability to handle up to eight control units and 192 devices on the basic interface, which can operate in the multiplexor or burst modes. Up to four medium speed interfaces, which can address up to 16 devices, are available on the 2870. These selector subchannels only operate in the burst mode.

The basic interface can handle an aggregate rate of 110k bytes per second with no selector subchan-

Table 1. 2870 Data Rates in kbps*

Basic Interface	Selector Subchannel 1st	2d	3d	4th
110				
88	180			
66	180	180		
44	180	180	180	
30	180	180	180	100

*kbps = Thousand bytes per second.

nels attached. Lower data rates can be sustained with the subchannels as shown in Table 1.

Dual Data Paths

Dual data paths have been provided to enable the memories to reach any I/O device by at least two paths. Partly for reasons of reliability, the dual paths also provide flexibility in pathfinding in case one control unit is busy. All of the I/O control units have two tails, one to each I/O control element, which are under program control. This allows each channel controller to reach each I/O device, and via any control unit in the case of pooled devices. The 2973-2 Disk Switch, for example, allows any two 2311 Disk Storage Units to operate simultaneously through the two 2841 Storage Control Units. Because of the twin tails on the 2841s, each Channel Controller can have as many as two 2311 disks operating through it simultaneously.

Similarly, any four of the sixteen 90KC 2400 tape units may be operating with any combination of channels. The 2973 switch and two-tailed control units are unique to time-sharing configurations.

Pathfinding

With more than one possible logical route from memory to a device, a pathfinding routine is necessary. The pathfinding routine will locate the first available logical route in which the channel and control unit are neither busy nor unavailable. In Fig. 1, for example, there are eight possible paths for data to get to memory from a tape unit.

The routine is serially reenterable. The supervisor enters the pathfinder by giving it a symbolic device address. From a Symbolic Address table, the low order bits of the actual I/O address, called the device address, are immediately available.

The Symbolic Address table points to a Device Group table where all possible device paths for a group consisting of similar devices in a common pool are given.

The pathfinding routine, knowing from the device path which channel and control unit are needed, checks the corresponding Channel and Control Unit tables for their availability.

In the System/360, a selector channel is busy if any device is operational on it. On the multiplexor channel, however, the data rates of each operational subchannel must be examined so that the total multiplexing data capability will not be exceeded by the addition of another device. Consequently, the multiplexor channel is not busy until the total "weights" of all devices attached exceeds 110kc. The "weights" of devices connected to the basic interface are their data rates and come from the Device Group table; the weights for any 90kc magnetic tapes on the selector subchannels are 15kc; for 180kc tapes the weights are 22kc.

If the channel or the control unit is busy, the path is abandoned. If all the possible paths are busy, the pathfinding routine returns with the appropriate "busy" or "not available" bit set. A reverse pathfinding mechanism also exists to clear the busy bits for the device, control unit and channel when an operation is completed. The tables are created at system generation time and can be modified by the partitioning routines.

Error Recovery Procedures

When a machine error is determined by CPU hardware, a machine check interrupt occurs in the CPU and this same signal is broadcast to all other CPU's in the system, which receive such indications as malfunction (external) interrupts.

The original CPU will be put in wait state with interrupts masked, thus preventing it from disrupting the total system. One of the other CPU's in the system accepts the associated malfunction alert; the others going into the wait state. It is the function of the active CPU via the "recovery nucleus" to identify the failing unit in order to remove it from the active system. Each CPU has a recovery nucleus in a different memory module.

When a less disastrous fault occurs in the system, such as failure to read a record correctly from a storage device, the time-sharing monitor will invoke a standard retry routine. If this retry routine fails to read the record correctly, it will report this information to the time-sharing supervisor. The supervisor will log this information and will then call for a system error analysis program, which will decide which units are to be eliminated from the resource table in the supervisor.

The decision as to which unit or units to drop from the resource table is made by examining the recorded error environment information, then determining the partitioning which would have the least impact on system performance. For example, when a fault occurs in a unit which has two data paths, the system error analysis program will analyze the fault to determine if one of the data paths or the unit itself should be eliminated from the resource table. The program will not eliminate an operational unit from a resource table if there is at least one data path to that unit.

Messages will be sent to the operator when a data path is eliminated from the resource table, but no maintenance action will be taken until either all data paths to the unit are out or until the customer engineer and operator decide that maintenance is required. At this time, the customer engineer will call for the diagnostic monitor and begin the diagnostic procedures. If a CPU or memory element fails, a warning message is broadcast to all active terminal users who might be affected.

Remote Access

The Time-Sharing System will support the IBM 1052, 2741 and 2260 Display as remote terminal devices. The terminals may be connected locally as operator consoles, or remotely as user terminals. The 2702 Transmission Control and 2848 Display Control accept data, serially by bit, and the transfer to memory is made one byte at a time. The 2701 Data Adapter Unit is the general interface enabling remote on-line attachment of IBM Model 20s, Model 30s, other computers, data sources, and plotters. The interface can be a serial-by-bit communications adapter or the parallel data adapter. The latter is an interface which can accept data up to 1,200,000 bytes per second and up to 48 bits at a time.

DYNAMIC RELOCATION

Multiprogramming

While the typical scientific computing installation may have many large production programs in its work load, it will have many more programs which are small compared to the hardware resources. The maximal claims made by such programs do not include all of the I/O units or all of the core storage. Moreover, these claims are indeed maximal in that they include space in the storage for instructions, intermediate results, and initial and final values which are highly dependent on the precise data set used in a particular execution of the program. The claims include channels and I/O units, the use of which is also highly dependent on the data set. Some recent measurements indicate that for many programs a storage area two or three times as large as necessary is claimed.

Multiprogramming is a processing mode in which a control program attempts to honor the hardware claims of several distinct programs simultaneously. The object is to keep the CPU busy executing problem program instructions rather than allow it to remain idle during I/O operations which may arise in a problem program or in a control program which is removing one problem program from the machine and bringing in another. The goal is to increase throughput.

A further refinement in multiprogramming comes from the observation that many machine runs consist of a compilation and an execution or an assembly and an execution. In a multiprogramming system, one would therefore discover many copies of the compiler, assembler, I/O routines, and programs from a math library existing in core storage at the same time. Consequently, the notion of common, reenterable programs arise.

A crude picture of time-sharing shows a control program which, during these moments of idleness with respect to one console, outputs all of the optional storage, inputs a previously outputted store image for a second console, and then permits the execution of a transaction for that console. There is a cost in this operating mode in terms of time used for the exchange of core images in order to respond to console requests. In fact, the larger and more complex the program, the more useful on-line debugging can be, and the greater the cost of exchanging problem programs. This cost is an especially painful burden since the debugging process is often such that no more information is required for a console transaction for a large program than would be required for debugging a small program.

What is needed in view of the application is a system technique which encourages the transaction aspect of time-sharing for debugging purposes, a technique for elimination of redundant copies of popular functions, and a facility for dynamic core allocation. An ability on the part of the control program to put a problem program in ready status without honoring the complete storage claims and without the necessity of altering the addressing structure of the program is of great value. Properly

arranged, it permits effective multiprogramming and multiprocessing.

This ability can be obtained through the notion of a logical store and the high-speed monitoring of addresses used by problem programs, and the conversion of these logical addresses into other physical addresses. Virtual storage is defined as the entire storage which can be reached by the logical addressing scheme. Thus, with 24 bits, there are 2^{24}, or 16,777,216 byte locations in virtual memory. With the 32-bit relocation option, 2^{32} or 4,294,967,296 bytes are addressable in virtual memory. However, in either case, the physical storage in the sample configuration would be only 786,432 bytes. The dynamic relocation scheme described here is the method by which the virtual storage is mapped into the physical storage.

Dynamic relocation is achieved by treating the addresses supplied by the program as logical addresses, or relative addresses. A logical address is identical to a physical address when the relocation feature is not operative. When relocation is employed, the logical address is that address known by the program. The physical address is the address (after any relocation) presented to memory for a reference. The logical addresses are translated by means of a relocation table to physical addresses when storage is addressed.

The relocation function provides the ability to interrupt a program and record it on external media such as a file or drum and at a later time to return the program to main storage at different storage locations without disturbing the execution of the program except for the time element involved. The locations at which a program and its data are stored are assigned by the relocation table and occur in 4096-byte blocks. These blocks need not be contiguous even though they may be addressed by a contiguous set of logical addresses. The physical fragmentation of programs is thus not apparent to the programmer.

Relocation Operation

Space-sharing is facilitated by breaking user programs into segments and sectioning these segments into pages. By breaking programs up into pages, physical memory may be allocated in page increments. Only those active pages are brought into physical core storage.

The Model 67 CPU may operate in either the relocation or nonrelocation mode. With the 32-bit option, the CPU may also run in either the 24- or 32-bit mode. The modes are specified by bits 4 and 5 of the extended mode Program Status Word (XPSW) as follows:

Bit 4	Bit 5	Modes
0	0	No relocation, 24-bit address
0	1	Relocation, 24-bit address
1	1	Relocation, 32-bit address
1	0	Data exception

The function of the Program Status Word is explained below (under "Other Features"). All normal instructions are relocated when bit 5 is set. Addresses of control words and data in I/O operations, however, are not relocated. Addresses generated by the CPU or channels for interruption purposes, such as timer, CSW, and PSW addresses, are not relocated. The standard, 24-bit relocation scheme is described in this section.

All logical addresses are formed using full 32-bit arithmetic. The sum of the 32-bit base register specified in an instruction as R1, the 32-bit index register specified as X1, and the 12-bit byte address, or displacement, less the 8 high order bits forms the 24-bit logical address. The logical address is broken into three sections of 4, 8, and 12 bits specifying the "segment" number, the "page" number, and the byte, or "line" number, respectively. Figure 2 shows this breakdown.

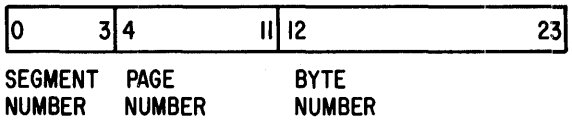


Figure 2. Logical address.

There are 4096 bytes per page, where a page of programs resides in a block of core storage. With 8 bits for the page number, there are 256 possible pages per segment. Each segment (1,048,576-byte address space) can contain a program. It can contain a data set. It can also be a million-byte area of working space. If the segment number is ignored, each such program, data set, or space begins at address zero. In the 24-bit scheme, several routines will be packed in the same segment.

The relocation scheme operates essentially as follows. Each task in the system requires for its operation one segment table and a page table for each segment used. These tables are developed by the monitor as the task is created and as new segment and page requirements are made by the task. As re-

quired pages are fetched into physical core, the monitor enters in the page table the physical location of the logical page. During program execution, the hardware automatically does a table look-up on each address as it is referenced by the user and the physical address is chosen for each logical address reference. If the user references a location in a page that is not in physical core, an automatic interrupt occurs to the monitor, which then sets up a page turning routine to fetch the missing page. The monitor then gives the CPU to the next user in the queue. The waiting program is held in the wait status until its required page has been fetched and assigned somewhere in physical core. It is then returned to the queue of active users and takes its turn vying for CPU time. All of this is transparent to the user. To him, the memory is as many as 16 segments of as many as 1,048,576 bytes each.

Let us look at the relocation scheme in more detail. There is a 32-bit Table Register, program addressable, shown in Fig. 3. Bits 0–7 of the Table

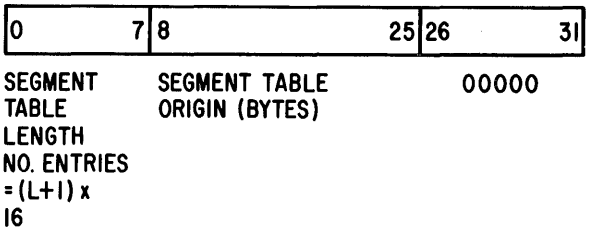


Figure 3. Table Register.

Register specify the length of the segment table, and bits 8–31 specify its origin. The segment table must be located at an address which is a multiple of 64, and thus bits 26 to 31 of the Table Register must be zero. If any bits 26 to 31 are one, a specification exception occurs (program interruption with bit 21 set). Bits 0–3 of the logical address are added to bits 26–29 of the Table Register. The resulting 24-bit address (TR_{8-31}) points to a unique 4-byte entry in the segment table.

The number of entries in the segment table is 16 times the number formed by the sum of the Table Register, bits 0–7, and one. Thus, the minimum length, with bits 0–7 zero, is 16 entries. These bits 0–7, with 4 appended low order virtual bits of one, are compared with the segment number. If the segment number is greater, a relocation exception is recognized (program interruption with bit 16 set). Obviously, in the 24-bit relocation scheme, the logical address cannot specify more than 16 entries,

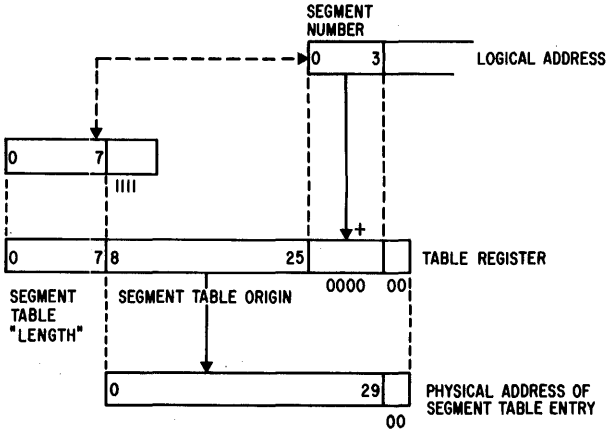


Figure 4. Segment table entry.

the minimum size; a greater-than compare can never result; and this check is academic.

Each 4-byte entry in the segment table is similar to the Table Register. Bits 8–31 specify the origin of a page table, while bits 0–7 give its length. Each page table is originated at an address which is a multiple of 16 so that bit 31 of the segment table entry must be zero. If bit 31 is one, a relocation exception is recognized.

The page number, bits 4–11 of the logical address, is added to the page table origin, bits 23–30 of the segment table entry, to give the unique page table entry. Each entry consists of two bytes. As with the Table Register, a relocation exception occurs if the page number is greater than the page table length. The minimum page table has bits 0–7 zero, corresponding to one entry, which translates addresses in the range 0–4095. The maximum page table has 256 entries, one for each page in the segment.

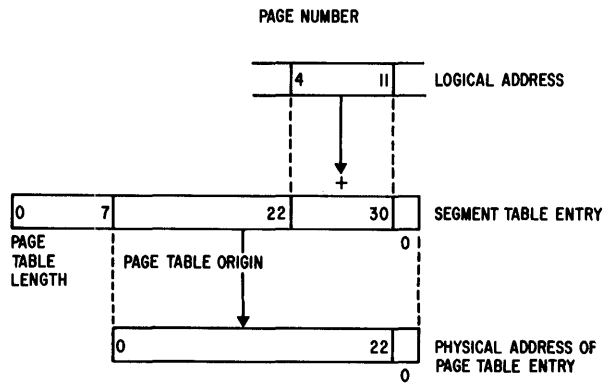


Figure 5. Page table entry.

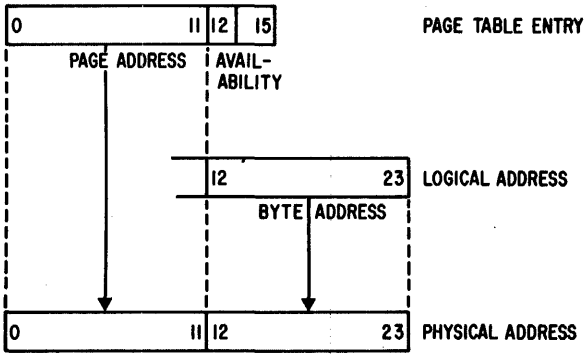


Figure 6. Physical address.

The 2-byte page table entry consists of 12 bits specifying the high order 12 bits of the physical address. The low order 12 bits of the address are the same as the corresponding part of the logical address. There is no relocation within the page, consequently. Bit 12 of the page table entry is the availability bit. When zero, the respective program page is in core and the entry may be used for relocation. When a one, the address may not be relocated because the desired page is not in core, and must be brought in under monitor control. Therefore, a protection exception (bit 19) is recognized and the instruction is suppressed.

If the instruction uses variable length fields, a look-ahead operation is performed first to see if the storage page boundary will be crossed and, if so, whether the new page is available. If the next page is used but unavailable, execution is suppressed.

Bits 13 to 15 of the page table entry are undefined and must be zero. When a relocation exception occurs, no storage reference is made. The logical address that would have been translated is recorded in the Relocation Exception Address Register. It may be inspected there by the new Store Multiple Control instruction.

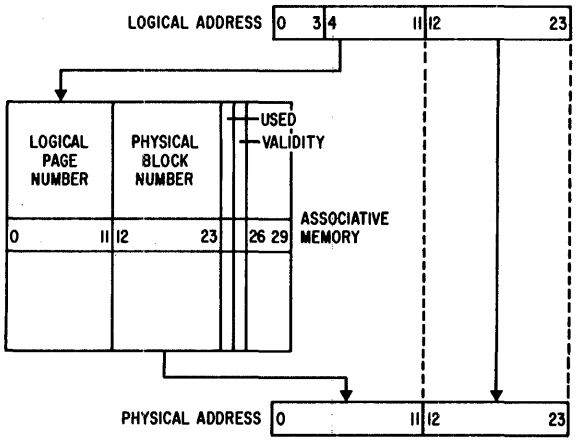


Figure 7. Associative compare.

Implementation

Although the dynamic relocation is performed entirely by hardware, one can see that two memory accesses would be required for each relocation. Each operand would require three memory accesses instead of one thereby greatly degrading performance. Therefore, the implementation of the Model 67 dynamic relocation has been modified.

A high-speed associative memory in local store is located in each IBM 2067 CPU. It contains 8 registers, each 30 bits wide. When an address is to be translated, the logical segment and page addresses are compared to bits 0–11 of each entry in the associative memory, in parallel. If there is an equality, bits 12–23 become directly the page address; the physical address and the relocation is completed without memory access. The use of the associative memory adds only 150 nanoseconds to the cycle time for each relocation.

Bit 25 in each association memory entry tells whether that entry refers to a page that is in core. If a zero, the corresponding page is not in core, the entry is invalid, and it is not used in comparison.

All bits in position 25 are reset upon a change in the Table Register since all the entries would be invalid. Bit 24 is set to one as each entry is used in relocation. When that bit vector is all ones, they are reset and the cycle repeats itself. When no comparison is found, the relocation hardware must look through the segment and page tables to find the physical page address. The new page table entry is then put into the associative memory. The new entry will leave bit 24 set to one. It will displace in the associative memory the next entry from the last one used whose "use" bit (24) is zero. By this algorithm, the least used entries are replaced and the associative memory holds the most frequently referenced page numbers to avoid the table look-up procedure.

A ninth associative register is a relocated instruction counter which is updated along with the instruction address bits of the Program Status Word. Therefore, there is both a logical and a relocated instruction counter. Upon a successful branch or crossing of a page boundary, the new logical address is relocated and the relocated instruction counter is updated by hardware.

Figure 8 is an overall picture of how the relocation works.

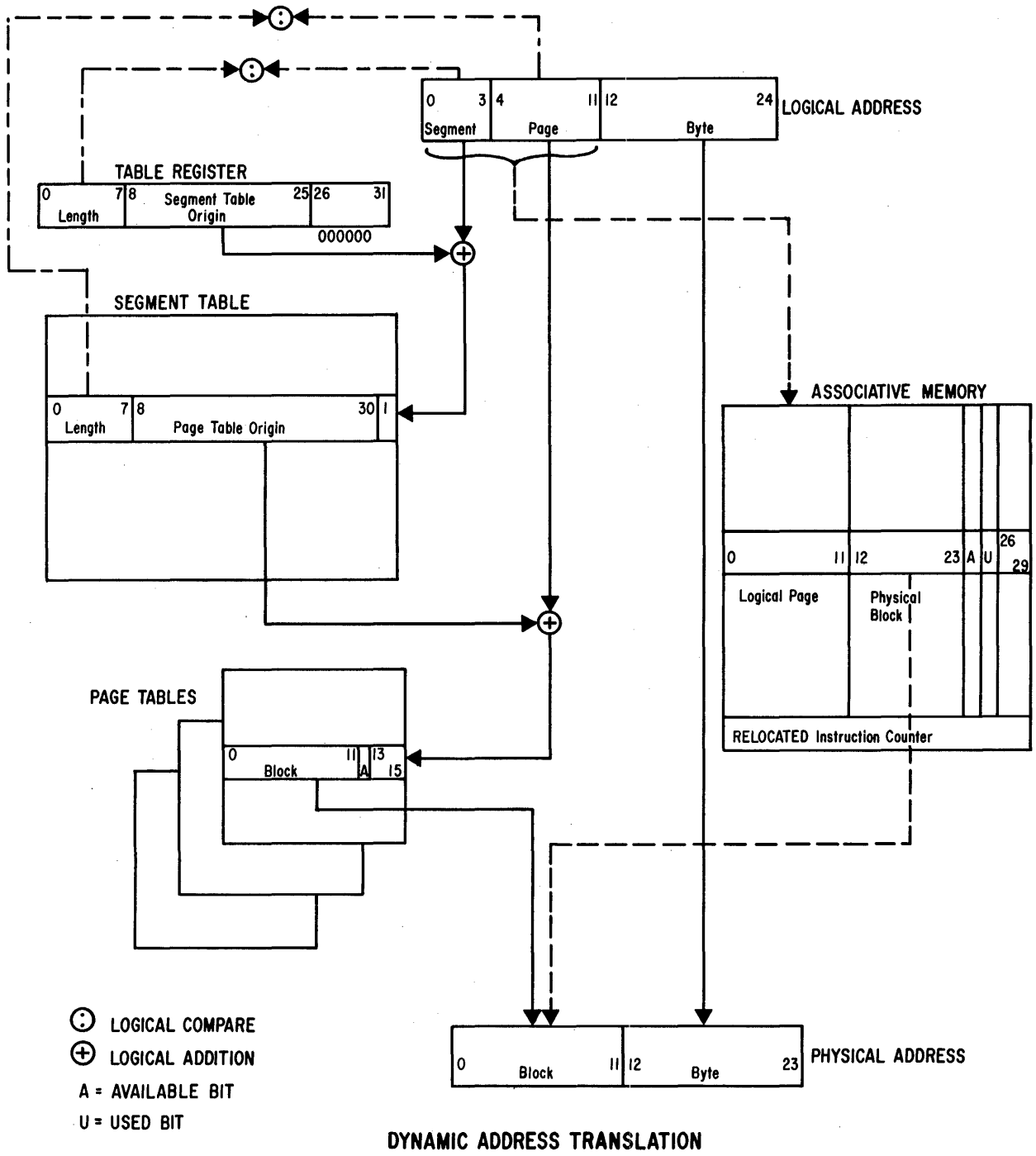


Figure 8. Relocation address translation.

An instruction Load Real Address has been added which inserts into a general register the dynamically translated address of the operand. This enables the supervisor to check the translation tables and find what a physical address is for a corresponding logical address.

We now have a scheme by which we can use all 16,777,216 bytes of virtual storage. Our logical addresses can cover the entire range from 0 to $2^{24}-1$ without regard to the amount of physical storage. The mapping from logical store to physical addresses is done by hardware, with supervisor calls

needed only to call into core storage a page of coding from external storage.

Virtual Memory

Although a programmer can address byte $2^{24}-1$, he will not be able to make use of the entire physical storage. There is a permanently resident supervisor which is not in a user's virtual memory and is not relocated. It runs in the System/360 supervisor state and so it can execute input/output and other privileged instructions. A user can only get to those pages in physical core that are in his virtual memory. All tables are managed by the supervisor. A user's Table Register points to a Segment Table whose entries in turn point to the Page Tables which define valid program pages and thus define acceptable addresses. Another user's virtual memory is made available simply by changing the Table Register. Virtual memory can therefore be defined as a set of relocation tables. Paging, I/O and interrupt handling are done by the supervisor which is protected from user programs because it is not in any user's virtual memory. Between the problem programs and the resident supervisor, there is an intermediate level of privileged service programs, as Fig. 9 shows. The figure also shows that the service programs are protected from errant problem programs by being in the supervisor state.

This is dynamic, hardware-aided address relocation. It has created a new concept of virtual storage, in which the user need not worry about the limits of physical storage nor about where his program really is in core. With relocation, users share common reentrant coded subroutines to avoid redundant copies. Complete storage claims of programs are honored only when needed, greatly reducing core requirements. True multiprogramming with several programs sharing core is possible for the first time, and this is the heart of the Model 67.

OTHER FEATURES

Extended Program Status Word

The Program Status Word (PSW) is a 64-bit program-addressable word that contains all the detailed CPU and program status information necessary to describe the present condition of the machine. It includes bits to mask off I/O and program interrupts, bits that can be sensed following a logical operation (cc), the program protection key and the

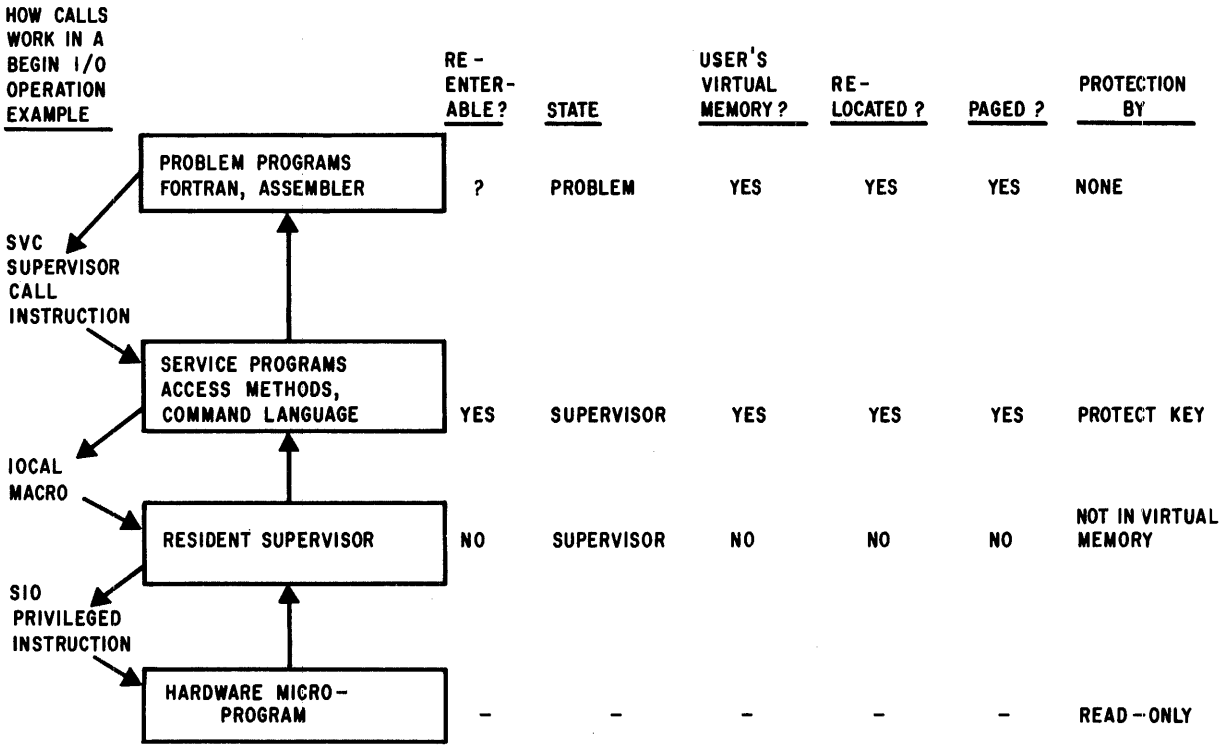


Figure 9. Protection levels.

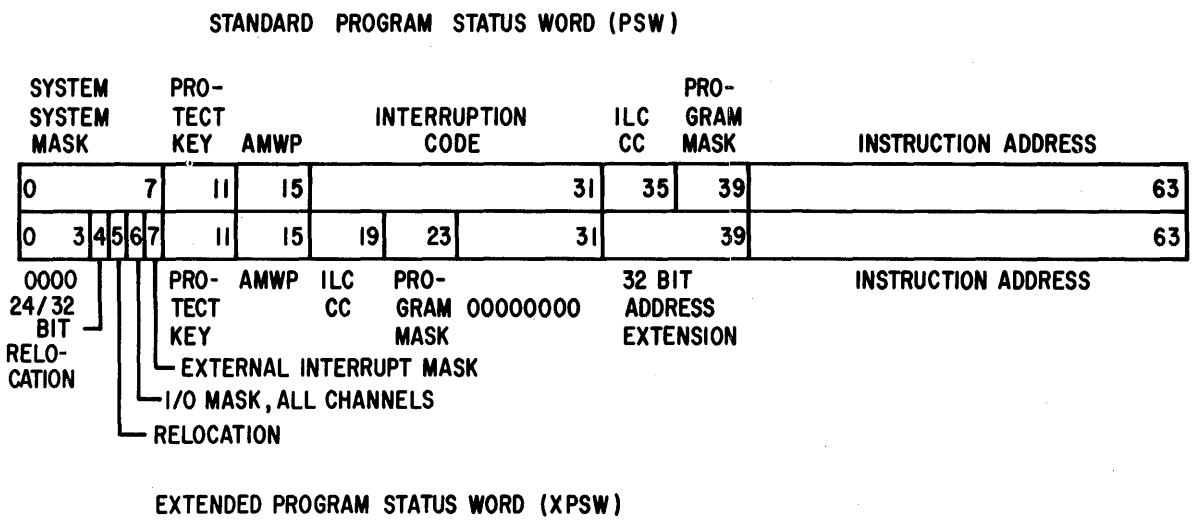


Figure 10. Program status word format.

instruction counter. Upon receiving an enabled interrupt, a new PSW is automatically loaded while the old PSW is stored in a unique core location, thereby immediately entering an error routine with a different machine status. The five types of interrupts are I/O, machine-check, program exception, supervisor call, and external signal and each type causes a unique PSW to be loaded.

The Model 67 makes use of an Extended Program Status Word (XPSW) which has some expanded functions, while other functions are left in various control registers. The normal and extended PSW are shown in Fig. 10.

The native mode of the Model 67 is that of any other System/360. After power-on, there is no relocation and a standard PSW format is used so that it is compatible with System/360. However, the Load Multiple Control instruction can load bit 8 of control register 6 to make use of the Extended Program Status Word. When using the XPSW, the interruption codes are placed in core locations 14 through 23.

Fetch Protection

To achieve effective System/360 time-sharing, it is necessary to provide for confidential files, which are available only to privileged users. Accounting data, password information and personnel files are examples of restricted files. There is also the need for protection of one task against an errant problem program. Consequently, a read protect feature has been incorporated in the Model 67 by adding a fifth, fetch-protect bit to the 4-bit storage key.

There is a 4-bit key associated with every 2048-byte block in System/360. A problem program is allowed to write into that block only if the key matches the current key in the Program Status Word. An exception occurs if there is no match. The block is likewise protected against I/O data by the requirement of a similar matching key in the Channel Address Word. On the Model 67, if the fifth bit is a one, the corresponding 2048-byte block is fetch (read) protected as well whenever it is read protected. Fetch protect without write protect is neither possible nor desirable. The privileged instructions Set and Insert Storage Key now transmit seven instead of four bits to and from the storage key.

The fetch and write protection features are used to keep a problem program from service routines in its virtual memory, as Fig. 9 shows.

There are two other means of protection in the Model 67. One is the limitation of a problem program to addresses within its virtual memory, as defined by its set of page tables. Attempts to transfer, read, or write outside of one's virtual memory results in a relocation exception.

There is also the supervisor state in System/360 which restricts nonprivileged programs to certain instructions. This effectively protects the supervisor against unintentional or intentional memory accesses that are out-of-bounds.

Rolling out of a user's page at the end of his time slice is necessary if the memory space is needed for another task. However, if nothing has been written into that page, there is no need to swap it out be-

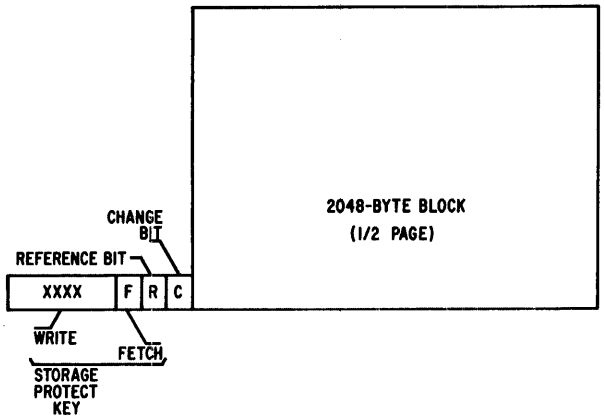


Figure 11. Fetch protect.

cause a valid, up-to-date copy of the page already exists on the paging device. Two extra bits on the Model 67 have been added to the standard System/360 protect key, one to indicate if the corresponding 2048-byte block has been referenced, and one to indicate if the block has been written in. These are tested by the Insert Storage Key instruction to determine if paging is necessary.

Figure 11 shows the fetch protect and reference bits.

Control Registers

A set of up to 16 control registers is provided on the Model 67 to implement various features and to allow for the increased number of memories and channels that a CPU may address. Each may be as long as 32 bits. Some registers may be used to sense positions of switches and are not actually "hard" registers. The control registers may be loaded (where possible) and stored using the two new instructions, Load Multiple Control and Store Multiple Control. A list of the control registers is given in Table 2.

When using the Extended PSW, the I/O channel masks are used from control registers 4 and 5, and are controlled by XPSW bit 6.

Extended I/O Addressing

Each CPU can now address and mask up to 28 channels. The channel address is designated by bits 19–23, instead of 21–23, of the logical address in addition to the remainder of the address in bits 24 to 31. The condition code 3 is set in the PSW if a nonoperational channel is addressed. Figure 12 shows the bit structure of an I/O device address as used by the Start I/O instruction.

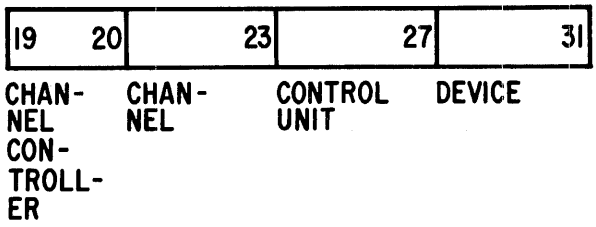


Figure 12. I/O address format

Timer

Each CPU contains an interval timer, which permits necessary time-accounting functions to be performed in addition to the above mentioned program monitoring. The timer has a counting interval of 13 microseconds, corresponding to a frequency of 19.2 kc. Counting takes place in bit 31 of the timer location, storage location 80. Actual implementation includes the use of an internal register to reduce storage interference to the level of the standard 60-cycle timer. Location address 80 is monitored to assure an updated timer content whenever this location is referenced.

Prefix

Each CPU uses absolute core locations 0–127 for PSW's, channel address words, channel status words, timer residence and initial program loading. Were these locations common, they would be shared by several CPU's and interference between CPU's would result. Therefore, to provide each CPU with separate assigned storage, a quantity called a prefix is used to relocate dynamically the first 4096 storage locations. In multiprocessor operation each CPU is normally assigned a unique prefix and hence the sharing of these preferred locations is avoided. Alternate prefixes are provided for each CPU in case of malfunction. The identity of the CPU executing a program may be determined at any moment by the setting apart of one of the addresses in the range 0–4095 as the address of an identifying location, and then loading an identification in each corresponding physical location.

The prefix area also contains the recovery nucleus, machine check information and temporary register storage when a base register is unavailable.

Each CPU has a prefix area in a different memory module for reasons of reliability.

Partitioning

A Time-Sharing System which is designed for availability may have enough redundant major

Table 2. Control Register Functions

Control Register No.	Bits	Function
0	0–31	Dynamic relocation Table Register.
2	0–23	Relocation Exception Address Register.
4, 5	0–63	Extended PSW I/O channel mask for channels 0–31; unassigned mask bits.
6	0–3	Machine check mask extensions for channel controller 0–3.
	8	Extended control mode.
	24–31	External interrupt masking.
8, 9	0–63	Status of core storage partitioning switches. One byte/memory module; one bit/tail. A one indicates that connection is established.
10	0–31	Core storage address assignment. 4 bits/each memory module, containing bits 11 to 14 of the assigned core storage address.
11	0–15	Status of channel controller partitioning switches. 4 bits/each controller; 1-bit/tail. A one indicates that a connection is established.
	16–31	Channel address assignment (as viewed from the CPU executing the STMC instruction), 4 bits/CPU. A field containing 1111 indicates that for the particular CPU all channel controllers are assigned their prewired addresses (i.e., channels 0–7, 8–15, 16–23, 24–31). A field containing 3 zeros and a one indicates that, for the particular CPU, only the channel controller corresponding to the bit position which is a one is addressable, and its channel addresses are 0–6. No other bit combinations are possible in these 4-bit fields.
12, 13		States of control-unit partitioning switches, with at least 2 bit positions assigned to each control unit. A one indicates that connection is established.
14	24–27	States of direct control partitioning switches, one bit for each CPU. A one indicates that the direct control interface of the corresponding CPU is connected to the other CPU's.
	28–31	States of prefix deactivation switches, one bit for each CPU. Zero indicates that the prefix of the corresponding CPU is deactivated.

systems components so that it is possible to divide or partition the system. Partitioning can be achieved without any additional hardware by relying on the programs in each system to refer only to those components which have been assigned to their use. However, often a more absolute means of partitioning may be required, such as when undebugged supervisors and real-time experiments might otherwise penetrate subsystem boundaries. This is done by physically partitioning the Model 67 by means of switches at a separate unit, the 2167 Partitioning Console, which allow any combination of memory modules, CPU's, channel controllers, I/O control units and I/O devices to be connected to the same system. Figure 13 shows the partitioning switches in the sample configuration.

The addresses of the memory units can be individually set under switch control for partitioning. In multiprocessing, the addressing for any particular byte is the same for either CPU, and the modules are addressed contiguously. In partitioning, the addresses of each module may be set to a multiple of

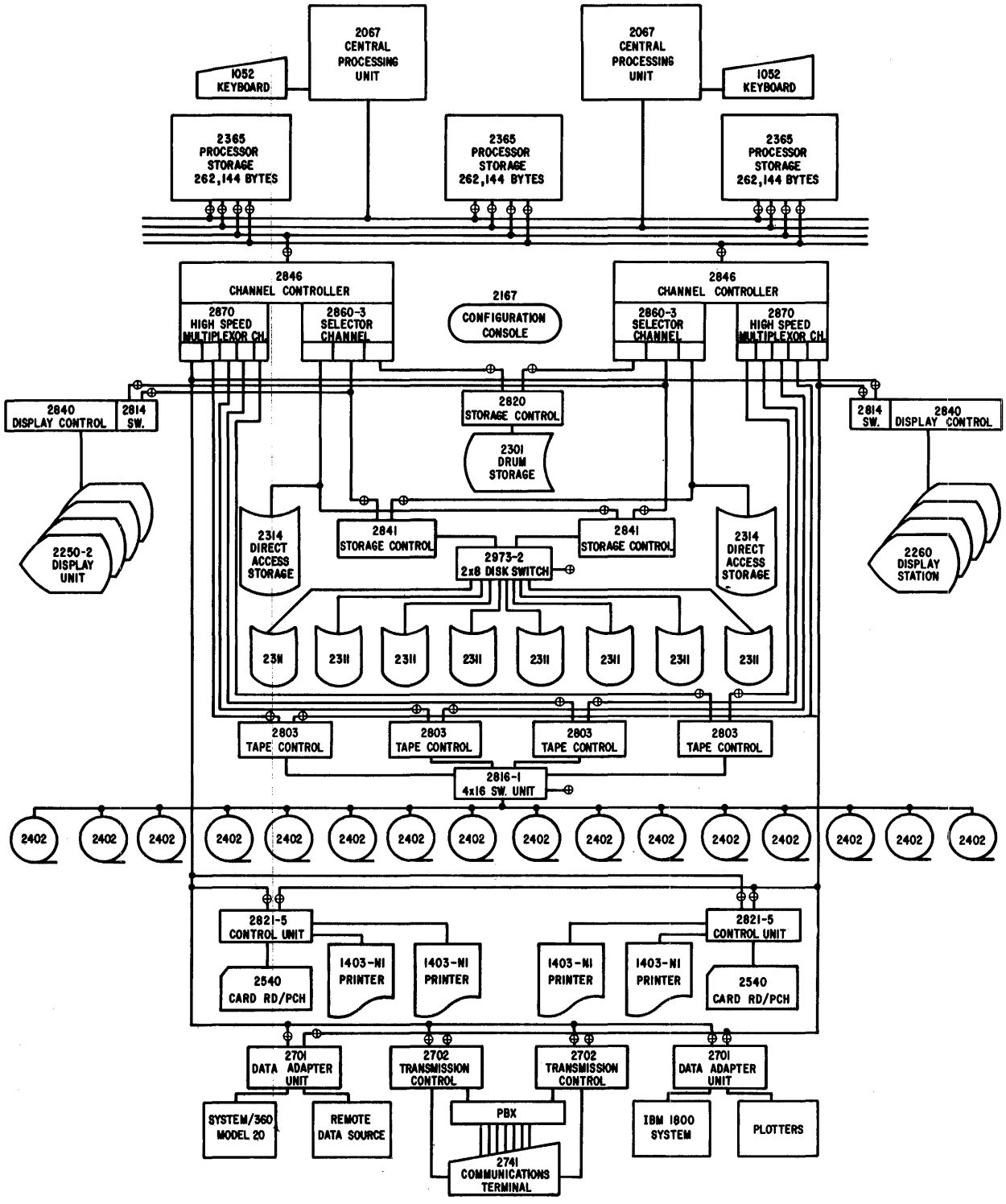


Figure 13. Configuration of typical Model 67 with partitioning switches.

the array size. Thus, with three memory modules, one partitioned CPU might have memory addresses 0–256k while the other CPU has 0–512k. This is referred to as floating memory addressing. An invalid address indication results if an unavailable memory unit is addressed.

In a partitioned mode, the floating channel address switch determines which CPU's may initiate commands on which controller. Bits in control register 11 reflect the setting of the channel address switch. In a multiprocessing mode using the Extended PSW, any CPU can address any device on any channel and the floating channel address switch is ignored. If a path is disabled, a command to that channel causes condition code 3 in the PSW to be set, indicating the channel is not operational. The CPU-memory, channel-memory, and CPU-channel control lines can thus be severed for partitioning.

The 4 × 16 IBM 2816 Tape Switch normally connects any of the pool of 16 drives to any tape control unit at program speeds. To partition, a plugboard enables each tape unit to be excluded from or connected to each control unit. Tape drives can thus be connected to separate channels, or shared on a channel. The IBM 2973-2 Disk Switch works similarly. The status of the partitioning switches may be sensed by means of the Store Multiple Control instruction. It is, therefore, possible for the two CPU systems to operate as two single processors, as independent processors sharing common storage and I/O units, or as a single multiprocessor system.

Reconfiguration can be achieved dynamically by means of privileged operator commands. The DETACH command will logically separate the specified unit from the system without disruption of services. If an I/O device is specified, activity on the unit is allowed simply to cease to achieve logical partitioning. If a memory unit is specified, the user data is allowed to be paged out without reassigning the core blocks. When the user areas are free, any remaining supervisor pages are moved to a remaining storage element. Any prefix area in the memory is reassigned to another memory element. When activity has "dried up" on the requested unit, a request is made for the operator to set the switch to partition off the unit. When the action is completed, the program will test the switch setting by checking the bits in Control Register 8, 9, 12 or 13. The supervisor then sets the appropriate bits in the path-finding device table to indicate unavailability and confirms physical partitioning by a message to the operator. The DETACH command therefore allows off-line operation by requiring TSS to gracefully withdraw from the affected units. An ATTACH command reverses the above procedure.

When the system is to be reconfigured, a PARTITION command can be given to logically partition the system according to one of several cataloged configurations. Again, after activity has "dried up," the supervisor first asks the operator to set the partitioning switches, then tests the switches and acknowledges physical partitioning.

TIMING

Memory Cycle Time

The basic storage cycle of the Model 67 is 750 nanoseconds. One double word of 8 bytes can be fetched every 750 nsec, interleaved with another 8-byte word 375 nsec after the first provided the double words have addresses which are alternately even and odd multiples of 8.

If repeated accesses were made to a byte within a group whose first byte has an address which is an odd multiple of 8 bytes, and then to one whose group has an address as an even 8 bytes, the memory cycle time would be 375 nsec. Furthermore, System/360 instructions are 2-, 4-, and 6-bytes long, with most being 4 bytes. Therefore, many pairs of consecutive instructions will occur in one 8-byte double word so that the second instruction will be available "free" with no access time. Therefore, memory accesses for operands will occur at an average rate of between 375 and 750 nsec.

The delay caused by priority determination of the four tails at each memory is 150 nanoseconds. This delay only occurs if consecutive accesses are not made from the same CPU or channel controller. There is no delay caused by priority determination where consecutive accesses are made from the same CPU or channel, such as with continuous instruction fetches by one CPU.

If the relocation action is active, relocation requires another 150 nsec if the address is found in the associative memory; 2100 if not. Since the basic CPU cycle rate is 200 nsec, the CPU clock is actually stopped for 150 nsec ("stuttered") to allow for the associative compare. The clock is blocked for 2.1 microseconds if no valid associative compare occurs while the page table entry is fetched and loaded into one of the associative registers. Since the instruction counter is kept in relocated form, a relocation delay occurs only during a branch in-

struction or when the instruction counter crosses a page boundary, Input/output accesses are also not relocated, so that the 150 nsec relocation delay applies only to memory accesses for data.

The maximum memory cycle time would therefore be 750 + 150 + 150 or 1050 nsec, assuming no interleaving, assuming a different CPU requests service each time, and assuming the only accesses are for operands. The effective cycle rate is considerably less than 1050 nsec and depends on the instructions used, the location of data and other program dependent factors.

Figure 14 shows the floor plan for the two-CPU four-memory configuration. Because some memory

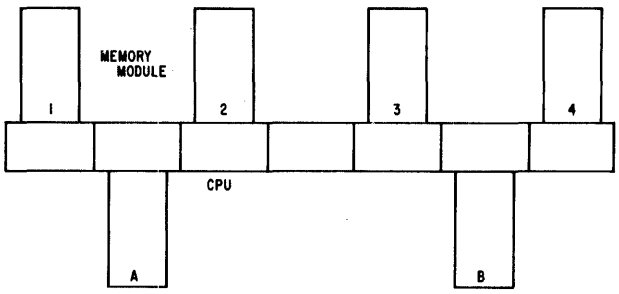


Figure 14. Floor plan of typical Model 67.

modules are physically more distant from some CPU's, the signal travel time is increased, and memory access time is degraded. For the above system, the following table shows the additional memory times in nanoseconds due to table length.

	To Memory Module:			
From:	1	2	3	4
CPU_A	0	0	50	100
CPU_B	100	50	0	0

Because this is a symmetric multiprocessing system with one copy of the supervisor in core, no attempt is made to optimize its location in a "midway" memory module. Therefore, the cable delay for memory fetches may be considered roughly an average of these figures, or 38 nsec for a 4-memory system.

To achieve a single average overall memory access figure would be difficult because of the assumptions about the program that must be made. For purposes of discussion, including average cable length delay, including priority and relocation delay for operands and taking advantage of some interleaving of instructions a conservative figure of 800 nsec will be used.

Data Rates

The data paths between CPU's, memories, Channel Controllers and channels are eight bytes wide. Between the channel and I/O control units, the path is one byte wide. Parity is on the byte level. The data rates and bandwidths for the devices and channels are shown in Table 3 for the configured system. The CPU in a typical program makes 540 storage references in a 1000-microsecond interval. This means a data rate of 540,000 double words per second, or 4,320,000 bytes per second.

A conservative memory access time for a three-memory module Model 67 system, as shown in Fig. 1, is 800 nsec. Since the four storage bus systems are independent, all three storage units may be executing storage cycles concurrently, thus resulting in an effective storage data rate of three double words per 800 nsec., or 30,000,000 bytes/second. The table assumes concurrent use of all three storage units and shows that the memories are being accessed at an average rate of 36% of their capacity. To be sure, if both CPU's and all I/O activity refer to the same unit, the bandwidth is exceeded and CPU operation is delayed. However, in no case will the I/O activity be restricted by the channel or memory bandwidths.

A simulation run was made with only one processor executing the instruction mix and without any cable or priority determination delay. This run was used as the base run. Next, a model assuming four memory modules and two CPU's was simulated. All memories in the model have an equal chance of being selected regardless of the device making the request or the number of other devices contending for the storage unit at that time. A random amount of interleaving is assumed, and an instruction mix which uses 42% of the available memory cycles was assumed. Cable and priority delays are included. The following table giving the simulation results of the model relative to the simplex Model 67 as a function of the I/O data rate shows the expected system degradation. Degradation is defined as the expected increase in job run time.

I/O Rate	System Degradation
0 megabytes	8.8%
1.6	9.7
3.2	10.8

The data rate of 1.6 megabytes, is about average for the configured system with the drum and disks

Table 3. Date Rates and Bandwidths

Data Source	Max. Data Rate (system as configured in Fig. 1)	Max. Data Rate (bandwidths)
2301 Drum (4×10^6 bytes cap.)	1200 kbps*	1300 kbps*
2314 Disk storage (207×10^6 bytes cap.)	312	1300
2311 Disk storage (7.3×10^6 bytes cap.)	156	1300
Total, 2860 Selector channels	1678	3900
Four 2402 tapes (90kc)	360	640
Basic 2870 MPX activity	10	110
Total, 2870 HS MPX channel	370	
Total, 2846 channel controller	2048	—†
CPU A, average access	4320	
CPU B, average access	4320	
Average CPU activity	8640	
Total memory access	10,688	30,000

*Thousand bytes per second.
†2846 data rate capacity exceeds requirements for this configuration. Exact rate to be determined.

in operation. Nevertheless, the additional degradation of 0.9% in job run time is small. The overall 9.7% includes the priority and relocation delays and is the penalty to be paid for the advantages in the flexibility of shared memories and throughput increase with multiprogramming.

GROWTH

The configuration shown, although typical, is by no means the only one possible. System/360 Model 67 was planned with growth and flexibility in mind. As experience is gained with the time-sharing system, better systems balances may be obtained as a function of the type of applications. For example, an increase in memory size may very well be the primary requirement to achieve higher throughput in a certain installation.

The minimum time-sharing system consists of one processor, one memory and three channels. The independent memory modules of 256k bytes each may be increased to eight. Each module may have up to eight tails for eight independent bus systems. The total on-line directly addressable 750-nanosecond storage thus becomes 2,097,152 bytes. The system may grow from one to four CPU's.

The system may expand to include four IBM 2846 Channel Controllers. Each controller may have up to six selector channels, four high-speed multiplex subchannels, and 192 low-speed subchannels, as long as the maximum allowable data rates listed above (under "Other Features") are not exceeded.

Each selector channel may have up to eight control units and 256 I/O devices. Thus, the I/O capacity of the Model 67 is almost unlimited. Presumably with the maximum number of devices, control units, channels, and channel controllers, the total number of I/O devices that may be connected to a multiprocessor Model 67 is 7168.

The Model 67, when operating in the nonrelocate mode, is completely compatible with the rest of System/360. In fact, a simplex Model 67 operates identically as a Model 65 when running in the nonrelocate mode. Therefore, Operating System/360 can run on a complete or partitioned Model 67. The Time-Sharing System/360 will have a monitor, conversation FORTRAN and assembler, PL/1, COBOL and sort-merge. The TSS monitor will support as terminals the standard IBM 1050 and 2741 "Selectric" typewriters, as well as 2250 and 2260 display units. Eventually remote digital and analog devices will be tied in and will operate with the Model 67 in a time-shared, data-logging mode. It is expected that conservatively at least 100 termi-

nals can be active at one time in a two processor system such as described here.

In one installation, it is planned that about 50% of the work will be conventional batch processing and will be run as "background" to the time-sharing. The on-line card readers, card punches, printers and plotters will be used with all the background jobs, and those terminal-oriented jobs which require them. The Time-Sharing System will support a SPOOL operation.

It is expected that both CPUs will be used for the time-sharing mode of operation during prime hours. At other times, the second CPU will be partitioned and used exclusively for batched jobs under Operating System/360 or else with real-time experiments.

With dynamic relocation, independent memories, a new bus system and dual data paths, the Model 67 provides a revolutionary method of operation. The hardware, when first delivered in April 1966, will be IBM's most advanced commercially available data processing system. The Time-Sharing System/360 programming system, when available later, will allow the Model 67 to fully realize its potential.

ACKNOWLEDGMENTS

The author wishes to recognize the original logical and systems design work on the Model 67 and on Time-Sharing System/360 done by Dr. A. Blaauw, Dr. E. Bertram, Mr. H. A. Kinslow and many others of IBM's System Development Division.

BIBLIOGRAPHY

Comfort, W. T., "A Computing System Design for User Service," *Proc. FJCC 1965*, Spartan Books, Washington, D.C., 1965.

"IBM System/360 Principles of Operation," IBM Document, Form A22-6821-1.

"IBM System/360 Model 67 Time-Sharing System Technical Summary," IBM Document, Aug. 1965.

"Time-Sharing System/360 Development Workbook," IBM Internal Document.

"System/360 Model 67 Time-Sharing System Preliminary Technical Summary," IBM Document, Form C20-1647-0.

A DATA MANAGEMENT SYSTEM FOR TIME-SHARED FILE PROCESSING USING A CROSS-INDEX FILE AND SELF-DEFINING ENTRIES

E. W. Franks
System Development Corporation
Santa Monica, California

INTRODUCTION

The Time-Shared Data Management System (TDMS), under development at System Development Corporation (SDC) for use in its Research and Technology Laboratory, is intended to provide the users of the SDC Time-Sharing System with a powerful set of tools for the manipulation of large volumes of formatted, that is, not free text data. The functions to be provided include the description of data, the storage of files or data bases into the computer environment, the retrieval of the data either in response to human query or under program control for processing by other programs of the system, and the maintenance of data already loaded.

TDMS is for the use of subscribers to the Research and Technology Laboratory's facility; many of these users are not professional programmers. This imposes the requirement that the system be controlled by a nonprogrammer-user-oriented language. The data management function for which TDMS is the instrument is by no means the sole function of the computer system in the laboratory. Furthermore, it operates on a time-shared basis with the other functions performed by the computer and may therefore be used simultaneously by several users. This aspect of the environment imposes a requirement to provide responses acceptable to on-line users of the time-sharing system in circumstances where there may be many users and where the volume of data from which responses are required is very large. The organization of the data is designed to optimize on-line retrieval of this kind where the criteria for selecting data from the file are not known in advance. The two aspects of TDMS emphasized in this paper are the user-orientation features and the file organization scheme.

STRUCTURE OF TDMS

TDMS is designed to operate under the control of a Time-Sharing executive *program* using IBM S/360 computers. One feature of the system is that it will be easy to adapt the system to various models of the IBM S/360 computers. Although the system will at first be designed for model *65* IBM S/360 computers, as advances are made to the computers, the system can be adapted to increasingly sophisticated versions, such as the proposed model 67. Communication with the programs is by means of the TDMS language, which is a reactive, or dialogue, type of language, whose rules of use are always available to the user on request to the program.

The control program of TDMS can be contacted through the time-sharing system by any user from either remote or local stations. The control program of TDMS will enter a dialogue with the user. After the control program has communicated the

user's requirements to the time-sharing executive and to the computer console operators, as appropriate, the user will be granted access to various functions of the system without having to be aware of the program structure of TDMS. From the user's point of view, he is dealing only with the program he called through the time-sharing system, although, in fact, in the course of performing data management operations, he may have used and communicated with as many as half a dozen different programs.

In addition to the control program, TDMS includes a data description translator, a data load program, an on-line query and update program, a report-generator program, and a maintenance program which, in addition to performing batched updating, permits the user to create a subset of a file, to merge files and to restructure files including the computation or generation of new data elements. Space and time limitations preclude presentation of a more detailed description of the programs incorporated in TDMS. However, to describe the user orientation features and file organization scheme, it is necessary to first say something about the underlying philosophy of data upon which TDMS is based.

CONCEPT OF THE ENTRY

A TDMS data file or data base, as it is often called, is a collection of information sets or entries. Each entry contains information about one object. The object itself need not be named, but may be understood from the description provided by the name in the data base itself. Thus, a data base describing the personnel in a corporation might have one entry for each employee, and yet, within that entry, it would never be necessary to state explicitly "this is the description of a person." The kinds of data collections encountered in Command Control problems and other management problems are seldom as straightforward as the example just given. Each entry in a data base describes an object, but the objects are not all of the same kind. For example, a resources file may contain entries dealing with factories and other entries dealing with training schools. It is essential, in such circumstances, for each entry to identify the object which it describes. What TDMS allows, in fact, is the accomodation of more than one logically consistent file in the same file structure. This makes the cross-coordination of different kinds of data much easier and more efficient in an on-line environment than would be the case if separate files had to be run together and matched.

The logical structure of the TDMS entry is a collection of predefined elements or descriptors. Each entry will have a subset of these elements appropriate to the object being described. For example, the entry describing factories would have an element "GROSS PRODUCT IN THOUSANDS OF DOLLARS," but would not have the element "AVERAGE SIZE OF GRADUATING CLASS OVER LAST 10 YEARS." The reverse would be true of an entry describing a training school.

The TDMS data base is not organized into sort hierarchies such as COUNTY within STATE within COUNTRY. Provision is made, however, to accomodate naturally occurring hierarchies in the data. For example, in a data base defining tactical military organization, an entry might exist for a group. The Group Headquarters, names of staff officers, mission, and so forth, would pertain to the whole group. Each company of the group, however, might be in a different location, and each might have a specific subordinate mission. One possibility of handling this situation would be to establish separate entries for each company, each containing an element labeled "GROUP TO WHICH ASSIGNED." But, because TDMS is a general system, there would be no special magic in that particular label which would enable the system to know that these entries were really part of the group description. To ensure retrieval of the whole set, the retriever would have to know of the existence of this element and to use it as part of the retrieval key expression. To solve this problem, TDMS permits the automatic association of data connected by a natural hierarchical relationship through the device called a repeating group. A repeating group is, in effect, a set of subentries which are part of an entry. Thus, the elements in an entry which belong to one of its subordinate repeating groups may have several values within that entry, but only one value within each of the subentries. The flexibility of this device is such that, on the one hand, it will accomodate a simple multivalue element like "PROFESSIONAL ASSOCIATION MEMBERSHIP"; on the other hand, an order-of-battle file with only three basic entries, ARMY, NAVY and AIR FORCE, would contain all the subordinate organizations appearing as repeating groups within the three basic entries. As this statement implies, repeating groups may themselves contain repeating groups to any level of nesting.

DESCRIPTION OF DATA

The best way to understand how data is described is to take a hypothetical example and show the process being performed. Let us imagine that our hypothetical user is a mail-order merchant who handles a variety of merchandise. Let us suppose he has access to TDMS and the SDC Time-Sharing System through a teletype machine in his office. Let us further suppose that he is so committed to using a system that he does all of his paper work on that one teletype.

His first task is to describe his data to the system. He contacts the TDMS control through the Time-Sharing Executive. He may request a list of the functions available, but, in this case, we assume that he knows that the function he wants is called DEFINE. The DEFINE program then asks him to name his data base. He responds by typing in

COMPANY OPERATIONS

From now on he is able to refer to the descriptions he will supply and to the collected data itself by this name.

The data our merchant is about to describe will exist in two forms: outside the computer system, the data will exist as data input; inside the system the data will exist as the stored file. The following conventions exist for input data. The data input to TDMS always exists as card images on tape, or as input entered by teletype, but the scheme is the same in either case. Each data element is preceded by an identifying number field, and the set of elements and repeating groups constituting one set or entry is terminated by a special symbol selected by the user. The sequence of the elements is immaterial except that the elements in a repeating group must all be listed before the elements of the next repeating group or nonrepeating element. Thus, after receiving the name of the data base, the DEFINE program asks the user for the terminating symbol. In this case, let us say that the user chooses the term ALL. When the data is loaded, the system will know that whenever the term ALL is encountered in the input, and the term is not preceded by an identifying number, the last of the data for a particular entry has been received.

The user now proceeds to name the various elements of data he will be dealing with. He may request the system to spell out the rules for the description process, and, if he is uncertain of his typing skill, he may request the ECHO function, under which the program types back what he has input, giving him a chance to make corrections to what he has just typed before proceeding.

The elements of data are listed one at a time on the teletype. First the identifying number which will appear on the input is given. Then the name of the data element is stated. Following the name is the specification of the data type in one of the following three possible types:

NAME	(alphanumeric character string)
INTEGER	
DECIMAL	

The names of repeating groups, or subentries, are given in the same way—first the identifying number, then the name of the repeating group, then the term REPEATING GROUP (abbreviated as RG). Data elements within a repeating group are specified like other data elements, except that, following the data type specification, the name of the repeating group to which the element belongs appears. Thus the order in which elements are described does not matter. Finally, as an option, input legality check information may be inserted. This information may be a list of acceptable values, one or more ranges of values for numeric data, or a data format description. A list of some of the data input elements entered by the hypothetical merchant is shown in Example 1, below.

Example 1

```
 1 ENTRY TYPE (NAME) VALUES
       CUSTOMER PRODUCT
 2 CUSTOMER NAME (NAME)
 3 CUSTOMER CATEGORY (NAME)
       VALUES ACCOUNT PROSPECT
 4 ACCOUNT SYMBOL (NAME) FORMAT
       L999
 5 PRODUCT (NAME)
 6 ACCOUNT HISTORY (RG)
61 DATE OF ORDER (NAME IN
       ACCOUNT HISTORY) FORMAT
       09[/]99[/]99
62 AMOUNT OF ORDER (DECIMAL IN
       ACCOUNT HISTORY)
63 BILL OF MATERIAL (RG IN
       ACCOUNT HISTORY
631 MERCHANDISE (NAME IN BILL OF
       MATERIAL)
632 QUANTITY (INTEGER IN BILL OF
       MATERIAL)
633 UNIT PRICE (DECIMAL IN BILL OF
       MATERIAL)
634 ACTION (NAME IN BILL OF
```

```
      MATERIAL) VALUES SHIPPED
      BACK/ORDER
635   SUBTOTAL (DECIMAL IN BILL OF
      MATERIAL)
636   STOCK CODE (NAME IN BILL OF
      MATERIAL) FORMAT LL999
  7   ADDRESS (NAME)
  8   CURRENT STATUS (NAME) VALUES
      PAID OPEN
  9   BALANCE (DECIMAL)
 10   CODE (NAME) FORMAT LL999
 11   WAREHOUSE (NAME)
 12   UNITS ON HAND (INTEGER)
 13   UNITS ON ORDER (RG)
131   NUMBER ORDERED (INTEGER IN
      UNITS ON ORDER)
132   SOURCE (NAME IN UNITS ON
      ORDER)
133   ORDER NUMBER (NAME IN UNITS
      ON ORDER) FORMAT 009LLL
134   COST (DECIMAL IN UNITS ON
      ORDER)
```

The list is, of course, by no means the complete set of elements which would be required for the hypothetical operation. It is sufficient, however, to illustrate the features of the descriptive language.

The data base contains two types of entries—customer entries and product entries—so that billing and mailing operations and inventory control operations can be performed from the same file. The first element described, ENTRY TYPE, specifies which kind of data is included in a particular entry. Only two values are possible for this item of data, namely CUSTOMER and PRODUCT, and these values are listed, following the word VALUES for checking the legality of input. Input Element 2, identified as input by a field containing the number "2" preceding the data value, occurs only if the entry is the customer-type entry. The third element, also applicable only to the customer-type entry, shows a distinction between actual customers (value ACCOUNT) and hoped-for customers (value PROSPECT). The PROSPECT entries would be entered for mailings or in response to queries. Although no such elements are shown in the example, data about correspondence, brochure mailings, and areas of interest would probably be included in such entries.

Element 5, PRODUCT, is the first element described which would be applicable to the inventory type of entry. The next element in this category does not occur until Element 10, the product code. The legality check for this element is a format check. The L's stand for letters and the nines for numbers. Thus value AA010 would be a legal product code and value A33 would not. An additional example of format control is shown in Element 61, where the slashes in the data are enclosed in square brackets, indicating that these exact characters must occur. The example shows several repeating groups. The first of these, ACCOUNT HISTORY, occurs in customer-type entries. The user has chosen to number the inputs for the repeating group 6 as 61, 62, etc. This is an example of a user-devised convention, and is not required by TDMS. The repeating group, ACCOUNT HISTORY, itself contains a repeating group, occurring for each order recorded; this repeating group is BILL OF MATERIAL. The third repeating group is Element 13, UNITS ON ORDER, which relates to the inventory type of entry.

The list given is merely the description of the data given to TDMS. It is not the data itself. To clarify the significance of the description, two examples of input data are given below, one for a customer and one for stock.

Example 2

Input Data for a Customer-Type Entry

```
1)   CUSTOMER   2)   JOHN Q JONES
     3)   ACCOUNT
4)   J021   6)
     61)   5/21/64   62)   205.63   63)
     631)   TABLECLOTH   632)   17
          633)   5.40   634)
          SHIPPED
     635)   91.80   636)   TC301   63)
     631)   PLACE SETTINGS   632)   3
          633)   37.81   634)
          SHIPPED
     635)   113.43   636)   SV002
7)   2000 LONDELIUS ST LOS
     ANGELES CALIFORNIA
8)   OPEN
9)   105.00
```

Example 3

Input Data for a Product-Type Entry

```
 1)   PRODUCT
 2)   TIKI FIGURES
10)   TK000   11)   ZELZAH   12)
      205   13)
131)   50   132)   PORYNESHA KK
      YOKOHAMA
133)   127PKK   134)   410.00
```

When the user has finished entering his description, he may have it presented to him for checking. He may add, delete, or change a description already made at any time by calling the REDEFINE function. He needs to know very little about the operation of the program or about computers. Technically, he need only know that data may be numeric or nonnumeric. The logic of the organization is the logic of the data itself as it appears to him. Once the data is described in these user-oriented terms, he may load the data into the TDMS system at any time by calling the LOAD program. The LOAD program expects inputs which agree with the description given. Discrepancies are logged, and the user may have them logged on-line for immediate correction. Once the data is entered, it may be called by name, and again the user does not need to know how it is stored or accessed. He may perform spot queries for fact retrieval. He may describe formats of output and call a report generator to make up bills or bookkeeping summaries. In the example given, the inventory is presented as it would appear if listed by product within a warehouse. He may want to obtain summaries by product, regardless of warehouse location, or he may want a summary of all products by warehouse. He is not restricted by the organization implicit in the way he has chosen to define his data. This freedom results from the way the data is actually organized in the computer.

TDMS DATA BASE STRUCTURE

The data structure created by TDMS when the input data is loaded is designed to optimize retrieval in an on-line environment if it is assumed that the user has no prior knowledge about what data is most likely to be retrieved or what criteria will be used to select data for retrieval. It is also assumed that retrieval will be requested from the file on the basis of some Boolean expressions given in terms of data elements and values. Such Boolean expressions define a subset of the data base, namely, those entries in the data base for which the Boolean expression is true. Frequently, however, instead of requiring the entire contents of this data subset only certain values from the qualifying will be needed. Thus, the selection path is entered with a combination of element names and values associated with them. This defines a list of entries which meet the criteria. The retrieval path is then entered with a list of entries and a list of element names for which values are required from these entries. The data base organization is designed to optimize both the selection of qualifying entries and the retrieval of the specified element values.

From the data user's point of view, the data base appears to be a collection of values to which he may wish to refer. These values have two sets of associations. In the first place each value is part of the total description of one of the objects in the data base. In the second place each value is both a value for a specified element and a member of the set of all values for that element. The TDMS organization of the data base reflects both types of value association, the element set and the object set.

The actual values are stored according to the element set relationship. That is, for each element there is a block of storage for the unique values occurring for that element. Each value is stored only once, regardless of how many entries of the input data may contain it. Associated with these lists of unique values are two other groups of lists. The first of these has the function of ordering the raw value list algebraically, or in the case of symbolic values, alphabetically. The items on the value list are stored in a random arrangement; the value list orders the values in the sequence in which they appear in the input. The ordering list is generated to speed up search by permitting the use of binary search techniques. The second group of lists associated with the blocks of values is the entry group. For each entry there is a list of the elements which were found in the entry and a reference to the place in the value list for that element where the specific value for that entry may be found. This part of the organization is represented schematically in Fig. 1.

For the purposes of selection, the ordering lists, in addition to pointing to values in the value list, also point to occurrence lists. The occurrence lists are lists of entries in which each value of each element occurs. Thus, in order to make a selection on the basis of a Boolean expression, the ordering list is searched for values which meet the various criteria. When a matching value is found, a list of entries containing this value is obtained. The various lists obtained for different parts of the Boolean expression are merged, using AND or OR logic; the result is a final list which represents a subset of the data base that meets the criteria for selection. This list of entries is then used in combination with the names of the elements to be retrieved to obtain the values of these elements from the appropriate value lists.

The entire retrieval process becomes clearer if we take an example. Let us imagine that our mail-

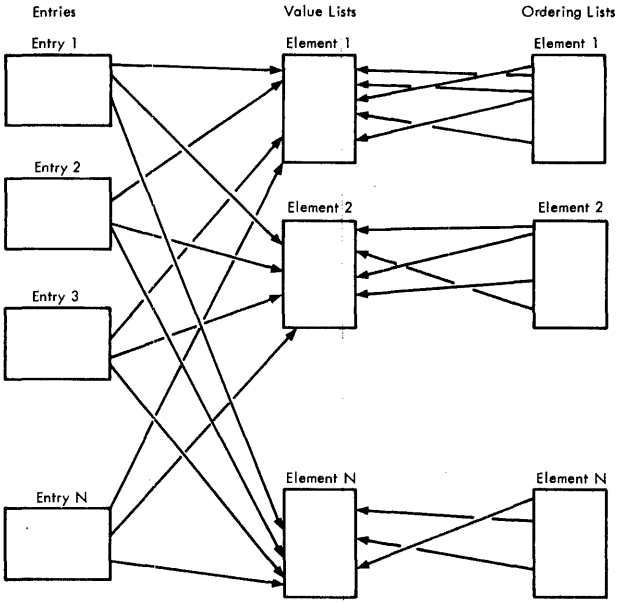


Figure 1.

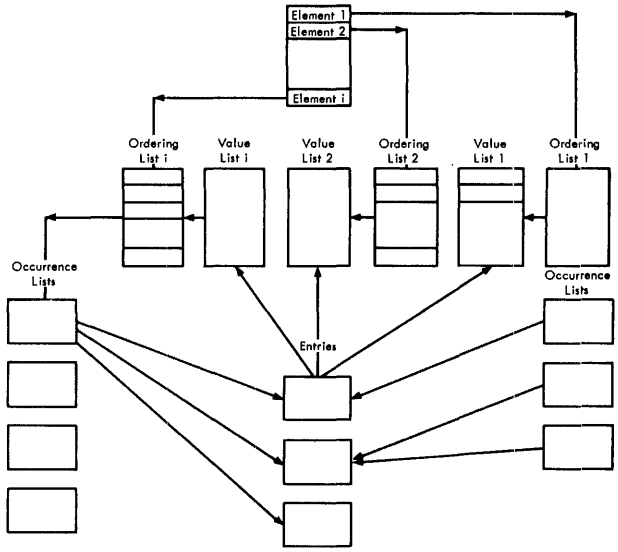


Figure 2.

order merchant wishes to obtain a list of the customers who currently owe him money—an accounts receivable list. He might do this through the TDMS QUERY program with the following request:

PRINT CUSTOMER NAME, BALANCE
WHERE CURRENT STATUS EQUALS OPEN

The subset of the data base to be selected is the set of entries which have the value OPEN for the element called CURRENT STATUS. Through the Data Definition Table the program converts the name CURRENT STATUS to the address of its ordering list. This list is accessed, and is found to have only two entries, one for the value PAID and one for the value OPEN. The value OPEN points to a list of the entries which have the value for CURRENT STATUS. Then entry references are converted to entry list addresses by means of a directory table. The qualifying entries are accessed, and, for each one, the pointers to the value lists for the elements CUSTOMER NAME and BALANCE are followed, and the values are recovered and printed.

The query in the above example is a simple and straightforward one, not involving AND and OR logic, and not concerned with the complexities of nested repeating groups. It does serve, however, to introduce the entire data base structure as created by TDMS when the data is loaded. Figure 2 shows this structure schematically.

The route is from definition table to ordering list. The ordering list permits an efficient examination of the value list for qualifying values. For each qualifying value there is an occurrence list. The occurrence list points to the data base entry where the value occurs. Each entry, in turn, points to the value lists for all the elements present in it. The occurrence list actually points indirectly to the entry via a directory of entry addresses. In cases where a value occurs only once for a given element, the ordering list bypasses the occurrence list and points immediately to the directory to save storage space.

So far little has been said about the directory table, which, in its simplest form, simply lists the address of each entry in the sequence in which the entries were loaded. In cases where repeating groups are involved, however, the directory assumes greater importance, since it is here that the hierarchical restrictions imposed by repeating groups are observed. This concept is best explained by means of an example. Again using the merchandising data base, let us imagine that the user wishes to obtain a list of customers who have ordered sardines in quantities of 100 cans or more as part of an order totaling $100 or more. In this case, "sardines" is a potential value for the element MERCHANDISE and "100 cans or more" represents a potential value for the element QUANTITY. Both elements occur in the repeating group BILL OF MATERIAL. Both must occur in the same group. In other words a value of 100 for QUANTITY is not sufficient to qualify the entry unless it is directly as-

sociated with the value SARDINES for the element MERCHANDISE. Furthermore, the co-occurrence of these two values does not qualify the entry unless the total order of which the sardines are a part equals or exceeds $100. It is necessary then to find entries in which one of the values for AMOUNT OF ORDER in the repeating group ACCOUNT HISTORY is equal to or greater than 100 at the same time as the values for MERCHANDISE and QUANTITY in the BILL OF MATERIAL for that particular order meet the criteria respectively of "sardines" and "equal to or greater than" 100.

This complex matching problem is solved by carrying an entry for each repeating group, as well as for each data base entry proper in the directory table. In the case of directory entries for repeating groups, instead of an entry address there is a reference to the directory entry for the next higher level in the hierarchy. In this way it is possible to determine whether values for two elements of the same repeating group (in the example SARDINES and 100 or more cans) actually occur together. If they do, the references from their respective occurrence tables will be the same. Then, by following the pointer from this directory entry to the next higher level, namely to the particular order in ACCOUNT HISTORY to which it belongs, it is possible to see whether or not the total order was equal to or greater than 100 dollars. The entry containing this information will qualify if an entry number in the directory for an occurrence of AMOUNT OF ORDER greater than or equal to 100 dollars is the same as the next higher entry pointed to by the directory entry meeting the MERCHANDISE and QUANTITY criteria.

The fact that the elements dealt with are parts of repeating groups is determined by the selection program from the definition table, as is the relative level in the hierarchy of each repeating group. In summary, the following is the path followed in response to a request phrased

```
PRINT CUSTOMER NAME WHERE
    AMOUNT OF ORDER GQ
100 AND MERCHANDISE EQUALS
    SARDINES
AND QUANTITY GQ 100
```

The program determines that the selection criteria elements are members of repeating groups. Starting at the highest level where the element appears, it accumulates a list of entry numbers in the directory table, for which AMOUNT OF ORDER qualifies. It then accumulates a list of entries for MERCHANDISE equal to SARDINES and a list of entries for QUANTITY greater than or equal to 100. These last two lists are ANDed together to eliminate entries with insufficient sardines and entries with a sufficient quantity but the wrong merchandise. The resulting intersection list is then converted to the next higher level by substituting the "up" pointers from the directory. The converted list is then ANDed with the AMOUNT OF ORDER list to produce a list of fully qualifying entries. This is not, however, the final step, since the AMOUNT OF ORDER list contains entries for a repeating group, ACCOUNT HISTORY. This must be converted to actual entry references again by substituting the "up" links, which now results in a list of basic entries. These entries are retrieved, and the output values, in this case, CUSTOMER NAME, are retrieved and printed exactly as in the first simple example.

BACKGROUND

The data handling techniques of TDMS have evolved over several years of research and experiment conducted at SDC, and the new system benefits from experience gained elsewhere. In particular, the idea of the cross-reference file was developed and tested in an experimental data management system called LUCID, and was refined and expanded in TSS-LUCID (Time-Sharing LUCID) which is currently in operation at SDC on the IBM ANFS/Q32 computer under the Time-Sharing System. The cross-reference file is that part of the data structure which consists of the value lists, the ordering lists and the entry directory table. The concept, and indeed, the name of the repeating group is derived from the ADAM system of the MITRE Corporation. The inadequacy of LUCID in dealing with the natural hierarchies occurring in data prompted this borrowing. What is entirely new in TDMS is the entry structure which has been termed the "self-defining entry." In LUCID the entry association is determined solely from storage juxtaposition. The values are tightly packed in the entries. The values also occur in the value list, thus duplicating storage requirements. Furthermore, much additional storage space is required to accommodate bits of storage assigned to data elements not actually present. In the case of multiple value elements with assigned bit locations, this arrangement requires a great deal of space for empty data

base storage. Most important, however, in motivating the development of the new structure, are time considerations. In a general system the locations of packed data elements are known to the program through parameter tables. An average of more than 100 machine instructions required to convert such parameters to an actual retrieval, and, in iterative operations, the process is likely to be very slow. TDMS does away with packing parameters so that everything is standardized, and in all probability, the number of instructions required to move a data element is always fewer than ten.

OTHER OPERATIONS

The heart of TDMS has been described in some detail. The data description language has been presented to give some idea of the essential simplicity of the approach to the system and, thus, of its suitability for nonprogrammer users. The operation of retrieval has been explained with some examples of the on-line query language being used as illustrations. The retrieval mechanism is the same throughout the system, whether it is triggered by an on-line query or by the execution of a report-generator function. What has not been covered is the arithmetic capability of the system. The extent of this capability is illustrated by the following query which shows that the system accepts arithmetic expressions involving elements of data both as output specifications and as selection criteria.

```
PRINT SUM OF HOURS WORDS
    * HOURLY WAGE
WHERE 1966 - BIRTHDATE
GR 21
```

To optimize operations such as the above without sacrificing efficiency in cases of simple retrieval, the value lists for numeric elements contain both the symbolic form of the values originally input as well as binary representations of them in either integer of floating point format. In this way the original value can be retrieved and printed without going through a conversion routine, and arithmetic and magnitude comparisons can be made in the binary mode.

CONCLUSIONS

TDMS is a generalized system which makes no a priori assumptions about the way in which the data will be used. In cases where this is known, the data can be converted to the more conventional hierarchical format by the maintenance program so that the efficiency of specific usages can be maximized. Nevertheless, the basically general approach is sound. The life expectancy of a special-purpose data management program is short, and in terms of cost effectiveness, likely to be very poor. Our experience has been that the collectors and users of data approach their problems initially with a somewhat vague and largely intuitive notion of the uses to which a data base will be put. It is only as they begin to use the data that its full utility becomes apparent. TDMS is an attempt to give users a facility which does not preclude the easy and inexpensive evolution of data management procedures, and which, at the same time, is remarkably efficient as generalized programs go. It is designed for the nonprogrammer user. We do not like to say it is for the unsophisticated user, because the more sophisticated he is in the terms of his own data and his own problems, the better TDMS will serve him.

AN ANALYSIS OF TIME-SHARING COMPUTER SYSTEMS USING MARKOV MODELS*

J. L. Smith
Systems Engineering Laboratory, The University of Michigan
Ann Arbor, Michigan

INTRODUCTION

The development of RQA[1] (Recursive Queue Analyzer), a program for the numerical solution of the stationary distribution of large scale Markov processes, has made possible the accurate analysis of large stochastic systems with modest computational costs. In particular, time-shared computer systems with their random program and user characteristics are examples of systems which can be modeled as multidimensional Markov queueing processes and analyzed by the method. Having obtained a solution for the limiting state probabilities of the model using RQA, one can readily derive many time average performance and usage characteristics. Thus a useful tool is available to provide guides in the design and modification of such systems and to forecast user response and system capacity in terms of the number of users and the operating statistics.

In current time-sharing systems the major problem is the sharing of high-speed memory. Economic considerations have limited the availability of high-speed memory from which user programs can be executed. Hence large capacity slower access time memories have been added to these systems in a manner which allows most efficient use of the high-speed memory.[2] In effect there may be many levels of memory with capacity and access time increasing as we go down the levels, and there will be continual transfer of information between these levels. Thus queues arise not only for the use of the central processor but also for the use of high-level memory and data channels. In practice only a certain number of user programs can be allowed to occupy the highest levels of memory without serious reduction in some performance criteria. Useful models depict the important queueing phenomena in this regard.

We now proceed to describe the process of modeling a time-sharing system and illustrate some results for a particular system.

DESCRIPTION OF A TIME-SHARING SYSTEM

Figure 1 shows a block diagram of the major hardware components of a time-sharing system which is representative of current designs using a single central processor. There are three types of memory, the high-speed core memory and memory modules A and B which represent two lower levels of memory with increasingly larger capacity and slower access times. A number of remote communication consoles and the necessary data channels for the interconnection of all components complete the hardware.

The high-speed core memory would be operated on a paging or segment and paging scheme[3,4] to allow maximum benefit from the use of common

*This work was supported by Rome Air Development Center under Contract No. AF-30(602)-3553.

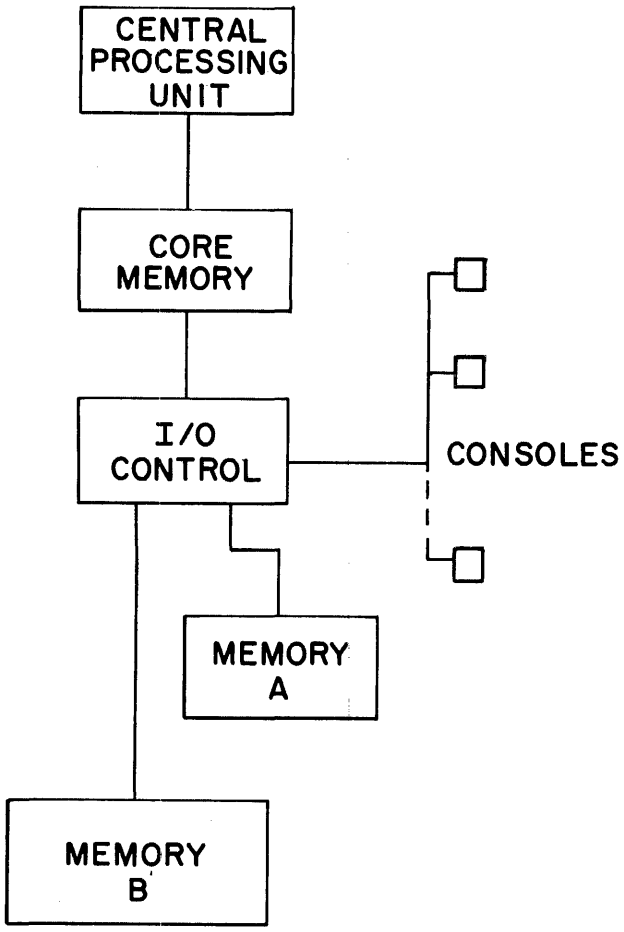


Figure 1. Block diagram of a time-sharing computer system.

routines and multiprogramming. Part of the core would be devoted to an executive program and communication tables and the remainder to use programs.

Memory module B represents a very large-capacity file store (for example several disc memory units) wherein each user of the system is assigned a storage area solely for his own use. Files of source decks and binary decks and data files are stored here. When a user is operating at a console he will activate some of these files and they will be transferred under executive program control to allotted pages of the core memory for processing. They will then be subject to swapping procedures between core memory and module A as described below.

During each user's session at a console he will use his own files, many system programs, and information generated at the console. For efficient memory sharing it is desirable that only the current working section of each user's program be resident in core and that other sections be readily available to enter core on a swapping or overlay operation. Memory module A (typically consisting of high-speed drum units) acts as a core store overflow medium to contain such currently active files and system programs. As many working sections as possible should remain in core in order to take advantage of averaging their demands on system processors. When the system is serving a large number of users one would expect frequent changes in the working section currently executing in the CPU, and continual traffic of pages or segments between memory module A and the core store.

DEVELOPMENT OF A QUEUEING MODEL OF THE TIME-SHARING SYSTEM

We will now derive a queueing model of the time-sharing system described, indicating assumptions involved and some further details of the system operation necessary to complete the model.

The executive program will include a scheduling algorithm for allotting user programs use of the CPU and other system processors. When a user program is assigned the use of the CPU we assume that it executes for a random length of time which is short compared with the average time an operator takes to interact with the system. Scheduling algorithms may induce short execution phases by penalizing long programs and assigning execution time limits. Also the following three types of events will cause random execution phase lengths.

1. Transfer of control in a users program is required to a segment or page which is not residing in core and thus pages must be swapped or overlaid from module A before execution can begin or continue.
2. Console output has been generated and the users program is ineligible to execute until some further information or command is supplied by the user.
3. A user file or program previously unused at the current session has been called by the user program and must be loaded from module B before execution can continue. Alternatively the user has elected to store a file.

Thus the program of a user currently operating at a console will always have associated with it one or more of the following phases of system operation:

1. CPU execution.
2. Segment or page swap (or overlay) from module A.
3. Operator response.
4. File transfer to or from module B.
5. Queueing for phases 1, 2, or 3.

Programs may also queue for information transfer in the console data channels, but the significant operations are the thinking and response generation by the user and these may occur simultaneously when each user has his own console.

The queueing model shown in Fig. 2 is based on these five phases of operation. The service operations corresponding to phases 1 through 4 are depicted by blocks containing a parameter ($1/\mu_1$, $1/\mu_2$, $1/\mu_3$, or $1/\mu_4$) which is the mean execution time in that phase. Wherever queues may form this is indicated by a circle containing the value of the maximum possible queue length. It is assumed that there are N consoles in use at all times. The maximum queue lengths for swapping and file transfer operations have been designated N_1 and N_2 respectively. There is also a limit on n, the sum of the entries in these two queues ($n < N$), and this limit is the maximum number of user programs which can concurrently have pages of core memory assigned to them. If the system reaches the state in which both these I/O queues are full, then there is no user program in core memory eligible to execute. Likewise it is implied that a maximum of $N_1 + N_2$ programs in the queue for CPU execution can have sections resident in core memory.

Once a program completes a CPU execution phase it is assumed that it always generates a request for one or more of the operation phases 2, 3, and 4; that is it does not simply remain in core to await rescheduling in the CPU. This assumption should be accurate when the system is operating near its maximum capacity of users, for then competition for the use of the high-speed core will result in reassignment of the core space used by programs which complete or are interrupted by the scheduler. Further, under these conditions a program waiting for a user response would almost certainly lose its core memory assignment; therefore it is assumed that any completion of a CPU execution phase which results in user activity also results in a swapping operation to reassign the pages of core memory involved. Thus we denote the probabilities with which a user program generates requests on completion of phase 1 for phase 2, phases 2 and 3, and and phase 4 by p, q and r respectively ($p + q + r = 1$). These probabilities would be functions of the program statistics and the system operating rules.

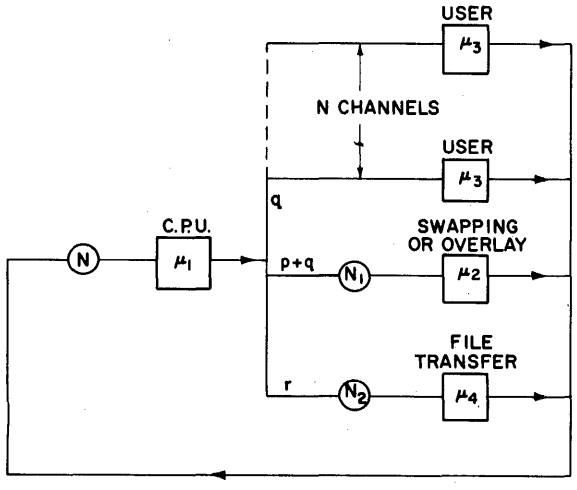


Figure 2. Queueing model of the time-sharing system.

It is assumed that each CPU execution phase involves some executive program execution for scheduling, monitoring interrupts, setting up I/O operations, etc.

ANALYSIS AND INTERPRETATION OF THE MATHEMATICAL MODEL

It is emphasized that the only property of a mathematical model required for its solution by RQA is that it be a Markov process consisting of a closed class of states. There is an upper limit imposed on the number of states of the model in accordance with storage restrictions of current computational facilities. For details of the RQA program and the theory underlying this approach to stochastic system analysis, the paper given by V. L. Wallace and R. S. Rosenberg at another session of this conference should be consulted.

The results which can be obtained from analysis of the model described using RQA are of the following nature. Assuming or given statistics on the programs, user responses and data transfers, we can examine relationships such as the response received by each user versus the number of users, or the change in this response which can be obtained by controlling the usage statistics or increasing processor capacity. To analyze the model we must first develop a state description (for example n_1 programs awaiting CPU execution, n_2 programs awaiting operator response, n_3 programs awaiting page transfers and n_4 programs awaiting file transfers, so that the four variables n_1, n_2, n_3, n_4 describe the

state of the system). RQA uses a transition intensity matrix to solve for the stationary probabilities of the system existing in any state. The allowable state transitions of the model are determined by the system constraints and the transition intensities are a function of the model parameter values. There is some restriction on the probability distributions of the model describing the random execution times. These must be of a particular class of distributions satisfying the Markov property and so are based on the exponential distribution (although some exceptions can be made[5]). A considerable range of distributions derived from the exponential distribution exist with properties useful for modeling real systems. In general it is the mean values of these distributions (e g., $1/\mu_1$) which appear in the transition intensity matrix, although for the derived distributions more than one parameter is necessary.[6]

The parameters of the model must be related in a simple manner to measurable system characteristics. A meaningful subdivision of user operation at a console has been defined as a user interaction.[7] This is the act of a user requesting and receiving service from the system and involves the user thinking, generating an input, waiting for system response and observing the output. It has been proposed that the number of interactions during a console session is a good measure of the amount of useful work accomplished by a user. In the model a new interaction begins each time a user program enters phase 3 at some console. Thus the following characteristics defined in terms of the model parameters are useful in describing the system:

$\frac{1}{\mu_1 q}$: Mean CPU execution time (plus executive overhead) for a user program per user interaction.

$\frac{p}{q} + 1$: Mean number of swapping or overlay operations per user program per user interaction.*

$\frac{r}{q}$: Mean number of user file transfers per user program per user interaction.

Other characteristics can be equated to individual model parameters.

Some measures of performance and system usage readily calculated from the vector of stationary probabilities are essential for interpretation of this solution. Two parameters which measure different aspects of the system response to a user are now defined.

1. *User busy fraction:* The average fraction of each interaction period that a user is busy, that is, making a response.

The absolute value of this parameter depends on the relative mean values of the user program processor times and the individual user's mental and physical response times. However it probably represents the average user's subjective evaluation of the system.

2. *User program response:* The average fraction of the total time a user program is eligible to use system processors (CPU and data channels) that it does actually use them.

This parameter is a measure of the overall queueing delays experienced by user programs. If there were only one system user, this parameter value would be 1.

A measure of the total useful system output is given by

3(a). *Program throughput:* The rate of completion of user programs.

or

3(b). *Interaction rate:* The rate of completion of user interactions.

Finally, parameters indicating the fraction of time various processors of the system are in use identify capacity limitations.

A PARTICULAR APPLICATION

We now discuss the use of several variations of the model proposed above ("Development of a Queueing Model") for the analysis of a small special-purpose time-sharing system.

The system corresponds to Fig. 1, and the accompanying description with the following restrictions. Memory modules A and B are the one unit (a disc file) using the same data channel. The core is not operated on a paging scheme but it is segmented on a coarse scale with hardware protection between segments. Segments are assigned to individual user programs and during execution these programs generate, under executive control, I/O operations with the disc file. These operations consist of the transfer of fixed-size blocks of information and they can be considered equivalent to page swaps in the general model. There is only a small number of

*This characteristic should show considerable dependence on the number of pages of core storage allotted to a user program.

users so that in general each user program can be assigned a segment of core memory which is not reassigned during most of his interaction periods. Most user programs will be executed repeatedly to perform information storage and retrieval or command and control functions. These programs will be loaded from the disc file as required.

The loading of a new user program in this system corresponds to a file transfer in the general system. Additional tables to be transferred in the loading operation increase the time of use of the disc data channel to an average of approximately two seconds for the total loading operation. Fixed-size blocks of information transferred in the swapping or overlay operations instigated by a user program have an average transfer time of 150 milliseconds. As the same data channel is in use for loading new programs and swapping operations the executive I/O scheduling algorithm must resolve conflicts by priority. Two possible alternatives for the algorithm, priority to loading operations and preemptive priority to swapping operations, were incorporated in different models and their effect is illustrated in the results.

The phases of program execution previously defined have the same meaning in these models once we equate file transfers to the loading of a new user program. In each model we assumed that the distributions for the execution times in phases 1, 2, and 3 were negative exponential with appropriate means. These assumptions give a degree of simplicity to the models; however there is evidence[7,8] that this type of distribution is to be found in practice. Two distributions with identical means were used in different models to describe the execution time for phase 4 (program loading). These were the exponential and the second order Erlang. The Erlang distribution has smaller variance than the exponential and also gives very small probability of short loading times. These characteristics were considered representative in describing the program loading time for this system.

We now discuss the results from three queueing models which correspond to Fig. 2 with the following qualifications:

Model 1: Exponential distribution for program loading time, priority to program loading operations for use of the disc file data channel.

Model 2: Erlang distribution for program loading time, priority to program loading operations for use of the disc file data channel.

Model 3: Erlang distribution for program loading time, preemptive priority to swapping operations for use of the disc file data channel.

We will not discuss the state descriptions necessary to incorporate these details in the mathematical models.

RESULTS

Because of the essentially unchanging class of user programs in this application it is convenient to use the performance parameters given above ("Analysis and Interpretation of the Mathematical Model") and defined on a per program basis. These parameters are plotted in Figs. 3–7 as functions of the number of users. All times have been normalized to the mean transfer time involved in phase 2. The system characteristics are defined below in terms of the model parameters:

$\frac{1}{\mu_1 r}$: Mean CPU execution time (plus executive overhead) for a user program.

$\frac{1}{\mu_2}$: Mean block transfer time between disc file and core memory.

$\frac{1}{\mu_3}$: Mean user response time.

$\frac{1}{\mu_4}$: Mean user program loading time.

$\frac{q}{r}$: Mean number of user interactions per user program.

$\frac{p}{r}$: Mean number of block transfers per user program.

N : Number of users.

Most of the results given are for model 3. Figure 7 and Table 1 include results from models 1 and 2 for comparison. All the general inferences made from the model 3 results could also be made from the results of the other models.

Figure 3 shows how performance is limited as user programs generate increasing numbers of block swapping operations. For these curves the mean user program execution time is short $\left(\frac{1}{\mu_1 r} = 1\right)$, and hence the CPU is idle most of the time. When each user program makes only light use of the disc channel $\left(\frac{p}{r} = 5\right)$, we see that the user busy fraction stays high for at least three users. Nevertheless the user programs experience significant queueing de-

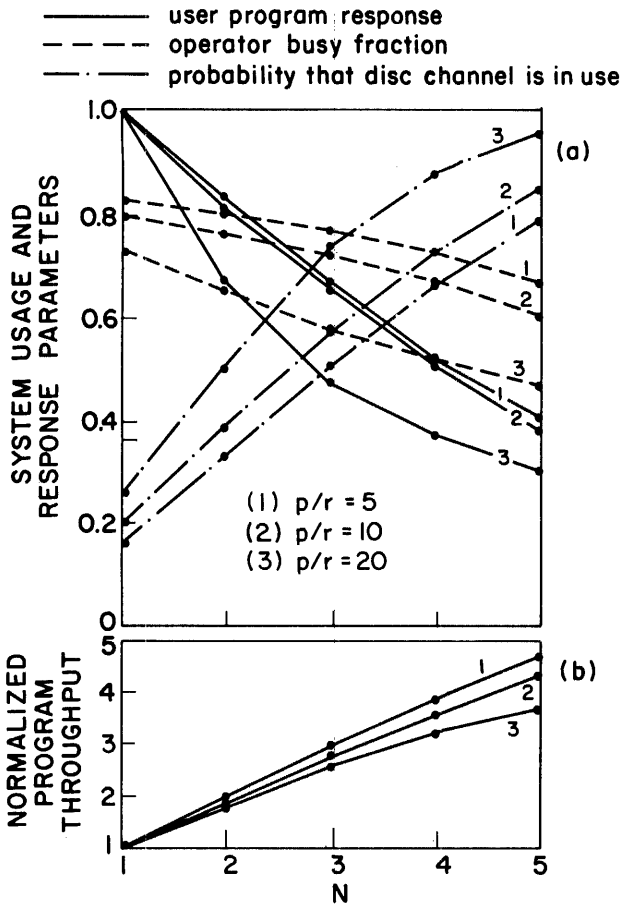


Figure 3. System performance parameters from model 3 for $1/\mu_1 r = 1$, $1/\mu_2 = 1$, $1/\mu_3 = 20$, $1/\mu_4 = 15$, $q/r = 5$.

lays (note user program response) since the increase in the number of users causes the disc channel use to rise noticeably.

When disc channel usage increases to $p/r = 20$, the disc channel becomes saturated. With five users it is in use 95% of the time, and the user busy fraction drops from 0.73 for one user to 0.46 for five users. The task program response curve indicates the queueing delays in the disc channel.

The program throughout is plotted in Fig. 3b showing that for $p/r = 5$ and five users, programs are completed at 4.7 times the rate when there is only one user. The reason is that the dominant factor in program completion rate is the user response time and users can respond in parallel. However for $p/r = 20$, the program throughput is only 3.7 for five users; here queueing in the disc channel is causing significant delays and the user response time is no longer the dominant factor in the completion rate.

For the program statistics assumed in Fig. 3 the CPU was not in use more than 5% of the time. In Fig. 5 a contrasting set of statistics has been chosen in which the CPU capacity is now the performance limit. User response time does not solely determine the completion rate even when there is only a single user of the system. Queueing delays for use of the CPU become significant as soon as there is more than one user. Performance is fairly insensitive to the range of disc channel usage chosen. Owing to the saturation of the CPU, the program throughput increases very little for more than three users, and for five users the user busy fraction has been halved. Note that the program throughput is slightly higher for $p/r = 20$ than for $p/r = 5$. The higher frequency of swapping operations for user programs increases the number of phases of CPU execution per program but reduces the average duration of each phase. The net result is to reduce queueing delays and thus improve the program throughput.

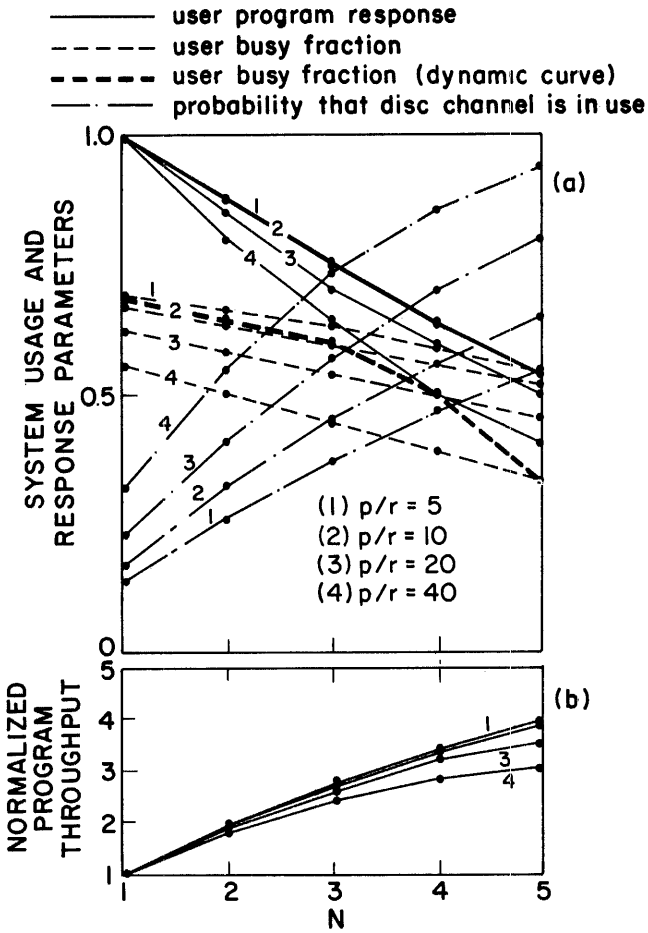


Figure 4. System performance parameters for model 3 for $1/\mu_1 r = 25$, $1/\mu_2 = 1$, $1/\mu_3 = 20$, $1/\mu_4 = 15$, $q/r = 5$.

In the discussion so far we have assumed that an increase in the number of users does not affect the number of swapping operations for each user program. However if the amount of core storage allocated to each user program is reduced as more users are allowed to use the system, the mean number of block transfers per user program p/r must increase. We can therefore treat the curves plotted for fixed values of p/r as static performance parameters, and to obtain the true performance parameters we must use a dynamic operating characteristic which gives p/r as a function of N. An example is shown in Fig. 4 where a dynamic curve for user busy fraction is plotted assuming the relationship between p/r and N given by the following table:

N	1	3	4	5
p/r	5	10	20	40

The result is a much more rapid degrading in performance with increasing N.

The aspects discussed have concerned the variation in system processor use and the resultant performance obtained by each user with different program statistics and a range in the number of users. It is apparent that the user response time in each iteration is a key factor in determining how many consoles can be serviced by this type of system. The user responses envisaged for this system are elementary so that the curves of Figs. 3, 4, and 5 correspond to rapid responses. If slower response times were expected the performance parameters for the same number of users would be considerably changed. This is illustrated by the curves of Fig. 6 where the mean response time is 2.5 times that for the other figures.

In Fig. 7 the same performance parameters are plotted as a function of the number of users for two sets of statistics. Two curves for each parameter are given corresponding to models 2 and 3. The curves

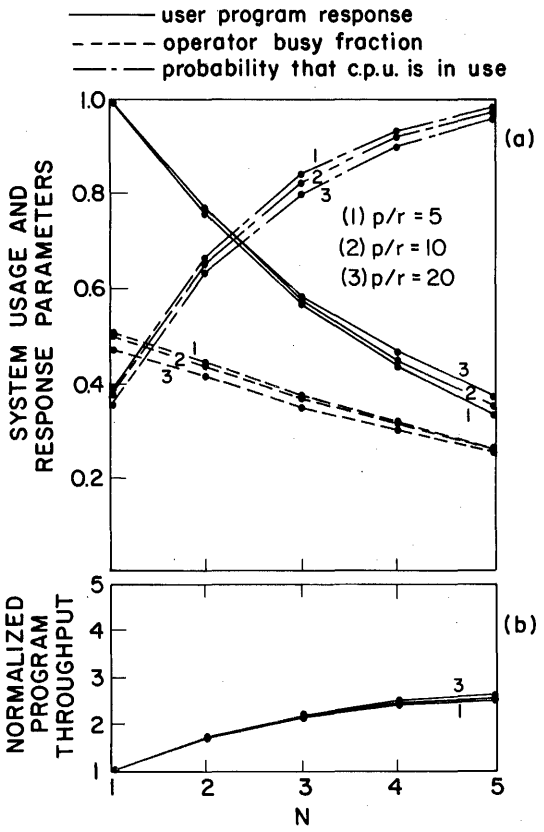


Figure 5. System performance parameter from model 3 for $1/\mu_1 r = 75$, $1/\mu_2 = 1$, $1/\mu_3 = 20$, $1/\mu_4 = 15$, $q/r = 5$.

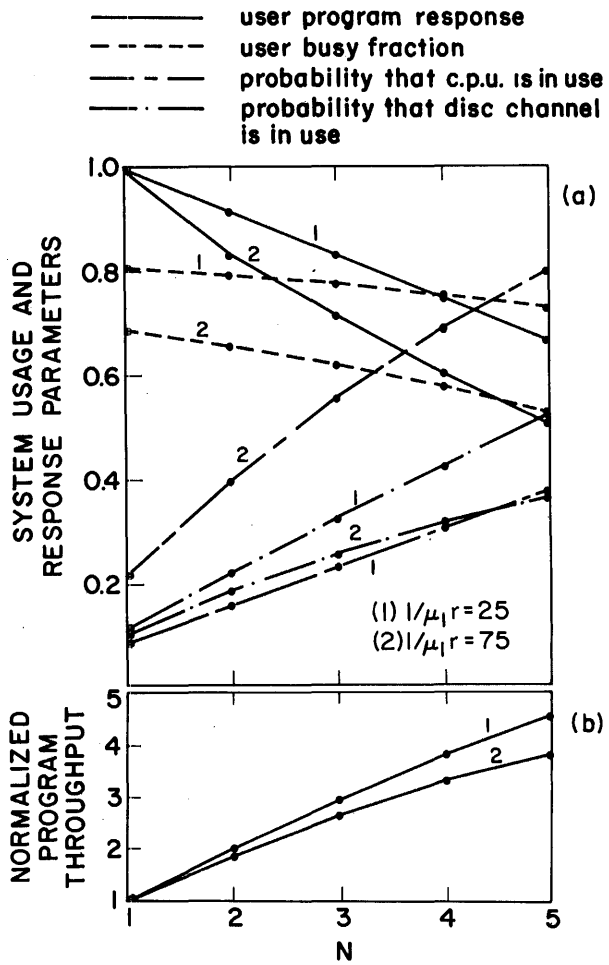


Figure 6. System performance parameters from model 3 for $1/\mu_2 = 1$, $1/\mu_3 = 50$, $1/\mu_4 = 15$, $q/r = 5$, $p/r = 20$.

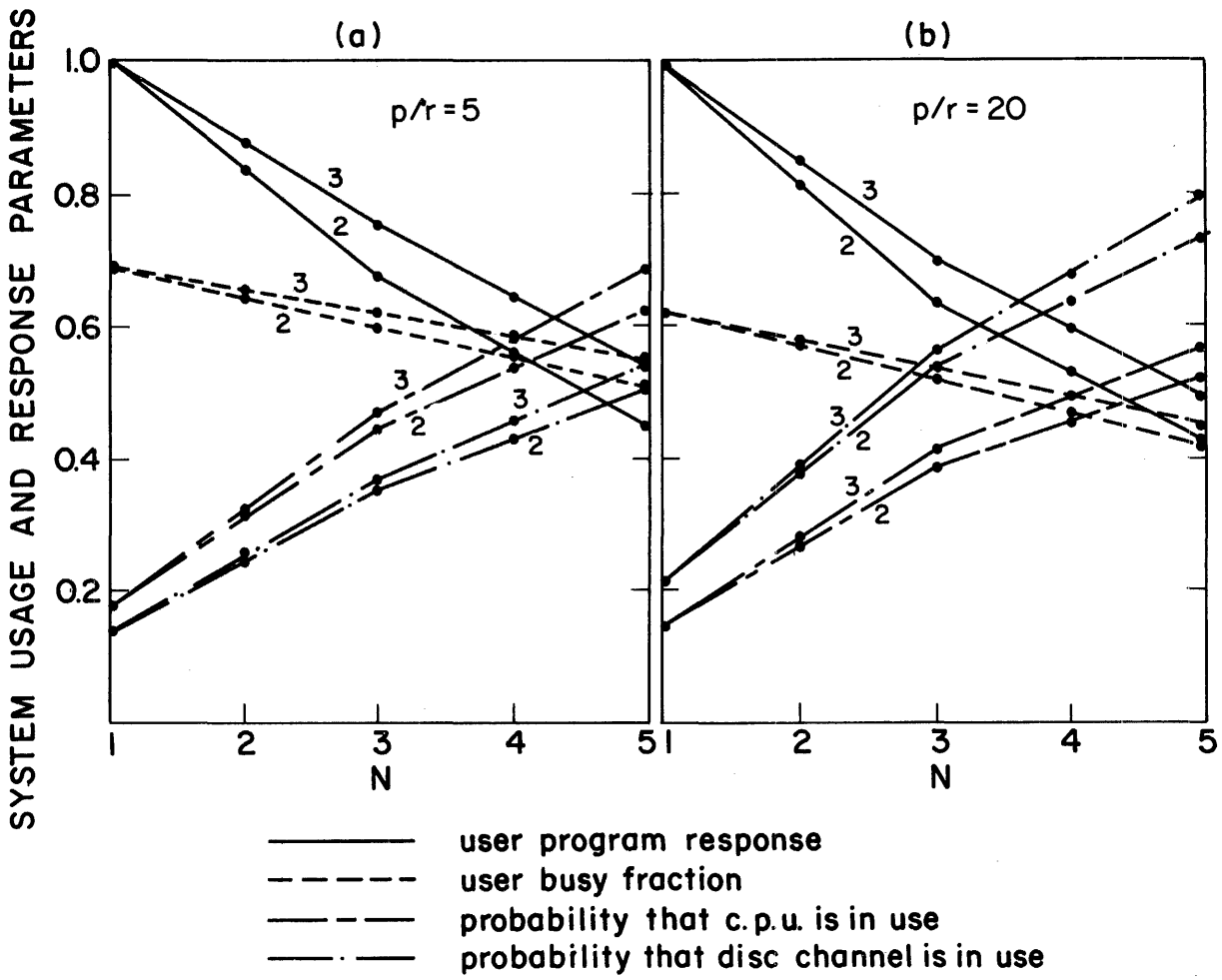


Figure 7. System performance parameters from models 2 and 3 for $1/\mu_1 r = 25$, $1/\mu_2 = 1$, $1/\mu_3 = 20$, $1/\mu_4 = 15$, $q/r = 5$.

Table 1

$\frac{1}{\mu_1 r} = 25$, $\frac{1}{\mu_3} = 20$, $\frac{1}{\mu_4} = 15$, $\frac{q}{r} = 5$

p/r	N	Prob. CPU is in Use		User Busy Fraction		Prob. Disc Channel is in Use			
						Program Loading		Swapping	
		Model 1	Model 2	Model 1	Model 2	Model 1	Model 2	Model 1	Model 2
5	1	.172	.172	.689	.689	.103	.103	.035	.035
5	3	.447	.450	.597	.600	.264	.268	.089	.090
5	5	.626	.633	.502	.507	.370	.377	.125	.127
10	1	.167	.167	.667	.667	.100	.100	.067	.067
10	3	.423	.425	.565	.568	.251	.254	.169	.170
10	5	.584	.589	.468	.472	.346	.351	.234	.235
20	1	.156	.156	.625	.625	.094	.094	.125	.125
20	3	.387	.388	.515	.517	.228	.231	.310	.311
20	5	.522	.524	.416	.418	.308	.311	.418	.419

show that the I/O strategy of model 3 (preemptive priority to block swapping operations) gives better performance for larger N. One would expect a significant advantage to be given by this strategy whenever there is frequent use of the disc channel.

In Table 1 performance parameters derived from models 1 and 2 with the same statistics are plotted. The loading process has been modeled by an exponential distribution in model 1 and by a second order Erlang distribution in model 2. For equal mean values of these distributions it is seen that the maximum difference in performance parameter values is approximately 1%. This indicates a good degree of insensitivity of these performance parameters to loading time statistics other than the mean value.

CONCLUSION

An example has been given of the use of Markov models in the analysis of computer systems. With sufficient statistics available on the user and program characteristics, useful predictions on the system performance and capacity could be made. The curve for the dynamic operator busy fraction in Fig. 4 illustrates the limitations involved in multiprogramming and the segmenting of programs. The nonlinear increase in the amount of page swapping, as the number of pages of core memory assigned to a program is reduced, is not unrealistic. As the effect on system performance is so marked it is suggested that considerable care will have to be taken in assigning a suitable working area of core to each user program.

The general model proposed for the time-sharing system of Fig. 1 involves approximations and assumptions on the operation of such a system and the probability of the service times. Experience to date has indicated that, provided one is only interested in mean value performance, liberal approximation and lumping of processing functions may be made in the modeling without changing the significant results. However it is possible to develop a more detailed and accurate mathematical model than that of Fig. 2 which is still solvable by RQA. Significant points which could be included are the interruption of user programs at discrete time intervals, specific representation of the executive program execution, and the use of multiple processors.

This modeling technique presents a useful and economic alternative to constructing general simulation models in the analysis of time-sharing computer systems.

REFERENCES

1. V. L. Wallace and R. S. Rosenberg, "Markov Models for Numerical Analysis of Computer System Behavior," this volume.

2. T. Kilburn et al, "One-Level Storage System," *IRE Trans. on Electronic Computers*, Apr. 1962.

3. J. B. Dennis, "Segmentation and the Design of Multiprogrammed Computer Systems," *J. ACM*, vol. 12, no. 4 (Oct. 1965).

4. B. W. Arden et al, "Program and Addressing Structure in a Time Sharing Environment," to be published.

5. P. M. Morse, *Queues, Inventories and Maintenance*, Wiley, New York, 1958.

6. A. L. Scherr, "An Analysis of Time Shared Computer Systems," Doctoral Thesis, Department of Electrical Engineering, MIT, June 1965.

7. E. G. Coffman and R. C. Wood, "Interarrival Statistics for TSS," System Development Corporation Document SP-2161 (Aug. 1965).

8. D. W. Fife and J. L. Smith, "Transmission Capacity of Disc Storage Systems with Concurrent Arm Positioning," *IEEE Trans. on Electronic Computers*, vol. EC-14, no. 4 (Aug. 1965).

AN OPTIMIZATION MODEL FOR TIME-SHARING*

Dennis W. Fife
Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

The proper design of scheduling processes for time-shared computers has provoked much discussion. One of the factors promoting discussion is the great variety of scheduling procedures which one can feasibly program, and which will operate with adequate efficiency. There is wide latitude for conjecturing possibly improved procedures. Moreover, the performance requirements of time-sharing demand more sophisticated schemes than one can analyze with simple queueing theory, yet it is not feasible to experiment with many alternative procedures in operational systems and produce quantitative evidence of their relative merits.

The scheduling techniques in use today are of two major types: round-robin procedures and multiple priority level procedures. A round-robin process[1,2] treats the queue of users uniformly, giving each program a "slice" of execution time and then swapping it for another. This conforms to one intuitive notion of time-sharing, inasmuch as each user may obtain an equal share of computer time on a short-term basis. Multiple priority schemes[3] allow the choice of a job for execution to be determined by its initial priority and the amount of execution time it has received. Compared to the round-robin, the priority procedure has greater flexibility inherent in the choice of initial priority assignments and the maximum execution time allocated to jobs of each priority level. By a proper simplification, a multiple priority scheme becomes a round-robin procedure.

The pioneers of the time-sharing field have experimentally found successful versions of the above procedures for their particular systems. But the future development of on-line computer systems will undoubtedly benefit from attempts to establish general quantitative properties of scheduling schemes, including ones which are not necessarily in current use. A good approach to take for this objective is to model the queueing situation resulting from a proposed scheduling procedure. Even more desirable however is an optimization technique which will allow one to model a class of scheduling procedures, and which will systematically synthesize an optimum procedure according to some specified criterion. The importance of Markov stochastic models in queueing theory suggests that the theory of Markov sequential decision processes[4-6] will provide the desired optimization model.

This paper describes some results produced by this type of model for a time-shared computer with four remote consoles and three queue levels for user jobs. Swapping and program loading time are included, and a rather general execution time distribution is treated. The optimal system has been computed for two distinctly different performance

*This paper is based upon research performed at the Systems Engineering Laboratory of the University of Michigan and submitted as a Ph.D dissertation in electrical engineering. The author is grateful for financial support from the U.S. Air Force Rome Air Development Center and the National Science Foundation. Assistance in publication was also provided by Project MAC at MIT.

measures, both related to response time. Although, a wide class of procedures is admissible, the optimal systems have much the same structure as those used in contemporary systems.

THE SYSTEM

The model concerns a hypothetical time-sharing system typical of those currently in operation, illustrated in Fig. 1. The mass memory is a large capacity magnetic drum, and we assume its revolution time to be 50 msec. The main core memory contains the Executive Control Program (ECP) for the system and a memory area allocated to the user program being executed. No multiprogramming is involved, so only one user program resides in core at any time. The user area is also taken to be of modest size, say 8000 words, allowing a user core image to be stored in one drum field. Thus swapping of two user programs can be accomplished in 100 msec minimum. In order to also accomodate ECP scheduling overhead we extend the swapping time to 150 msec.

The real time clock provides an interrupt every 50 msec, equal to the drum revolution time. In order to measure the passage of an arbitrary time interval, a timekeeping function is part of the ECP, and this activity occurs in response to every clock interrupt. A clock interrupt may also initiate the scheduling operation, as shown in Fig. 2. Because of this, the scheduling process involves discrete time steps, with a scheduling action possible occurring at any 50-msec step.

Two important conventions will be imposed on system operation. A user console may only have one command in process at any time. Also, every command must be initiated from a user console, thereby excluding one job from initiating another. As a result, the system may have at most four user jobs in process. The completion of any command marks the beginning of a user reaction time, after which another command arrives from the console.

One can conceive at this point of a very general scheduling procedure in which each decision requires a dual choice:

1. Selection of a job from queue to be executed.
2. Selection of a maximum time interval for execution of the job before returning it to queue.

The decision at any time could be based upon a variety of data, such as the accomplished execution time of jobs in queue, the size of job programs, the originating consoles, and the time at the decision point. We will only investigate a "context-free" case, in which jobs in queue are distinguishable only on the basis of the execution time each has already received and their time of arrival. Moreover, in keeping with the simplicity of existing systems, the allowable execution time intervals in (2) above will be restricted to three values in such a

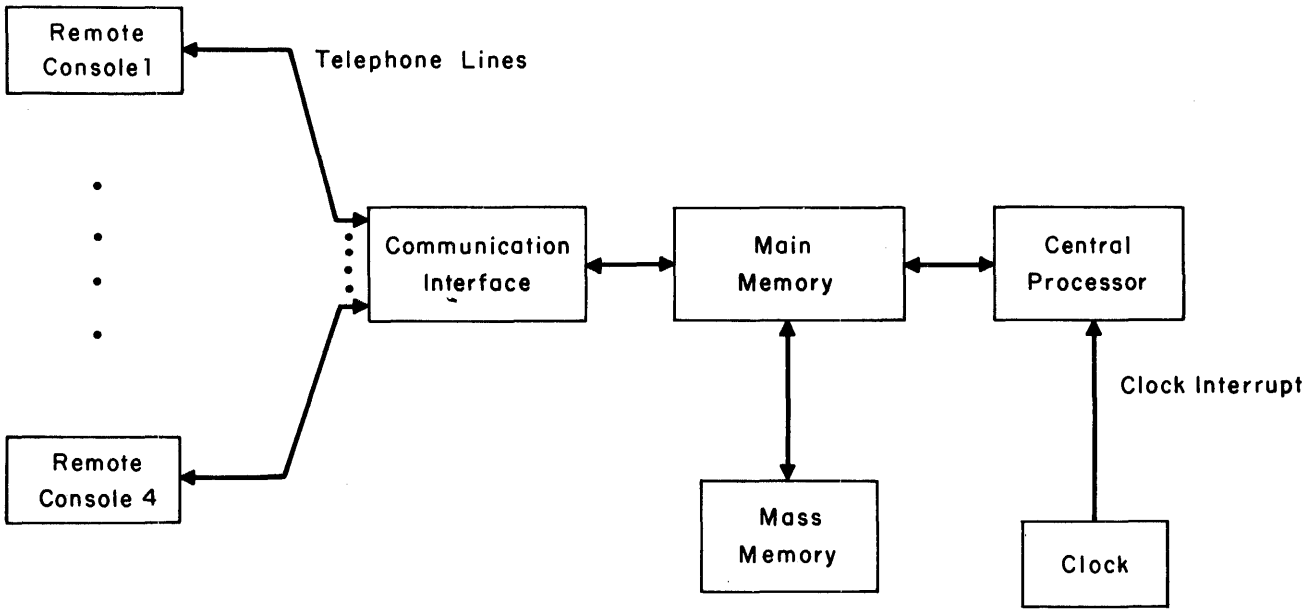


Figure 1. Equipment organization.

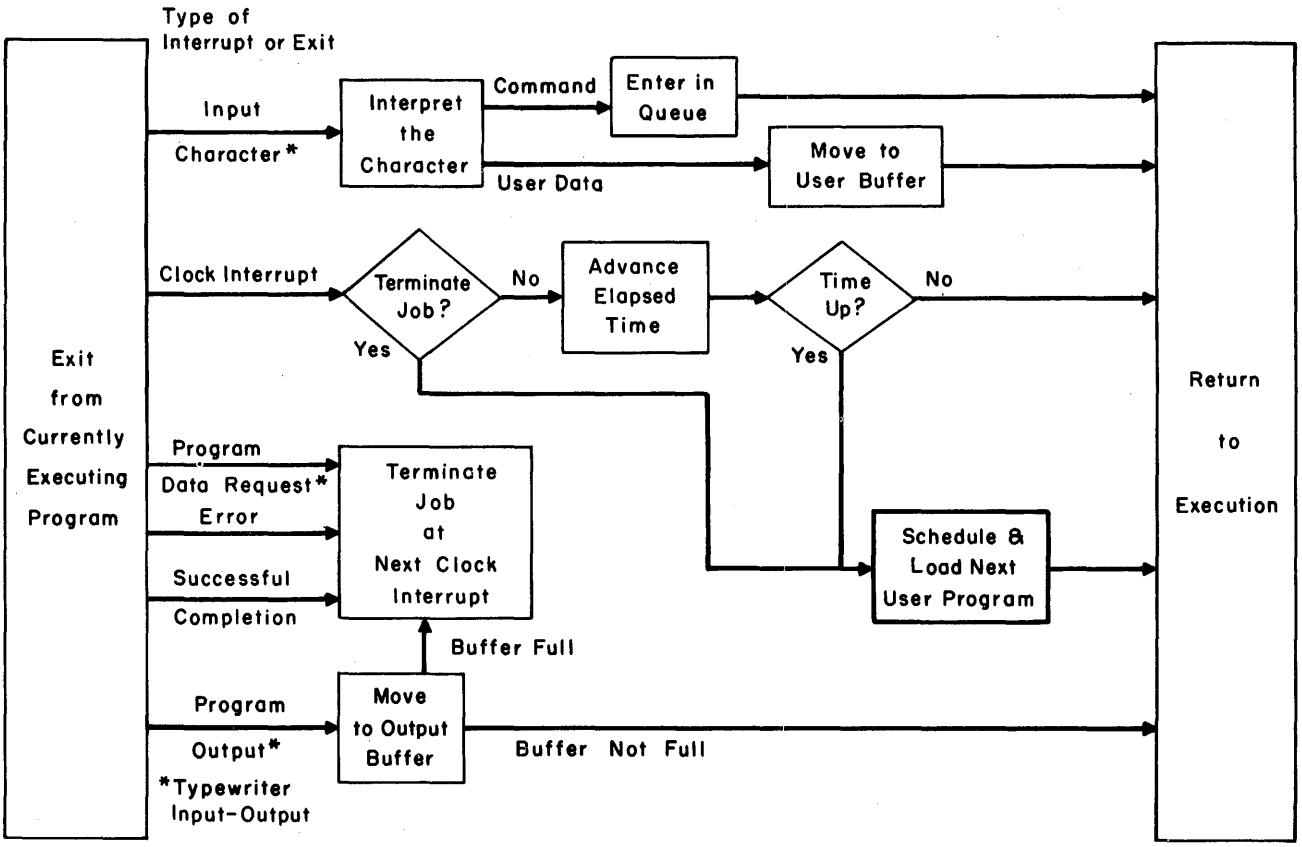


Figure 2. Executive control functions.

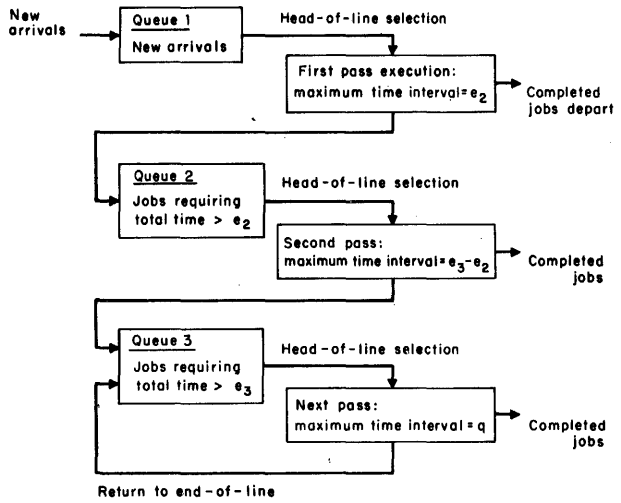


Figure 3. Multiple queue structure of the system.

way as to establish three queues as shown in Fig. 3. In this scheme only one job is executed at a time.

This structure is very similar to that in multiple priority procedures. But additional flexibility comes about because we may choose to serve a job from any queue that is occupied. There is no restriction on the choice, such as a priority ranking would impose. The choice can depend upon the number of jobs in each queue and will of course be influenced by the specified values for the maximum execution intervals, e_2, $e_3 - e_2$, and q. The model allows us to determine the optimum choice relative to a specified measure of performance, and to investigate optimal values of e_2, e_3, and q.

MEASURES OF PERFORMANCE

The quality of performance which users observe for a time-sharing system depends upon many factors. Reliability and ease of communication via suitable high-level languages are important examples of such factors. A queueing and scheduling study, however, emphasizes queueing delay as a performance factor. In doing so, one should recognize that the population of users generally consists of "interactive" and "background" users, where the distinction is loosely established by the average

amount of processor time per request. It seems clear that interactive users are usually more sensitive to delay, and constitute the more significant portion of the user population for a well-developed time-sharing system. Thus, without implying scorn for the performance requirements of background users, we will concentrate on the response time performance for interactive users.

Two quantitative measures of response time performance are of interest here. For the first we consider the system performance over any period of time to be measured by the sum of the intervals for all user jobs in that period. An improvement of response time performance can be achieved by seeking a scheduling procedure which minimizes this sum. Recognizing that job arrivals and completions are random, and taking an arbitrarily long period of time this optimization criterion becomes equivalent to minimizing the average number of user jobs in process at any instant of time.

The second optimization criterion arises from the realization that the minimum response time to any command is its execution time, and a user who requests a long computation must, by his own choice, be satisfied with a correspondingly long response time. Thus, for example, 0.5 sec additional delay on a job taking a minimum of 3 sec to do should not be as degrading to performance as the same additional delay on a 10-msec job. This reasoning suggests measuring system performance by a weighted sum of the response times for all jobs over some time period, where the weight applied to the response time of a job depends upon its execution time. Figure 4 depicts one weighting function which we have used to explore this case. Note that the first mentioned measure of performance corresponds to a unit weight for any value of execution time. The second optimization criterion therefore amounts to minimizing the average total weighted response time to user commands in any long time period of system operation. Table 1 summarizes the two criteria.

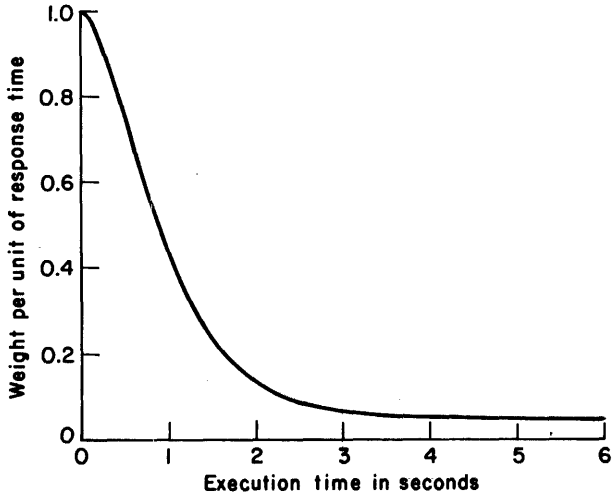


Figure 4. Weighting function for the second optimization criterion.

Table 1. Optimization Criteria

No.	Statement
1	Minimize average total response time for all user commands in a period of system operation.
2	Minimize average total response time for all user commands, each weighted according to Fig. 4, over a period of system operation.

MODELING

As one should anticipate, modeling of the time-shared system as a Markov sequential decision process depends upon certain idealizations. Among these is the assumption that the user reaction time, t_u, and the execution time per command, t_e, are independent random variables. These time intervals are the human time and the central processor time, respectively, in a man-machine interaction. The probability distributions treated are the following:

$$\text{Probability of } t_u \leq T = 1 - e^{-T/T_u} \qquad (1)$$

where T_u is the mean value of t_u, and

$$\text{Probability of } t_e \leq T = 1 - \gamma_1 e^{-\mu_1 T} - \gamma_2 e^{-\mu_2 T} \qquad (2)$$

where μ_1 and μ_2 are positive, and γ_1 and γ_2 are probabilities with unit sum. The mean execution time is $T_e = \gamma_1/\mu_1 + \gamma_2/\mu_2$. Equation (2) is called a hyperexponential distribution (Ref. 7, p. 19), and its use stems from some observations of execution time on the batch-processing operation of the Computing Center at the University of Michigan.[8] In Fig. 5 some points are given from these batch-processing observations, as well as two curves derived from Eq. (2) by different choices of the parameters. One sees that the observations could be fitted fairly well by a suitable member of the family represented by Eq. (2). We will concentrate on curves 1 and 2 of Fig. 5 as representative members of this family.

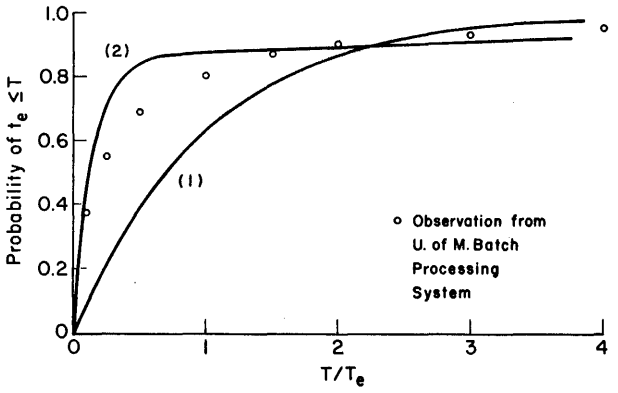


Figure 5. Execution time distributions.

Both distributions (1) and (2) above seem reasonable, but there is little data available to irrefutably justify them. Our batch-processing observations certainly provide some evidence supporting the general shape of the distribution for execution time. Also, there has been some statistical data from the MIT system[9] which indicates these idealizations are good approximations, as are the particular system parameter values we will use (see Table 2).

In addition we take constant values for both the swapping time, s, and the set-up time, S. The latter is the CPU time spent in initial relocation and linking of subroutines for a command. The ECP overhead occurring between successive scheduling decisions is neglected.

Based upon these assumptions and the physical properties of the system one can proceed to formulate a Markov model and derive the data needed for the optimization algorithm devised by Howard[5] and Jewell.[6] Such data includes probabilities on the command arrivals and the job completion during an execution pass, and the mean time durations between successive scheduling decisions. Also involved are quantities which measure the performance during an execution pass as determined by the optimization criterion of interest. A full discussion of the formulation of the model and how the necessary data are derived is beyond the scope of this paper. The interested reader should consult the references and the full report from which this paper is drawn.[10]

OPTIMAL SCHEDULING

The alternative choices of the queue to be served for each combination of numbers of jobs in the queues give rise to well over one billion different scheduling procedures for this system. Included in this number are the first-come, first-served procedure (FCFS), and the six priority procedures possible with this system. Among the latter, our results point especially to the "1-3-2 Priority," which assigns priorities 1, 3, and 2 to Queue 1, Queue 2, and Queue 3, respectively. (Priority 1 is top priority.) The optimization computation determines an optimal procedure from the admissible set for a given set of values of the system parameters listed in Table 2. The computation is fast enough, however, so that we have economically obtained solutions and general conclusions for the range of parameter values shown. These may hold for even wider variation of the parameters.

Table 2. System Parameters

Symbol	Definition	Appropriate Values
T_u	Mean user reaction time	20–30 sec
T_e	Mean execution time per interaction	1–4 sec
s	Swap time	150 msec
S	Setup time for relocation and subroutine linking	1 sec or less
e_2, $e_3 - e_2$, and q	Execution time allocations	Arbitrary, subject to modeling limitations

To begin the discussion, consider the first optimization criterion. The execution time distribution shown in Fig. 5 as curve 1, although not as appropriate as curve 2 perhaps, produces a rather interesting result. The optimal procedure is first-come, first-served for all parameter values. This makes sense intuitively, for FCFS avoids swapping time. Moreover, with this execution time distribution FCFS is always processing a job having minimum mean execution time to completion.

For the execution time distribution given by curve 2, one must consider both the optimal procedure and the optimal values for e_2, e_3, and q. The minimum value of e_3 permitted by the Markov model is 1.5 T_e, and any larger value gives poorer performance. A smaller value of e_3 is therefore likely to further improve performance. Subject to this limitation, optimization of the model indicates that the optimum procedure is 1-3-2 Priority and optimal values of e_2 and q are T_e and one clock interval (50 msec), respectively. This holds for

Table 3. Average Total Queue for Optimal Scheduling with Criterion 1

T_u	T_e	S	e_2	e_3	q	Optimal Policy	Aver. Queue
			seconds			*Priority:*	
30	4	0	0.6	6	0.05	1-3-2	0.8131
30	4	0	3.5	6	0.05	1-3-2	0.6691
30	4	0	4.0	6	0.05	1-3-2	0.6671
30	4	0	4.25	6	0.05	1-3-2	0.6690
30	4	0	4.0	6	0.50	1-3-2	0.6725
20	1	1	0.50	1.5	0.05	1-3-2	0.4893
20	1	1	1.0	1.5	0.05	1-3-2	0.4853
20	1	1	1.45	1.5	0.05	1-3-2	0.4862
20	1	5	1.0	1.5	0.05	FCFS	1.402

values of S not exceeding T_e. Table 3 gives the computed average total queue of jobs for typical cases. The minimum queue occurs for the optimal values of e_2 and q.

Now consider the second optimization criterion. The minimum value of e_3 permitted by the model is the larger of 1.5 T_e and 4 sec, the latter arising from a need to have constant weight for jobs taking longer execution than e_3 (see Fig. 4). Although this limitation has some effect, it is still surprising that the optimal system is much the same as for the first criterion. Moreover, the optimal system is the same for both execution time distributions (1) and (2) of Fig. 5, in contrast to the case of the first criterion. Table 4 shows typical values of the computed performance measure over a one-unit (50-msec) time interval.

Table 4. Average Unit Time Performance Measure for Optimal Scheduling with Criterion 2

T_u	T_e	S	e_2	e_3	q	Optimal Policy	Aver. Unit Time Performance
			seconds			*Priority:*	
30	4	0	2	6	0.05	1-3-2	0.0806
30	4	0	3	6	0.05	1-3-2	0.0782
30	4	0	4	6	0.05	1-3-2	0.0788
30	4	0	3	6	0.50	1-3-2	0.0821
30	1	1	0.50	4	0.05	1-2-3	0.1592
30	1	1	1.5	4	0.05	1-3-2	0.1591
30	1	1	2.0	4	0.05	1-3-2	0.1595
30	1	1	1.0	4	0.50	1-2-3	0.1599

Several important points emerge from the optimization of the model. The optimal system is substantially insensitive to the precise values of the system parameters. This makes it feasible to apply the results of the model to a physical system, where parameters are not known exactly and may change gradually. The fact that all optimal procedures are priority procedures is noteworthy both in regard to simplicity of implementation and the current practice in time-shared systems. The emphasis upon a minimum execution time allocation for the third queue indicates a need for very rapid preemption of low priority jobs. This is a contribution, for apparently no existing system allows preemption except after a significant delay.

This study leads us to suggest a scheduling procedure which should produce somewhat better performance than those considered here. The system would have two priority levels, with new jobs entering the first priority queue. A first priority job would preempt a lower priority job one clock interval after the former's arrival, and would then receive execution for a maximum time equal to T_e, the mean execution time of the population. After this it would be relegated to second priority. The lower queue would be served round-robin with an execution quantum much larger than T_e.

COMPARISON OF PROCEDURES

A much better picture of the performance of the optimal system can be obtained from the mean response time of a command, given the execution time required. Table 5 describes the optimal system and three other scheduling policies used in existing systems. The round-robin procedure is essentially a 2-1-3 Priority system, which places new arrivals at the head of the round-robin queue. Figures 6 and 7

Table 5. Scheduling Policies for Comparison

Policy	Type	e_2	e_3	q
Fig. 6 Cases:	*Priority:*		*seconds*	
Round-Robin	2-1-3	—	6.0	6.0
Priority A	1-2-3	2.0	6.0	8.0
Priority B	1-2-3	2.0	6.0	2.0
Optimum	1-3-2	4.0	6.0	0.05
Fig. 7 Cases:				
Round-Robin	2-1-3	—	1.5	1.5
Priority A	1-2-3	0.5	1.5	2.0
Priority B	1-2-3	0.5	1.5	0.5
Optimum	1-3-2	1.0	1.5	0.05

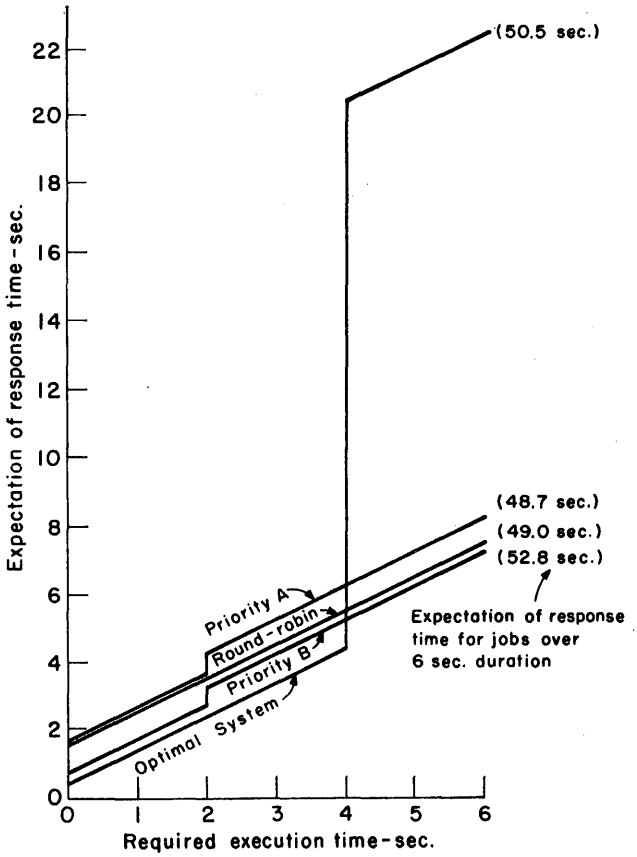


Figure 6. Response time vs execution time for distribution (2), T_u = 20 sec, T_e = 4 sec, S = 0.

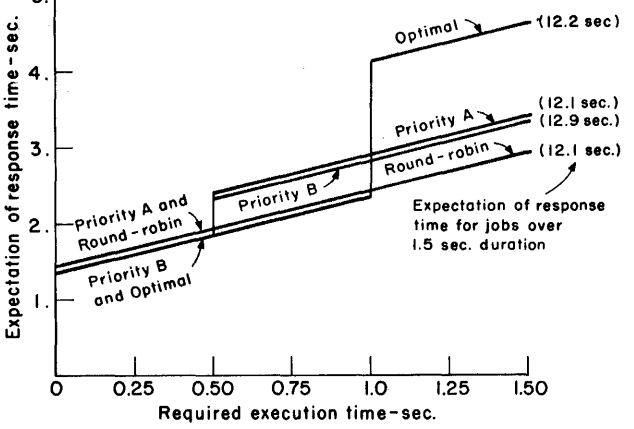


Figure 7. Response time vs execution time for distribution (2), T_u = 20 sec, T_e = 1 sec, S = 1 sec.

compare the mean response time for these policies. One can see that the optimal system achieves a smaller response time for trivial computations at a cost of larger response time for jobs requiring execution time T_e or greater. The other policies experience a large jump in response time at an execution time of 1.5 T_e, so this aspect of the optimal system should not be disturbing.

CONCLUSIONS

The optimization approach we have used is quite new for systems with queues, and the practical results one can obtain commend it for future studies.

The time-shared computer system we have modeled is quite typical of existing systems. Three noteworthy conclusions relative to time-sharing practice have been found. To begin, it makes no substantial difference under the performance measure we have treated whether or not one gives extra importance to the response time of trivial computations. The optimal system, subject to the limitations of the model, is a multiple priority scheme and is reasonably insensitive to the values of the system parameters. There is a need for more rapid preemption of lower priority jobs than presently used in operational systems.

Our results suggest a two-priority scheme where new arrivals are given first priority and are relegated to second priority after receiving the mean execution time. A new arrival should preempt a lower priority job as soon as possible. There is a need to study multiple priority schemes with more levels, and with greater flexibility in execution time allocations than we have treated.

REFERENCES

1. S. Boilen et al, "A Time-Sharing Debugging System for a Small Computer," *Proc. SJCC,* 1963, pp. 51–57.

2. J. Schwartz et al, "A General Purpose Time-Sharing System," ibid, 1964, pp. 397–411.

3. F. Corbató et al, "An Experimental Time-Sharing System," ibid, 1962, pp. 335–334.

4. R. Howard, *Dynamic Programming and Markov Chains,* MIT Press, Cambridge, Mass., 1960.

5. ——, "Semi-Markovian Control System," Tech. Report No. 3, Contract Nonr-1841 (87), Operations Research Center, MIT, Cambridge, Mass.

6. W. S. Jewell, "Markov-Renewal Program-

ming," *Operations Research,* vol. 11, 938–971 (1963).

7. P. M. Morse, *Queues, Inventories and Maintenance,* Wiley and Sons, New York, 1958.

8. E. Walter and V. Wallace, "Further Analysis of a Computing Center Environment," Systems Engineering Laboratory Tech. Report, University of Michigan, Ann Arbor, to be published.

9. A. L. Scherr, "An Analysis of Time-Shared Computer Systems," Project MAC.-TR-18, MIT, Cambridge, Mass (June 1965).

10. D. W. Fife, "The Optimal Control of Queues, with Application to Computer Systems," Cooley Electronics Laboratory Tech. Report No. 170, University of Michigan, Ann Arbor (Nov. 1965).

A DIGITAL SYSTEM FOR ON-LINE STUDIES OF DYNAMICAL SYSTEMS

T. C. Bartee
Harvard University,
Cambridge, Massachusetts
and
J. B. Lewis
Lincoln Laboratory, Massachusetts Institute of Technology*
Lexington, Massachusetts

INTRODUCTION

The study of dynamical systems with the aid of analog and digital computers has developed rapidly in the past two decades. Increased interest in systems described by differential equations which are nonlinear or have time-varying coefficients, has resulted in more reliance on techniques requiring on-line computation. Usually less is known a priori of how the solutions will develop or what parameter values or initial conditions should be used. The recent trend to using hybrid computers (combinations of digital and analog equipment) has been motivated by the desire to study complex dynamical systems with computer configurations which are designed with particular classes of problems in mind. Extensive use of display, plotting, and printing equipment as well as elaborate consoles attest the on-line capability of such computers.

Designing a computer for the on-line study of dynamical systems involves many factors, but among others, speed, accuracy, cost, and user convenience are particularly important. The system described here has emphasized user convenience so that experimental or "trial and error" computational methods are encouraged. An accuracy of 0.1% to 0.01% was considered adequate, and the speed (as measured by ability to solve problems in real time) is roughly equal to the fastest commercial digital computers. Although many modern analog computers are considerably faster in solving systems of ordinary differential equations, and much greater accuracy can be obtained on digital computers, this compromise still allows the study of many problems of great interest. The system is a laboratory experimental model rather than a production prototype, and the cost was kept low by using many components which were on hand. The added cost to the large time-shared system, of which it is a part, was relatively small.

The basic computer configuration is similar to many hybrid computers in that it includes a general-purpose digital computer and a special-purpose computer which is largely a collection of integrators. Most frequently, the special-purpose computer in a hybrid is a high-speed analog computer that meets the need for real-time simulation. In the system discussed here, the special-purpose computer is a high-speed digital differential analyzer (DDA)—a collection of digital summers which approximate

*Operated with support from the U.S. Air Force.

integration. (See Ref. 7 also.) One great convenience to the user results from the fact that the interconnection of the integrators is specified as part of the 86-bit words that describe the integrators. Thus, a patch board is not needed, and as a result, it is possible to write programs for a general-purpose computer to set up the DDA. It is the combination of hardware, which allows rapid, program-controlled changes in interconnection, and software, which translates a problem statement into interconnection information, that makes this system quite attractive. High-speed operation is important in reducing reaction time, and flexible controls that allow start/stop, display, sampling, and repetition are other notable features.

The interconnection of the small general-purpose digital computer, the LINC, and the DDA is described in the next section. The setup of the DDA is done by transferring information from the LINC core memory to the DDA core memory. The information in the LINC core memory is obtained by transfer from a large time-shared computer in which a mapping and scaling program operates. At present, interconnection to the Project MAC computer over a teletype line has been made, and connection to the Lincoln Laboratory IBM 360 System will be completed soon. The mapping and scaling can be done manually for simple problems and the results inserted directly into the LINC core memory. The operation of the system is described in a later section; an important feature is the special combination of LINC and DDA which the user may operate in an experimental fashion with less concern for the usual high charges for time on a large central processor. The large processor is used only when it is necessary to map or scale.

A HIGH-SPEED DIGITAL DIFFERENTIAL ANALYZER

Basic DDA Algorithms and Other Features

The design of early DDA's was centered about the use of magnetic drums, these being memory devices of reasonable cost with a serial operation which was especially attractive in the processing of a set of DDA integrators. The development of core memories of modest price with cycle times in the 1-microsecond region provides the designer with the possibility of making a DDA with a much higher speed and at quite reasonable costs.

A DDA in which a core memory is used requires a structure different from the usual one. In addition, when one desires to connect a general-purpose machine in a reasonable way, and in particular, to organize the DDA so that the general-purpose machine easily can load (or change) the interconnections of the integrators, the starting and intermediate values in the integrators, the scale factors, and so forth, a different organization becomes attractive. Each word in the memory stores the information concerning a single integrator, and all operations for updating an integrator are performed using parallel arithmetic.

Before proceeding with the organization of the machine, let us briefly examine the integrator algorithm and number system selected. These date back to MADDIDA, and (to the best of our knowledge) were the work of I. S. Reed. Each integrator in the system consists of a single word in the core memory. Variables in the system are represented by 24 binary bits including sign. The number system used is a 2's complement system with sign bit complemented. For 4-bit numbers, the number representation is as shown below:

1111 = +7	0111 = −1
1110 = +6	0110 = −2
1101 = +5	0101 = −3
1100 = +4	0100 = −4
1011 = +3	0011 = −5
1010 = +2	0010 = −6
1001 = +1	0001 = −7
1000 = 0	0000 = −8

Each integrator realizes the relation

$$dz = Cy\,dx \tag{1}$$

where C is a constant, dz an "output increment," and dx an "input increment," and y the integrand.

The definite integral of the above relation is

$$z(x) = z(x_0) + C\int_{x_0}^{x} y(\gamma)\,dx(\gamma) \tag{2}$$

This definite integral is approximated by a sum. The interval (γ_0, γ) is divided into n subintervals of length $\Delta\gamma$ so that $\gamma_k = \gamma_0 + k\Delta\gamma$, and we choose $x_0 = x(\gamma_0)$ and $x = x(\gamma)$. Therefore,

$$z(x) = z(x_0) + C\sum_{k=1}^{n} y(\gamma_k)\,\Delta x(\gamma_k) + \epsilon_n$$

where

$$\Delta x(\gamma_k) = x(\gamma_k) - x(\gamma_{k-1})$$

and

$$y(\gamma_{k-1}) + \Delta y(\gamma_k) = y(\gamma_k)$$

and ϵ_n is an approximation error. Finally, letting $y(\gamma_k)\,\Delta x(\gamma_k) = \Delta z(\gamma_k)$, the result is

$$z(x) = z(x_0) + C\sum_{k=1}^{n} \Delta z(\gamma_k) + \bar{\epsilon}_n \qquad (3)$$

where $\bar{\epsilon}_n$ is again an error term.

This mathematical representation shows that the DDA performs incremental computations rather than full-word operations, and the approximation to integration is usually called rectangular integration. As shown below, the computations are fixed-point, so that scaling is very important. Finally, it should be noted that the DDA solves initial-value problems.

Let us now consider a part of an integrator register. An integrator register can be looked upon as a "black box" with two inputs, *dx* and *dy*, an output *dz*, and a stored value *y*, such that relation (1) above is approximated.

In this machine there are 256 integrators which are sequentially updated. Each integrator is in reality a word in the core memory consisting of 24-bit *R*-register and a 24-bit *Y*-register, plus information as to which outputs from the other integrators comprise the *dy* and *dx* inputs to the integrator. There is also other information in each word, and this is discussed in the following section. Figure 1 is a block diagram of an integrator.

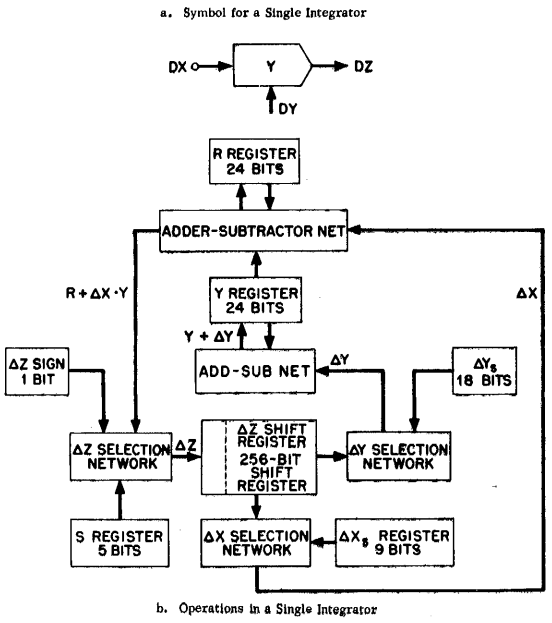


Figure 1. Block diagram of DDA integrator.

The *Y*-register part of each word contains values of the variable *y*. Provision is made so that the *Y*-register can be added to or subtracted from the *R*-register, and this is controlled by the *dx* input, which is a one-bit input that we call ΔX.

The overflow or carry from the most significant bit of *R* (each time *Y* is added to *R*) is called ΔZ, and it has value +1 if an overflow of *R* occurs and −1 if no overflow occurs.

The *dy* input is actually a sum of the outputs (or ΔZ's) from up to 16 other integrators. We call this sum ΔY and the particular ΔZ's which are added together to form ΔY are selected in a manner which is described later.

The ΔZ's from each integrator are stored in a circular shift register of 256 flip-flops. Each position in the shift register corresponds to an address in the memory. The integrators are processed or updated starting with the integrator at address 1 in the memory proceeding through the integrators until all those that are being used have been updated; then the integrator at location 1 is again updated followed by the others. The core memory is a split-cycle memory requiring 0.75μsec for each half-cycle so that it takes 1.5μsec to update a single integrator. A problem with 16 integrators would then require 24μsec for a single iteration or updating of all integrators.

During each updating of an integrator the following operations occur (see Figs. 1 and 2):

1. The ΔZ's in the shift register, the sum of which comprise the ΔY for the inte-

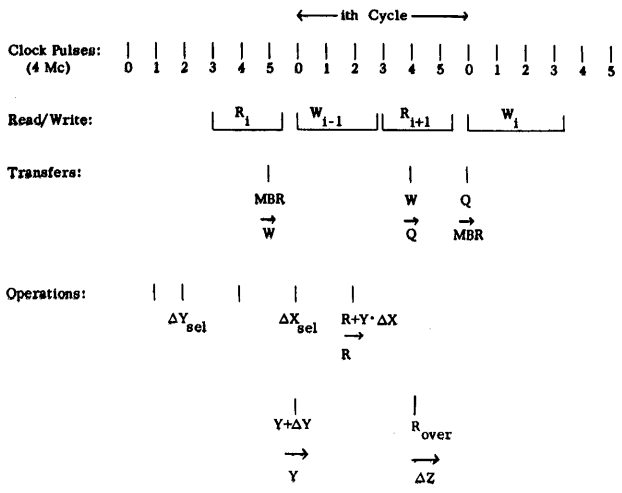


Figure 2. Timing for processing the *i*th integrator. The ΔY_s bits for the *i*th integrator are stored in the $(i-1)$th word.

grator being updated, are selected and added together and stored in five flip-flops.

2. The ΔX input is selected from the ΔZ shift register and stored in a flip-flop.
3. The value of ΔY is added to the current value of Y.
4. The new value of Y is multiplied by ΔX and this is added to R. The value of the overflow from R is then stored in the appropriate location in the ΔZ shift register.

Proof that this algorithm will yield a system in which the integrators approximate the relation given earlier may be found in Ref. 1. Other DDA's are described in Refs. 2, 3, 4, and 8.

A servo mode of operation for the integrators also is included, in which, instead of using the overflow from R as the value to be placed in the ΔZ shift register, the sign of y is placed in ΔZ each time. This makes it possible to obtain an increment which approximates a sum of increments. A bit in each integrator word tells the control whether servo mode or conventional mode is to be used with a particular integrator.

It is also possible to invert or complement the ΔZ output from a given integrator. A bit in each integrator word tells whether or not the overflow from R (or sign of Y in the servo mode) is to be complemented before being stored in the ΔZ shift register.

Description of Machine Organization

The basic difference between this DDA and others which have been constructed to date is that each integrator carries with it information as to which of the other integrators comprise the ΔY and ΔX inputs. Also, complete information is provided as to whether it is to be used as an integrator or in the servo mode, whether or not it should be sampled, and so forth. As a result, each word in the core memory contains 86 bits.

Figure 3 shows a simplified block diagram of the DDA. Each time an integrator is to be updated, it is selected and read into the memory buffer register, which is a part of the memory. The integrator word is then transferred into the W register. The parts of each 86-bit word are as follows:

1. R is the remainder, or running sum, as explained previously; 24 bits are used for R.
2. Y contains the current value of the variable y as explained in the previous section; 24 bits are used for Y.
3. ΔX_s contains 9 binary bits, 8 of these are used to select the value of ΔX from the 256-bit ΔZ register, or an independent increment, ΔT, is selected by the 9th bit.
4. ΔY_s contains 18 bits. This part of the register is used to select the ΔY-inputs to be added to Y. The selection can be made in three ways, and 2 bits of the register tell which way ΔY_s is to be used. The possibilities are:

   a) ΔY_s may be used as a linear mask; in which case, the 16 selection bits of ΔY_s are simply placed over the ΔZ register with the 8th bit on the ΔZ from the current integrator. If a given bit in ΔY_s is a 1, the corresponding ΔZ is selected to be added into ΔY; if the bit is a 0, the corresponding ΔZ is not added into ΔY.
   b) The first 8 of the selection bits of ΔY_s can be used to select a single ΔZ to be added to Y. Any one of the 256 ΔZ's can be selected in this manner.
   c) The first 8 selection bits of ΔY_s can be used to select a ΔZ and the second 8 bits to select another ΔZ, so that ΔY will be the sum of these two selected ΔZ's.

5. D consists of two bits and tells the display equipment whether or not the current value of Y is

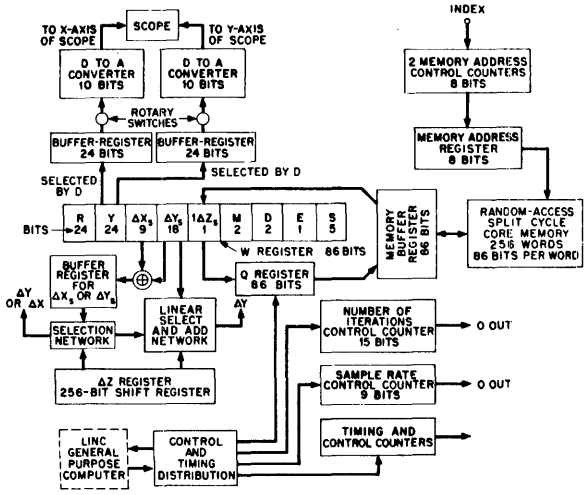


Figure 3. Block diagram of DDA.

to be displayed on the oscilloscope; and if it is to be displayed, whether it should be used to deflect along the X or Y axis of the scope.

6. E is a 1-bit register that tells whether or not the Y-value of the integrator should be given to the LINC computer for examination. The frequency at which the integrators are to be examined by the LINC computer is loaded into the DDA as the number of iterations to be performed between each examination. Only those registers which have 1's in their E-position will have their Y-values transferred to the LINC computer. At this time, the system also can change values in the Y-registers. This enables the user to introduce step functions or other functions and also to check on current programs, record data, or display it when required.

7. S contains 5 bits and is used to give the scale factor that determines the length of the Y and R registers for the integrator. S determines which carry or overflow from the added stages is examined to form ΔZ for a given integrator.

8. M consists of two mode bits. One bit indicates whether the integrator is to operate in normal or servo mode. The second indicates that the programmer wishes to stop the DDA if $C(Y) = 0$.

9. ΔZ_{sign} is a bit which indicates whether the ΔZ output is to be complemented.

The use of ΔY_s as a linear selector permits up to 16 ΔZ inputs to a single integrator i, but these must be within the interval $(i - 7, i + 8)$. A study of the problem of interconnecting integrators with this capability revealed that while most problems could be mapped by adding "dummy" integrators, it made the actual number of integrators in use in some cases unreasonably large. The addition of an optional selection mode allowing either one or two ΔZ's from among any of the 256 integrators alleviated this problem and greatly increased the apparent capacity of the DDA.

In setting up a problem, the LINC computer must give the DDA certain information in addition to that in the integrator. For instance, the number of integrators to be used is first entered into DDA circuitry. The machine is constructed so that the number of integrators processed is determined by this number. Integrators are processed in sequence, and one can use 16, 32, 48, ..., or 256 integrators in a problem. Thus for small problems only 16 integrators are processed each cycle of the machine, requiring only 24μsec. However, for a 100-integrator problem, 112 integrators would be processed each cycle and a single updating would require 168 μsec. Any unused integrators in these groups are processed, but no harm results except a slight loss in speed of operation. The LINC also loads a counter with the number of iterations or passes through the integrators which are to be made. The DDA will then stop after having performed the required number of iterations. Finally, the LINC loads the number of iterations between samples into the DDA. The sampling rate is determined by this. If the LINC loads the number N into the DDA, after each N iterations the DDA will pause on each integrator having a 1 in the E bit and transfer the contents (Y-register) of this integrator to the LINC. The LINC program examines each of the selected integrators in turn, and the value in each of the integrators also may be changed.

In order to use the memory efficiently, a split-cycle memory is used, and the integrators are not processed in a direct line. When the word associated with an integrator is called, and the memory delivers the word, there is not enough time to process the integrator and write the results back into the memory without a delay. We therefore process integrator i while writing the word for $i - 1$ into memory and reading $i + 1$ from memory. This is possible since an extra buffer, the Q-register, is included in addition to the W-register and the memory buffer register (MBR). Three transfers are involved: MBR to W, W to Q, and Q to MBR. Figure 2 illustrates the timing of the combined processing of i, the writing of $i - 1$, and the reading of $i + 1$ during a 1.5-μsec cycle.

A cause for interruption of the processing of the integrators is the overflow of a Y-register. The contents of a given Y-register can exceed the capacity of the (scaled) register in either a positive or a negative direction, and each Y-register is checked before and after the ΔY is added to see if overflow has occurred. If this happens, the DDA transfers this information to the LINC.

Finally, a stop on a zero crossing, (i.e., $C(Y) = 0$) can be programmed so that some decision capability is included. This allows the user to stop the DDA when certain dependent variables reach selected values.

METHOD OF USING THE SYSTEM

The basic input to the system that the user must supply is very much like the usual format used in the study of ordinary differential equations. The user must have his equation in first-order normal

form. After calling the mapping program, he first enters his equations. For example, if the original equation is

$$\ddot{q} - q\dot{q} - \sin q = 0 \tag{4}$$

let

$$\begin{aligned} q_1 &= \sin q \quad \text{and} \quad & dq_1 &= \cos q\, dq \\ q_2 &= \cos q & dq_2 &= -\sin q\, dq \\ q_3 &= q & dq_3 &= dq = \dot{q}dt \\ q_4 &= \dot{q} & dq_4 &= d\dot{q} = \ddot{q}dt \end{aligned} \tag{5}$$

and the required form is then

$$\begin{aligned} dq_1 &= q_2 dq_3 \\ dq_2 &= -q_1 dq_3 \\ dq_3 &= q_4 dt \\ dq_4 &= q_3 q_4 dt + q_1 dt \end{aligned} \tag{6}$$

This information is types as

$$\begin{aligned} \text{DQ1} &= \text{Q2*DQ3} \\ \text{DQ2} &= -\text{Q1*DQ3} \\ \text{DQ3} &= \text{Q4*DT} \\ \text{DQ4} &= \text{Q3*Q4*DT} + \text{Q1*DT} \end{aligned} \tag{7}$$

and the mapping program then generates a map. Although a number of arbitrary rules are used in the mapping, test problems and recent experience indicate that the program generates maps which are almost as efficient as those done manually in very simple problems. In complex cases, maps that would be quite time-consuming if done manually are generated in several seconds.[5]

The interconnection table that is generated by the mapping program is next used as an input to the scaling program. The user also must supply an estimate of the maximum magnitude of each variable, i.e., $|q_i|\max$, $i = 1, 2, \ldots$ This information is then used to compute an optimal set of scale factors using a linear programming routine which maximizes the sum of the number of bits used in all the Y-registers.[6] Finally, ϵ_0, a variable in the scaling program that relates problem time to machine time, must be specified within limits set by the scaling program.

The mapping and scaling are illustrated in Fig. 4 and Tables 1 and 2. Since the user generally does not need this information, it is saved in the large computer system and supplied only on special request. Figure 4 is a map of Eqs. (6), and Table 1 is the corresponding interconnection table. Table 2 is a table of scale factors that illustrates the interaction of scaling variables; the time scale and maximum values were chosen arbitrarily.

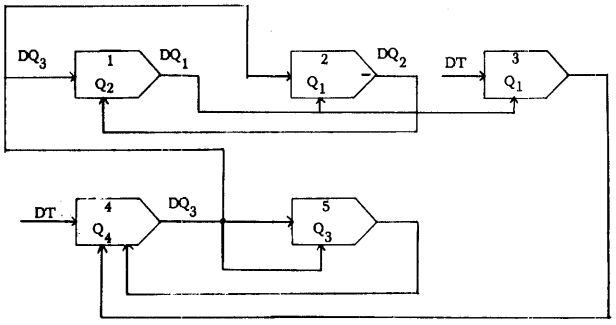


Figure 4. Map.

Upon completion of mapping and scaling, the initial contents of R (set to zero), ΔX_s, ΔY_s, M, ΔZ_{sign}, and S have been specified. The remaining inputs required of the user are the initial values $q_i(0)$, the number of iterations to be run, the sampling period, and an indication of what variables are to be displayed and sampled. The transfer of this binary information to the LINC is initiated, and after completion, the run on the DDA is made.

Table 1. Interconnection Table

Int. No.	DX-Input	DY-Inputs
1	4	2
2	4	1
3	0	1
4	0	3,5
5	4	4

NOTE: 0 indicates a DT input.

Table 2. Scaling Table

Int. No.	η	γ	α	ϵ	δ	ν
1	1*	−12	1	−11	−11	12
2	1*	−12	1	−11	−11	12
3	1*	−15†	6	−9‡	−11	17
4	3*	−15†	3	−12	−9	12
5	3*	−12	3	−9‡	−12	15

*Obtained from user estimates as in b) below.
†Determine DT scale relative to problem time.
‡These must be equal.

NOTES:

a) $\gamma_i, \delta_i, \epsilon_i$ are scales on dx_i, dy_i, dz_i, e.g., $dX_i = 2^{\gamma_i} dx_i$.
b) $\alpha_i \geq \eta_i$ where $2|y_i|_{\max} \geq 2^{\eta_i} > |y_i|_{\max}$, and q_i is y_i in the example.
c) $\nu_i = \alpha_i - \delta_i$ and $23 \geq \nu_i \geq 4$.
d) $\alpha_i + \gamma_i = \epsilon_i$.
e) $S_i = \nu_i + 1$, i.e., the S-bits of Integrator i.

Repeated runs may be made and new variables may be sampled or displayed by making changes at the LINC keyboard. If parameters in the differential equations are changed, rescaling may be required, and the scaling program is then called again on the time-shared computer. The same is necessary if overflow occurs because of wrong estimates on the $|q_i|$ max.

It is of interest to estimate some typical operating times. After input typing is completed, the reaction times are between several minutes and several seconds depending on the accessibility of the time-shared system programs. The loading and running time on the DDA is at most about 20 seconds for 2^{15} iterations of 256 integrators. The time for repeated runs depends largely on the time required for the user to enter new information at the LINC keyboard.

SUMMARY

The design of a system for on-line studies of dynamical systems has been described, and the details of operation of the high-speed DDA have been given. Two applications of the system which are showing its usefulness are

1. Spectral analyses of radar data in which a number of frequencies, spectral windows, smoothing times, and range gates were examined. On-line techniques enable the user to search for combinations of interest rather quickly. The LINC/-DDA combination can generate 30 spectral lines, compute a periodogram, and display the results in about 20 seconds.
2. Trajectory generation in simulation studies where the effect of changing parameters in an estimation algorithm are of interest. A set of 3-degree-of-freedom equations that includes atmospheric drag variations and gravitational variations requires 55 integrators. One run (i.e., one trajectory) requires 5 seconds in this case.

A subject requiring further study on the system is error analysis. Error prediction for equations that are integrated by incremental arithmetic operations is extremely difficult, but it is believed that more experience will provide some insight. Very little theoretical work has been done, and the combination of nonlinear equations, incremental methods, and quantization makes the prospect of estimating useful error bounds discouraging. It also may be noted that the sequential processing of the integrators introduces an ordering problem in the mapping program, and the effect of a given ordering scheme on computational errors is again difficult to predict.

Nevertheless, the system is proving very useful in problems where on-line searching and experimentation lead to more complete understanding of certain physical problems. The combination of general and special-purpose digital computers is of great value for studying complex dynamical systems.

ACKNOWLEDGMENTS

The authors wish to thank Dr. H. K. Knudsen for his valuable work on the mapping and scaling programs, and Mr. R. A. Carroll for his work in the planning and construction of the DDA.

REFERENCES

1. T. C. Bartee, I. Lebow and I. Reed, *Theory and Design of Digital Machines*, McGraw-Hill, 1962, pp. 252–269.
2. G. F. Forbes, *Digital Differential Analyzers*, G. F. Forbes, Sylman, Calif., 4th ed., 1957.
3. M. Palevsky, "The Design of the Bendix Digital Differential Analyzer," *Proc. IRE,* vol. 41, no. 10, pp. 1352–1356 (Oct. 1953).
4. J. M. Mitchell et al, "The TRICE—A High Speed Incremental Computer," *IRE Nat. Conv. Record*, Part 4, pp. 206–216 (1958).
5. H. K. Knudsen, "A Program for Automatic Mapping of Differential Equations on a DDA" (to appear).
6. H. K. Knudsen, "The Scaling of Digital Differential Analyzers," *IEEE Transactions G-EC*, EC-14, no. 4, pp. 583–589 (Aug. 1965).
7. O. A. Reichardt, M. W. Hoyt, W. T. Lee, "The Parallel Digital Differential Analyzer and Its Application as Hybrid Computing System Element," Simulation 4, n.z., pp. 104–113 (Feb. 1965).
8. M. W. Goldman "Design of a High Speed DDA," *Proc. FJCC*, pp. 929–949 (1965).

SIMULATION OF LOGICAL DECISION NETWORKS OF TIME-DELAY ELEMENTS BY MEANS OF A GENERAL-PURPOSE DIGITAL COMPUTER

Y. N. Chang and O. M. George
North American Aviation, Space and Information Systems Division
Downey, California

INTRODUCTION

For the purpose of this paper, a logical decision network is defined as a system whose elements can be in either of only two states, TRUE or FALSE. Therefore the operation of the system can be described by a set of simultaneous Boolean equations which are functions of time. These states must be defined specifically for each of the elements of the system, and in particular might represent yes and no, on and off, energize and de-energize a coil, "make" and "break" of a relay contact, presence and absence of voltage at a node, open and close a mechanical valve, etc.

Digital switching networks, relay control systems, production and manpower scheduling, and the transportation problem are examples of logic systems. In logic systems like the last two examples, the digital program is used to solve the problem. In the first two examples, where hardware is involved, simulation on a general-purpose digital computer is used to verify the time-sequential operation of the system before money and time have been spent in building it. A simulation should also permit one to vary the time delays in order to set tolerances for design specifications. A third application would be a study of the effects on the system of malfunctions of any elements. Finally, if one has simulated the checkout of various subsystems, it should be possible to simulate a combined system without recoding the Boolean equations. All of this can be done with the IBM 7094 program described herein. In addition, the time-sequential states of all the elements are plotted on the S-C 4020 in the form of an *n*-channel recorder chart.

The art of simulation involves building a mathematical model which represents the system. Boolean algebra is the natural mathematics for a logic system; however, Boolean algebra does not incorporate time per se. The authors propose and illustrate an extension of standard notation so that time relationships between cause and effect can be expressed explicitly in the Boolean equations that are written to describe the logical operation of a system.

MECHANIZATION OF A LOGICAL SYSTEM

Input and Dependent Elements

Any physical device (e.g., relay coil, switch, light, squib, or hydraulic valve) which has only two possible states is defined as a "logical element." A logical element can also be abstract (e.g., the set or reset action of a motor switch, the grounding of a buss, the short circuiting of a battery, or the presence of voltage at a node). Letter T is used to represent the "true" state (i.e., on, closed, energized, shorted, etc.) of an element, and the letter F represents the opposite "false" state. A device which has more than two states must be resolved into *n* logical elements. For example, an *n*-position rotary-type stepping switch must be programmed as *n* logical

elements where each element is in state T only while the switch makes contact at its position. In order to simulate the operation of a system by means of the digital computer program described in this paper, each logical element must be identified by a name of six or fewer legitimate alphameric characters.

Input elements (e.g., hand-set switches, or radio-command signals) are defined as those logical elements whose states are independent of the states of other elements. The states of input elements are assigned from time to time during the simulation by the entry of data.

A dependent element is defined as a logical element whose state is determined by solving a Boolean equation which expresses the state as a function of time-delay variables and/or the states of input elements.

Time-Delay Variables

A time-delay variable is always related to a logical element and, therefore, is identified by the same name as its associated control element. For instance, the coil of a relay (say dependent element DEPEND) may control the operation of two time-delay contacts. The states of these contacts would then be time-delay variables in Boolean equations. To indicate the source of control of the two contacts, the name DEPEND would be used for each appearance of the variables in the set of Boolean equations for the logic system. The time-sequential effect of the operation of each of the two contacts is determined by their specific delays.

A special Boolean algebra notation is introduced here to specify the time dependence of cause and effect for a variable. Two types of time delay are recognized. The α-delay is defined as the elapsed time between a change of state from F to T of an element and the resulting change from F to T of an associated variable. The β-delay is defined similarly for a change from T to F. These delays are separated by a comma and written as a numerical superscript on the name designating the variable (e.g., $V^{\alpha,\beta}$ is written as DEPEND^{5,3}). Each variable may have a different set of α and β values even though the variables are associated with the same logical element (i.e., a relay may have several different time-delay contacts). The omission of α and β implies that the change of state of the variable occurs instantaneously with that of the control element (i.e., no time delay).

As an example of this extended Boolean algebra notation, refer to the equations given below in the section "Application of Program to a Simplified System." In Eqs. (1–3) switches START and STOP are input elements (manually controlled), and we have a 2-contact relay, CONRLY, and a 4-contact device, TIMER, as dependent elements. Equation (1) defines the state of dependent element HORN as a function of three variables and relative time. Time is explicitly included by means of the α- and β-delays. The * represents the logical operation AND, + represents OR, and the bar over the third variable represents NOT. Note that a time-delay variable TIMER appears in all three equations but with different values of α. These three terms (variables) represent three of the contacts of the timing device, a dependent element whose state is defined by Eq. (2).

The validity of the results of any simulation depends on the accuracy with which the mathematical model represents the physical system. The Boolean program described in this paper is no exception to the rule. The fundamental principles of modeling a logic system are illustrated by the set of Boolean equations (1–13). Since the illustrative control system comprises only single-input electrical devices (relays and timers) connected in series–parallel groups, one can write the Boolean equations more or less by inspection once the schematic diagram, Fig. 3, has been drawn. However, an explanation of some of the details might be desirable.

The term (START + TIMER), in Eq. (2) for the state of dependent element TIMER, represents the fact that START is a momentary push button which starts the timer immediately, but TIMER continues in state T after the push button is released (i.e., after START changes from T to F). The meaning of the time-delay variable $\overline{\text{TIMER}}^{15,0}$ is that although dependent element TIMER changed from F to T state, the effective state of this term will not become F (i.e., complement of T) until 15 seconds later as indicated by the α-delay. A similar time-delay variable $\overline{\text{TIMER}}^{5,0}$ appears in Eq. (1). Therefore, the horn will start blowing when the start push button is pressed, but it will stop automatically 5 seconds later. The variable TIMER^{10,0} in Eq. (3) signifies that dependent element CONRLY changes from F to T state 10 seconds after TIMER = T. Note that STOP is a normally closed push button. The variable CONRLY in the term (CONRLY + TIMER^{10,0}) maintains the element in state T until push button STOP is pressed, even though the timer stopped automatically after running for 15 seconds.

A β-delay of zero is indicated for each time-delay

variable in Eqs. (1–3), which signifies that the "drop-out" action of the contact is instantaneous. For hardware, the delays are never zero but they may be small compared to other time intervals in the sequential operation of the system. Proper simulation may necessitate entering a small value, rather than zero, for such a delay. The behavior of dependent elements HORN, CONRLY and TIMER is independent of the value of β so the use of $\beta = 0$ is satisfactory for the simulation described in Eqs. (1–3). Conveyor A (Fig. 2) must not start until after conveyor B is in motion, and should shut down automatically if conveyor B stops. These requirements are expressed in Eq. (4) by the α- and β-delays on the time-delay variable CNVB[5,3].

A simple illustrative control system was chosen so that the use of α- and β-delays in a Boolean model could be explained. Many control systems incorporate multiple-input devices such as motor switches, latching relays, and flip-flops. The states of such a device should be related to the inputs by means of a Veitch diagram so that the Boolean equation will give a correct state for all combinations of the inputs. A discussion of these techniques for modeling logic devices is presented in the Appendix.

FUNCTIONAL DESCRIPTION OF DIGITAL COMPUTER PROGRAM

The functional organization of the complete program is shown diagrammatically in Fig. 1. The philosophy of the basic control program is similar to that of an earlier program[1] written by the authors in machine language for the IBM 704. In detail, however, the current program differs considerably because it has been coded for a seven-index-register IBM 7094 for operation in the FORTRAN IV IBSYS/IBJOB system, and new features have been incorporated as a result of use of the original program in the analysis of many control systems.[2]

Basic specifications for the current program were 1) efficient use of core and 2) maximum execution speed so that logical systems of 2000 or more elements could be simulated economically. The Boolean structure of the FORTRAN IV source language is not compatible with these specifications. Consequently, the major part of the program is written in the MAP symbolic language.

The Boolean equations which describe the logic system are coded in MAP and assembled as a subroutine named BOOLEQ as illustrated in the following section and also in Fig. 4. The basic control program solves the equations, checks for incompatible states, prints diagnostic comments, and increments time automatically to simulate the operation of the logic system. The desired sequence is directed by 1) states of input elements entered as data at any time, 2) the time delays (which may be

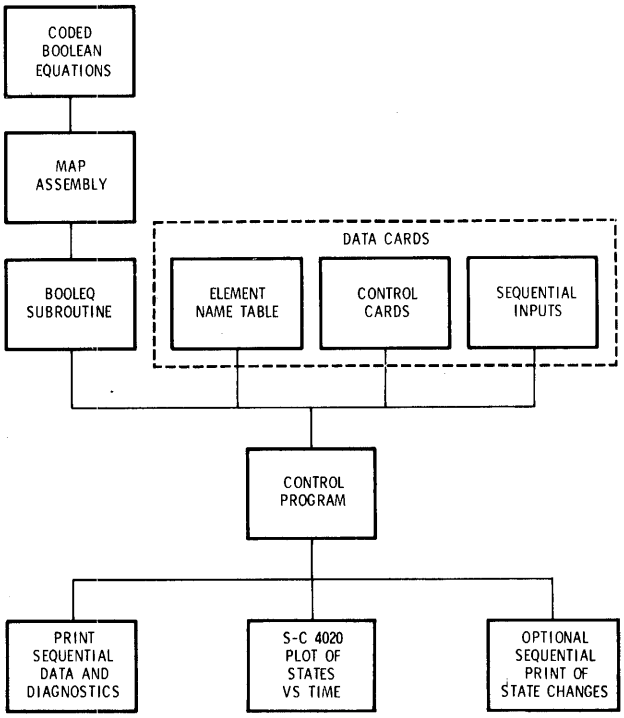


Figure 1. Functional diagram of Boolean simulator program.

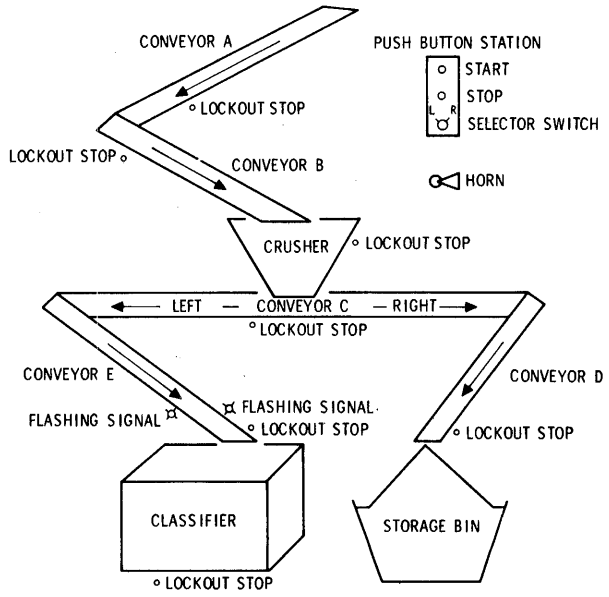


Figure 2. A conveyor system.

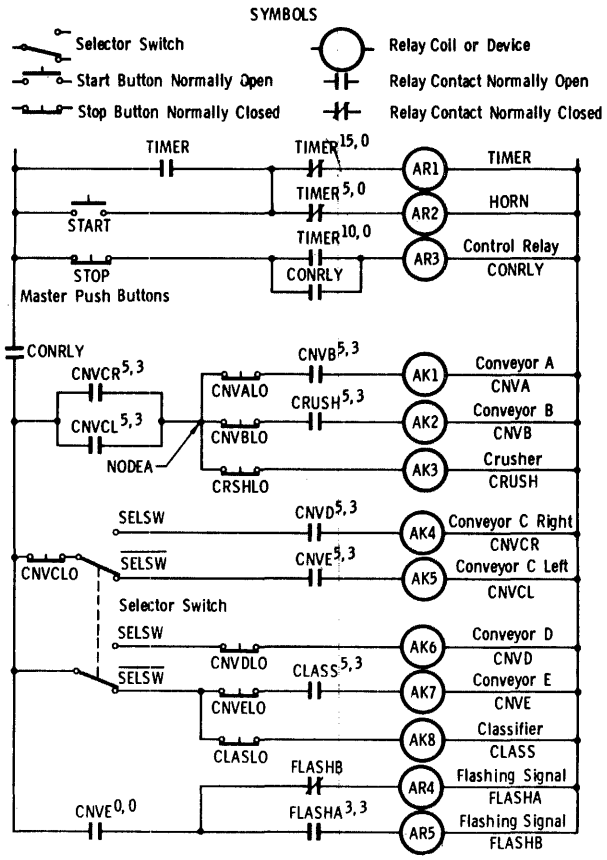


Figure 3. Conveyor system schematic diagram.

varied), and 3) malfunctions simulated by the entry of special data at any time.

At the option of the user, the states of selected elements are compared with specifications (the expected states at that time entered as data) and the results of the comparison are printed. If the specifications are met, the simulation continues. The user may direct the program to continue if a specification is not met; otherwise, it stops. Dependent elements may be "disabled" to a T or F state at any time to simulate the malfunction of a device. Subsequently, they may be "enabled" to simulate a repair or replacement. The "enable" feature is also used to initialize the state of a dependent variable. The normal α- and β-delays are specified in the coding of the equations in subroutine BOOLEQ but they may be varied at any time by a special data entry. At the end-simulation time, entered as initial data, the program plots the states of all the elements on the S-C 4020 as a time-sequential recorder chart. At the option of the user, a time-sequential tabulation of all the changes of state will be printed. Provision is made for entering a brief description of the system which will be printed on the S-C 4020 plot to identify the results.

The simulated operation of the system starts at time zero. The time is incremented variably, the increment being determined at any time by the time delays that are effective or by the entry of data or specifications. This automatic feature of the program relieves the user of the responsibility of analyzing the system and selecting the proper increment at each instant of time and ensures that the simulation will run at the maximum speed compatible with the delays and the inputs of data. At each increment of time, the program determines the stable state of each dependent element by a repetitive solution of the set of Boolean equations. The transition states that appear during these solutions may be used to detect critical and noncritical race conditions as defined by Caldwell.[3] A time-delay subroutine applies the α- and β-delays of a variable to the recorded time at which the associated control element changes state and thus determines the effective state of the variable at each increment of time. These delays permit the analysis of systems using make before break, and break before make, types of contacts.

Debugging routines print diagnostic comments and stop the simulation for indeterminate or incompatible states, or the improper entry of input states and specifications.

APPLICATION OF PROGRAM TO A SIMPLIFIED SYSTEM

The example described below was chosen to illustrate the procedures for using the control program and to show the input data and the printed and plotted results of an analysis of a switching network.

Description of the System

The physical arrangement of the system is shown in Fig. 2. Conveyor A transports ore from a loading area to the plant site. Conveyor B transfers rough ore from conveyor A to the crusher. After being crushed, the fine ore can be directed either through conveyors C and D to a storage bin, or through C and E to a classifier where the ore is separated according to size. Start and stop push buttons and a selector switch, which selects one of the two modes of operation, are provided at the master station. A horn warns workers off the conveyors prior to their movement, and several emergency lock-out push buttons and flashing lights

```
$IBMAP CONVYR                                                                   CCCCC010
*  BOOLEAN EQUATIONS FOR CONVEYOR SYSTEM SWITCHING NETWORK                      CCCCC020
*       MAP ASSEMBLED FOR USE WITH FORTRAN IV DECK BL-030                       CCCCC040
*            0. M. GEORGE, 41-200-300, 4 JUN 1965                               CCCCC050
MAXN    EQU     25              SPECIFY PROPER NUMERICAL VALUE                  CCCCC055
TABLE   CONTRL  TABLE                                                           CCCCC060
        USE     TABLE           ELEMENT-NAMES TABLE                             CCCCC061
        BSS     50              MUST BE MULTIPLE OF 50                          CCCCC062
*                             SET VARIABLE DIMENSIONS FOR ARRAYS                CCCCC070
C       CONTRL  C                                                               CCCCC075
        USE     C                                                               CCCCC076
        BSS     2*MAXN+1        ELEMENT STATES AND TIMES                        CCCCC077
BUFFA   CONTRL  BUFFA                                                           CCCCC078
        USE     BUFFA           INPUT-OUTPUT BUFFER SIZE,                       CCCCC079
        BSS     3*MAXN          BUT NOT LESS THAN 2 TIMES THE MAXIMUM           CCCCC080
*                               NUMBER OF BOOLEAN INPUTS AT ANY TIME            CCCCC081
        USE                                                                     CCCCC082
T       EQU     TEMP            DEFINES T AS ALTERNATE SYMBOL FOR TEMP.         CCCCC084
*                               10 ERASABLE LOCATIONS, TEMP TO TEMP+9           CCCCC085
*                                                                               CCCCC086
*                               FOLLOWING 'EQU' CARDS DEFINE THE                CCCCC087
*                               ELEMENT-NAME TABLE, I.E. THE ASSIGNMENT OF      CCCCC088
*                               INTERNAL SEQUENTIAL ELEMENT NUMBERS.            CCCCC089
START   EQU     C+1                                                             85CCCC01
STOP    EQU     C+2                                                             85CCCC02
SELSW   EQU     C+3                                                             85CCCC03
-------------------------------------------------------------------------------
FLASHB  EQU     C+23                                                            85CCC023
SPARE   EQU     C+24                                                            85CCC024
NODEA   EQU     C+25                                                            85CCC025
*                                                                               CCCCC090
BOOLEQ  SAVE                                                                    CCCCC095
*  START OF BOOLEAN EQUATION NO. 1 FOR DEPENDENT ELEMENT 'HORN'                 CCCCC100
        STR     HORN                                                            CCCCC105
        CAL     START   LOAD LOGICAL ACCUMULATOR WITH STATE OF 'START'          CCCCC110
        ORA     TIMER           LOGICAL 'OR' TO ACCUMULATOR                     CCCCC120
        SLW     TEMP            SAVE TEMPORARILY                                CCCCC130
*           CALL THE DELAY SUBROUTINE FOR CONTROL ELEMENT 'TIMER'               CCCCC140
*           BASIC UNIT FOR ALL DELAYS IS DECISECONDS                            CCCCC141
        CALL    DELAY(TIMER,50,0)                                               CCCCC150
*               ARGUMENTS ARE (CONTROL ELEMENT,ALFA DELAY,BETA DELAY)           CCCCC155
        COM             LOGICAL 'NOT'                                           CCCCC170
        ANA     TEMP            LOGICAL 'AND' TO ACCUMULATOR                    CCCCC180
        TRA     STATE           END OF BOOLEAN EQUATION FOR 'HORN'              CCCCC190
*  BOOLEAN EQUATION NO. 2 FOR 'TIMER'                                           CCCCC200
        STR     TIMER                                                           CCCCC210
        CAL     START                                                           CCCCC220
        ORA     TIMER                                                           CCCCC230
        SLW     T               OPTIONAL USE OF T FOR TEMP                      CCCCC240
        CALL    DELAY(TIMER,150,0)                                              CCCCC250
        COM                                                                     CCCCC270
-------------------------------------------------------------------------------
*  EQUATION NO. 13 FOR FLASHING LIGHT B                                         CCCC1400
        STR     FLASHB                                                          CCCC1401
*           ILLUSTRATES CODED 0,0 DELAYS FOR VARIABLE DELAY FEATURE             CCCC1405
        CALL    DELAY(CNVE,0,0)                                                 CCCC1410
        SLW     T                                                               CCCC1415
        CALL    DELAY(FLASHA,30,30)                                             CCCC1420
        ANA     CONRLY                                                          CCCC1440
        ANA     T                                                               CCCC1450
        TRA     STATE                                                           CCCC1460
        RETURN  BOOLEQ,1        END OF ALL BOOLEAN EQUATIONS                    CCCC1598
        END                                                                     CCCC1599
```

Figure 4. Sample page from subroutine BOOLEQ.

are installed at various points for safety reasons. The relay switching schematic diagram is shown in Fig. 3. Equipments are interlocked with time-delay relays so that no equipments can start until the downstream equipments are in motion. These same interlocks automatically stop the upstream components of the system when a piece of equipment fails. With the time delays and selector switch position shown in Fig. 3, the sequence of major events for a possible operation are given in Table 1.

Procedures for Using the Control Program

1. Assign identifier names (6 or fewer legitimate alphameric characters) to the input and dependent elements (refer to Fig. 3). Prepare a descriptive list of all the elements defining the T and F states for each. For the conveyor system, F means off and T means on, except for the selector switch where F means left and T means right.

2. Write Boolean equations for the dependent elements as illustrated in Eqs. (1–13). Use an * to indicate a logical AND, a + for inclusive OR, a − for exclusive OR, and a bar over the name (or an expression) to indicate NOT.

$$\text{HORN} = (\text{START} + \text{TIMER}) * \overline{\text{TIMER}^{5,0}} \tag{1}$$

$$\text{TIMER} = (\text{START} + \text{TIMER}) * \overline{\text{TIMER}^{15,0}} \tag{2}$$

$$\text{CONRLY} = \overline{\text{STOP}} * (\text{CONRLY} + \text{TIMER}^{10,0}) \tag{3}$$

$$\text{CNVA} = \text{CONRLY} * (\text{CNVCR}^{5,3} + \text{CNVCL}^{5,3}) * \overline{\text{CNVALO}} * \text{CNVB}^{5,3} \tag{4}$$

$$\text{CNVB} = \text{CONRLY} * (\text{CNVCR}^{5,3} + \text{CNVCL}^{5,3}) * \overline{\text{CNVBLO}} * \text{CRUSH}^{5,3} \tag{5}$$

$$\text{CRUSH} = \text{CONRLY} * (\text{CNVCR}^{5,3} + \text{CNVCL}^{5,3}) * \overline{\text{CRSHLO}} \tag{6}$$

$$\text{CNVCR} = \text{CONRLY} * \overline{\text{CNVCLO}} * \text{SELSW} * \text{CNVD}^{5,3} \tag{7}$$

$$\text{CNVCL} = \text{CONRLY} * \overline{\text{CNVCLO}} * \overline{\text{SELSW}} * \text{CNVE}^{5,3} \tag{8}$$

Table 1. A Simulated Operation of the Conveyor System

Time *seconds*		Inputs to, and/or Responses of, System
0	Input:	Press master push button (i.e., input START = T). Selector switch to the left (i.e., = F) for classifier operation is standard initial state; do not enter an INPUT.
0.1	Input:	Set START = F to simulate release of momentary push button.
5	Resp:	Horn stops blowing.
10	Resp:	Control relay CONRLY energized and starts classifier.
15	Resp:	Timer resets to 0, and conveyor E starts together with its associated flashing signals (they flash alternately).
20	Resp:	Conveyor C left drive starts.
25	Resp:	Crusher starts.
30	Resp:	Conveyor B starts.
35	Resp:	Conveyor A starts. All selected equipments operating.
50	Input:	DISABLE conveyor E to off (= F) to simulate a malfunction.
51	Resp:	Flashing signals stop.
53	Resp:	Conveyor C left drive stops.
55	Input:	Press safety lockout button (CNVELO = T). Set selector switch to right (= T) for storage bin operation while conveyor E is being repaired.
	Resp:	Classifier stops; conveyor D starts.
56	Resp:	Conveyors A and B and the crusher stop because the selector switch was changed.
60	Resp:	Conveyor C right drive starts.
65	Resp:	Crusher restarts.
70	Resp:	Conveyor B restarts.
75	Resp:	Conveyor A restarts.
90	Input:	Reset lockout button (CNVELO = F), set selector switch back to left (SELSW = F) and ENABLE conveyor E to off (= F) to simulate restoration of classifier operation after repair of conveyor E.
	Resp:	Conveyors D and C-right-drive stop; classifier starts.
93	Resp:	Conveyors A and B and the crusher stop.
95	Resp:	Conveyor E and associated flashing lights restart.
100	Resp:	Conveyor C left drive starts.
105	Resp:	Crusher starts.
110	Resp:	Conveyor B starts.
115	Resp:	Conveyor A starts.
118	Input:	Push master stop button (STOP = T). Stops entire system.
120	End of simulation.	

$$CNVD = CONRLY * SELSW * \overline{CNVDLO} \quad (9)$$

$$CNVE = CONRLY * \overline{SELSW} * \overline{CNVELO} * CLASS^{5,3} \quad (10)$$

$$CLASS = CONRLY * \overline{SELSW} * \overline{CLASLO} \quad (11)$$

$$FLASHA = CONRLY * CNVE^{0,0} * \overline{FLASHB} \quad (12)$$

$$FLASHB = CONRLY * CNVE^{0,0} * FLASHA^{3,3} \quad (13)$$

Note that conveyor C has been assigned two names: CNVCR for motion to the right, CNVCL for motion to the left. Also note in Eqs. (12) and (13), that the delays for variable CNVE have been specifically indicated as^{0,0}. This was done so that the Boolean equations contain nominal values which can be varied at execution time, like all the other delays, by the entry of special data. This feature is illustrated by the printout in Fig. 6, which shows that the β-delay was changed to 10 deciseconds.

The term [CONRLY * (CNVCR^{5,3} + CNVCL^{5,3})] appears in each of Eqs. (4–6). The code for subroutine BOOLEQ may be simplified by assigning name NODEA (see Fig. 3) to a point in the schematic diagram. Now we write

$$NODEA = CONRLY * (CNVCR^{5,3} + CNVL^{5,3}) \quad (14)$$

so that Eqs. (4–6) can be simplified to

$$CNVA = NODEA * \overline{CNVALO} * CNVB^{5,3} \quad (4')$$

$$CNVB = NODEA * \overline{CNVBLO} * CRUSH^{5,3} \quad (5')$$

$$CRUSH = NODEA * \overline{CRSHLO} \quad (6')$$

A further simplification can be made in the code for subroutine BOOLEQ when one has a string of terms of the form

$$\overline{A} * \overline{B} * \overline{C} * \cdots$$

By using DeMorgan's theorem, Eq. (11) could be written

$$CLASS = CONRLY * \overline{(SELSW + CLASLO)} \quad (11')$$

so that instead of complementing each variable, we OR the variables and complement the result.

3. Code the Boolean equations in the language of the particular digital computer being used. For the IBM 7094, a typical example follows for dependent element HORN, i.e., Eq. (1):

Code	Comment
STR HORN	Start of instructions for the horn
CALL DELAY (TIMER,5,0)	Control element is timer, α-delay is 5, β-delay is 0
COM	Complement the state (i.e., NOT)
SLW TEMP	Save temporarily
CAL TIMER	Load the logical accumulator with state of the timer
ORA START	OR the state of start switch
ANA TEMP	AND the result saved from DELAY
TRA STATE	End of Boolean equation

4. Assemble the code to produce a MAP subroutine named BOOLEQ, that is compatible with, and will be combined with, the machine-language control program (see the sample in Fig. 4). The Boolean equations need not be written or coded in a particular order. If there is a stable set of states at each instant of time, the repetitive solution procedure in the control program will terminate normally; otherwise, a diagnostic message is printed and the simulation is aborted.

INPUTS TO PROGRAM AT EXECUTION TIME

Figure 5 is a sample of the type of data sheets that were prepared for a simulation of the operation of the conveyor system. Note that a set of data cards, terminated by an * in column 1, is prepared for each data-read time. At $t = 0$, the element name table, a special control card and one to six title cards terminated by an * in column 1 must be the first cards of the data deck. These cards are not illustrated. Initial Boolean data are entered as shown on card No. 9. In addition, the basic criteria for the simulation must be entered as shown on card No. 10. An entry on this card gives the next time in basic units (deciseconds in this case) at which data can be read.

Note that element numbers, which are determined by the order of element names in the name table, are used in entering the T or F states of the element. A special data-reading subroutine is used to facilitate the entry of Boolean data in the relatively free form indicated on cards No. 9 and 18. This subroutine also permits the intermixture of Boolean and integer data cards; the type is indicated by a designator in the first field of each card. Subsequent to $t = 0$, it is only necessary to prepare cards for the Boolean data to be read at each data-read time, and

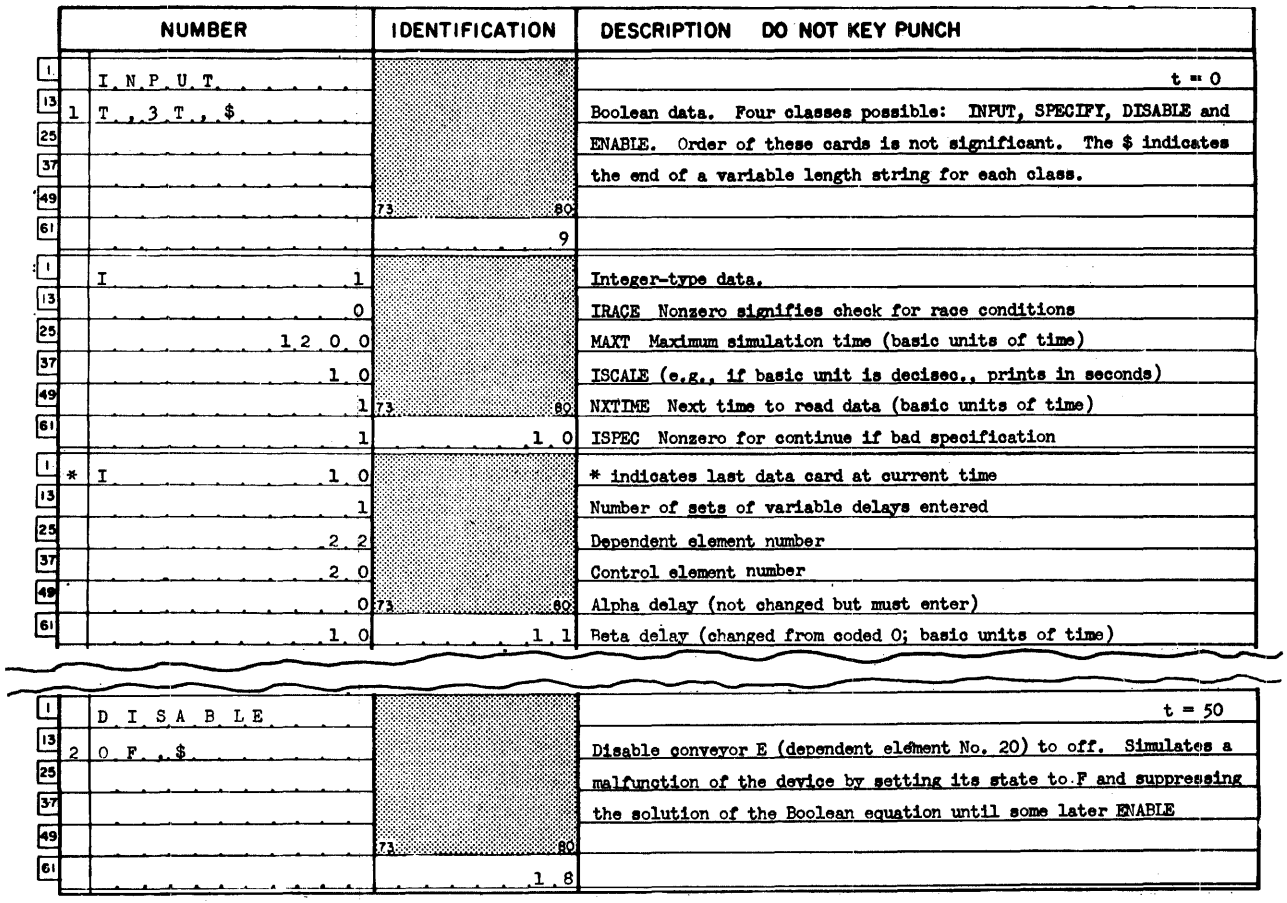


Figure 5. Sample of data sheets.

to designate NXTIME. The simulation is terminated automatically at time MAXT.

OUTPUT FROM EXECUTION OF PROGRAM

For the example shown in Fig. 3 and Table 1, input conditions were entered at times 0, 0.1, 50, 55, 90 and 118 seconds and specifications at various times. Several incorrect specifications were entered to illustrate the bad-specification printout. The data and specifications printout is shown in Fig. 6. The program can print a maximum of six lines of problem description (60 alphameric characters maximum per line). Unused lines are left blank. Note in Fig. 6 that a state was entered at time zero for input variable START only. The program automatically sets all the other states to F. No input element should ever be set to F by data unless it has previously been set to T. Also, note that operation of the spring-loaded master start button START was simulated by a data entry at 0.1 second signifying release of the button.

In addition to the data and specifications printout, the control program plots a complete time history of the states of the elements on the S-C 4020 (see Fig. 8). A print subroutine is provided to enable the engineer to obtain a selective time-sequential printout of the elements which changed state as illustrated in Fig. 7.

SUMMARY

The original program[1] has been used for the analysis of many aerospace control systems[2] and this experience provided the basis for the new program described in the present paper. One result was the development of a special data-reading subroutine to simplify the entry of Boolean data. Another result was a simplification of the printed output which eliminated the searching of records on tape and shortened the execution time. The con-

```
          CHECKOUT OF 8L-020 BOOLEAN ANALYSIS PROGRAM WITH EQUATIONS
                    FOR CONVEYOR SYSTEM SWITCHING NETWORK.
                      BASIC UNIT OF TIME IS DECISECONDS

                    O. M. GEORGE, 41-200-300, 4 JUN 1965


***  TIME          0.0 SECONDS
MAX ELEMENT NO. =   25      MAX TIME =        1200      SCALE =      10
CONTINUE = T     PRINT = T      NEXT CASE = F      SAVE STATE = F      RACE-CHECK INDICATOR = F

 INPUT VARIABLES ENTERED
            ELEMENT        STATE
               1             T

 VARIABLE DELAYS ENTERED
            ELEMENT     CONTROL     ALFA DELAY  BETA DELAY
               22          20             0          10


***  TIME          0.1 SECONDS

 INPUT VARIABLES ENTERED
            ELEMENT        STATE
               1             F


***  TIME          5.0 SECONDS

 SPECIFICATIONS ENTERED
            ELEMENT        STATE
               18            T
               21            T
               11            F
   FOLLOWING SPECS NOT MET
            ELEMENT        STATE   SINCE TIME ( SECONDS
               18            F           0.0
               21            F           0.0


***  TIME         50.0 SECONDS

 DEPENDENT ELEMENTS DISABLED
            ELEMENT        STATE
               20            F
```

Figure 6. Sample printout of inputs and specifications.

veyor system simulation summarized in the plot (Fig. 8) represents a two-minute operation of the system; the IBM 7094 execution time was only eight seconds. This increase in speed of execution was also due to the programming of a greatest-increment selection procedure which relieves the user of the responsibility of analyzing the system and selecting the proper increment at each instant of time. This automatic feature also ensures that no critical event will be missed as a result of making a time step that exceeds any effective delay. An S-C 4020 plot of the results was added which increases the accuracy and speed of analysis of a logical system. In most cases, this plot is sufficient and the maximum speed of execution is obtained by not entering any specifications and by not requesting a printout of the results.

Normally, the production of a digital simulator program involves two steps: creating the model, and then writing the program. SIMSCRIPT[4] is a generalized language that permits the model and simulator program to be written in a terminology that is more oriented to simulation than is the language of FORTRAN. The SIMSCRIPT preprocessor produces a set of routines which can be compiled by FORTRAN to produce machine-language decks for a general-purpose digital computer. We recognize the power of a generalized language for programming simulations, but wish to point out that for the class of problems (analysis and debugging of control systems, the transportation problem, and certain manpower and production scheduling problems) which can be solved by the special program

RESULTS OF BOOLEAN ANALYSIS

*** TIME	0.0 SECONDS									
ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	
*START	T	HORN	T	TIMER	T					
*** TIME	0.1 SECONDS									
ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	
*START	F									
*** TIME	5.0 SECONDS									
ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	
HORN	F									
*** TIME	10.0 SECONDS									
ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	
CONRLY	T	CLASS	T							
*** TIME	15.0 SECONDS									
ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	
TIMER	F	CNVE	T	FLASHA	T					
*** TIME	18.0 SECONDS									
ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	
FLASHA	F	FLASHB	T							
*** TIME	20.0 SECONDS									
ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	
CNVCL	T									
*** TIME	21.0 SECONDS									
ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	ELEMENT	STATE	
FLASHA	T	FLASHB	F							

Figure 7. Sample printout of results of simulation.

described in this paper, only the model needs to be programmed and debugged. The executive program is invariant; it has already been written and checked out for correct operation, simplicity of data entry, appropriate diagnostic messages, and a concise presentation of the results of the simulation.

The program described in this paper has saved thousands of dollars and man-hours by permitting the checkout of a control-system design before time and money are spent in building it. The program has also been used for the analysis of "race" conditions in relay circuits, single-point failures, and the effects of malfunctions or proposed changes in design.

Appendix

TECHNIQUES FOR MODELING LOGIC DEVICES

A logic device is described conventionally by stating the relationship of its outputs to the inputs. In this paper, instead of relating the outputs directly to the inputs, we establish an intermediate state for the device, or so-called dependent element. The outputs are then related to the dependent element by time-delay variables. To relate the states of the element to the inputs, it is necessary to have a complete logical description of the states, otherwise the simulation may give incorrect states to the element for those input conditions which are not defined. A good way to relate states to inputs is to use a Veitch diagram except for single-input devices in which case the state-to-input relationships are sequential (i.e., time-delay relationships) rather than combinational.[3] The model of the element is simply the Boolean equation for the Veitch diagram. The following examples illustrate this procedure.

Motor Switch

This device uses a motor to open or close a set of contacts. There are two inputs, one for the set action and the other for reset. Figure 9 is a schematic diagram of a motor switch in which it

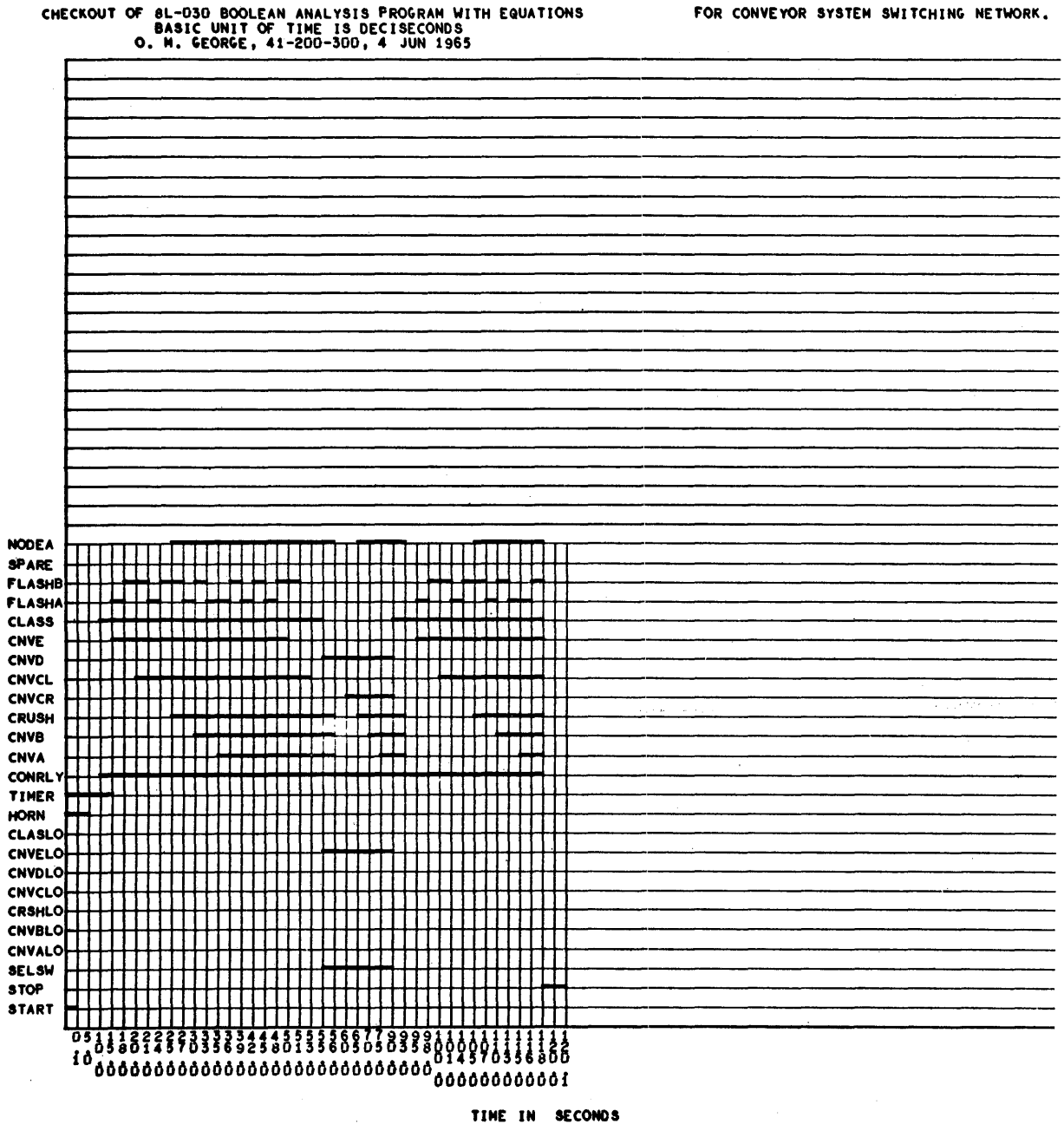


Figure 8. S-C 4020 plot of true states.

takes 10 milliseconds to open or close the contacts. Let us label the components and actions by the following identifiers:

MTRSW	the motor switch
SNODE	input node for set
RNODE	input node for reset
SET	set action of motor
RESET	reset action of motor

The action of the motor can be written very simply as follows:

$$\text{SET} = \text{SNODE} * \overline{\text{MTRSW}}^{0.01,\,0.01} \tag{15}$$

$$\text{RESET} = \text{RNODE} * \text{MTRSW}^{0.01,\,0.01} \tag{16}$$

The state of the motor switch, however, should first be represented by a Veitch diagram

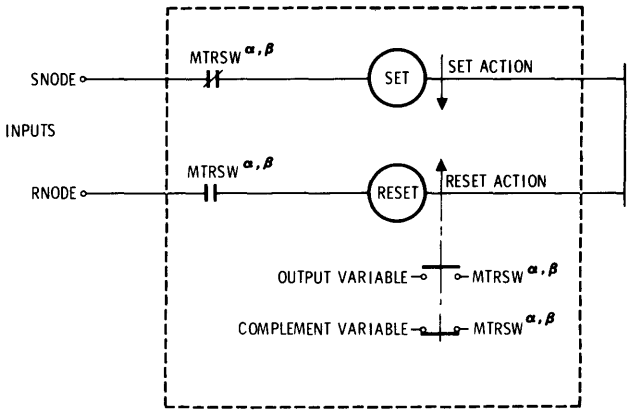


Figure 9. Motor switch.

	SET	$\overline{\text{SET}}$
RESET	impossible	F
$\overline{\text{RESET}}$	T	no change

so that all the possible states are included in the Boolean equation that is written for dependent element MTRSW. Note that one block of the diagram is labeled "impossible" because of the interlocks indicated in Eqs. (15) and (16). The Boolean equation for the motor switch can now be written from the Veitch diagram and comprises two terms as follows:

$$\text{MTRSW} = (\text{SET} * \overline{\text{RESET}}) + (\text{MTRSW} * \overline{\text{SET}} * \overline{\text{RESET}}) \quad (17)$$

The first term specifies the T state, and the second defines "no change" (i.e., it is a memory term). It is not necessary to have a term for the F state because it is implied by the nature of binary algebra.

Magnetic Latching Relay

In this device, two permanent magnets are used in conjunction with a set coil and a reset coil. When both coils are energized or de-energized simultaneously, there is no resultant force to move the switch; it remains in its previous state. When only one coil is energized, the relay is set or reset accordingly. Figure 10 is a schematic diagram of a magnetic latching relay with 5 millisecond α- and β-delays for the output variables. Let us identify the components by the following names:

LATRLY	the latching relay
SNODE	input node for the set coil
RNODE	input node for the reset coil
SET	the set coil
RESET	the reset coil

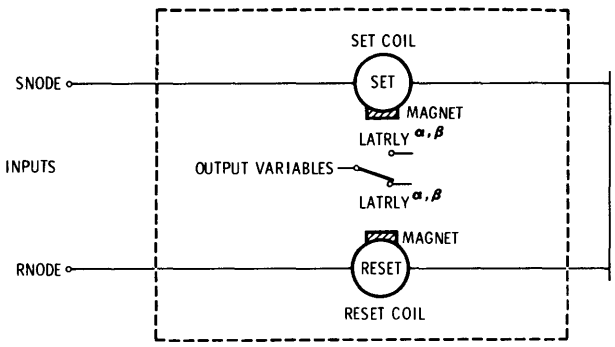


Figure 10. Magnetic latching relay.

The equations for the set and reset coils are

$$\text{SET} = \text{SNODE} \quad (18)$$

$$\text{RESET} = \text{RNODE} \quad (19)$$

The possible states of latching relay LATRLY may be shown in the Veitch diagram

	SET	$\overline{\text{SET}}$
RESET	no change	F
$\overline{\text{RESET}}$	T	no change

from which the Boolean equation

$$\text{LATRLY} = (\text{SET} * \overline{\text{RESET}}) + \text{LATRLY} * [(\text{SET} * \text{RESET}) + (\overline{\text{SET}} * \overline{\text{RESET}})] \quad (20)$$

may be written. This example is similar to that for the motor switch, but it shows how an additional item in the Veitch diagram affects the equation for dependent element LATRLY. The output of the device will appear as a time-delay variable $\text{LATRLY}^{\alpha,\beta}$ in the equations for other elements of the system. In a particular case we might have $\alpha = \beta = 0.005$ seconds.

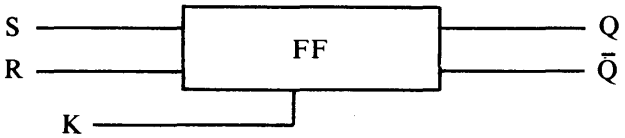
The latching relay is a frequently used component coding of its representation in the Boolean equations for the system, one should code a subroutine for such a device and merely call the subroutine with specific arguments (i.e., element names) each time that type of device is encountered. Figure 11 is the IBM 7094 code for a magnetic latching-relay subroutine.

Three-Input Flip-Flop

The three inputs might be set, reset and clock represented by S, R, and K respectively in the following diagram.

```
$IBMAP LATCH                                                          00C00C00
*              BASIC MAGNETIC LATCHING-RELAY SUBROUTINE               00CC0C01
*       USE AS FOLLOWS                                                CCC00002
*              STR    RELAY                                          00C00C03
*              CALL   LATCH(RELAY,SET,RESET)                         00CCCC04
*              TRA    STATE                                          00CCCC05
*       A PARTICULAR CASE MIGHT BE                                    00CC0C06
*              STR    LR234          BEGIN EQUATION FOR LATCHING RELAY 00CCCC07
*              CALL   LATCH(LR234,LR234S,LR234R)                     00CC0C08
*              TRA    STATE          END EQUATION FOR LATCH RELAY LR234 00CC0C09
       ENTRY   LATCH                                                  00C00010
LATCH  CAL*    4,4                                                    00C00011
       ORA*    5,4                                                    00CC0C12
       COM                                                            00C00013
       SLW     T                                                      00C00014
       CAL*    4,4                                                    00CC0015
       ANA*    5,4                                                    00CC0016
       ORA     T                                                      00C00017
       ANA*    3,4                                                    00C00018
       SLW     T                                                      00CC0C19
       CAL*    5,4                                                    00CC0020
       COM                                                            00C00021
       ANA*    4,4                                                    00CC0C22
       ORA     T                                                      00C00023
       TRA     1,4                  RETURN TO BOOLEAN EQUATIONS       00C00024
T      BSS     1                                                      00C00025
       END                                                            CCC00026
```

Figure 11. Magnetic latching-relay subroutine.



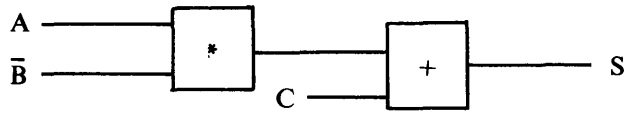
The output Q may be defined by the Veitch diagram

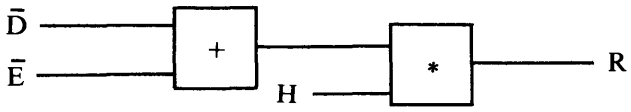
	S	$\bar{S}$	S	$\bar{S}$
R	no change	F	no change	no change
$\bar{R}$	T	no change	no change	no change
	K		$\bar{K}$	

from which the Boolean equation for the clock-controlled flip-flop is

$$Q = (\bar{K} * S * \bar{R}) + Q * [K * (R * S + \bar{R} * \bar{S}) + \bar{K}] \quad (21)$$

This type of flip-flop is used in digital logic design where the set and reset signals S and R might come from gating logic such as



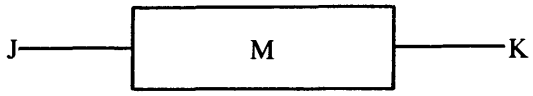


for which the Boolean equations are quite straightforward. For example

$$S = (A * \bar{B}) + C \quad (22)$$

$$R = (\bar{D} + \bar{E}) * H \quad (23)$$

The input signal K to the flip-flop might come from a "one-shot" pulse generator M, a single input device, where J is the input to device M which generates a pulse 0.5 msec wide.



$$M = J \quad (24)$$

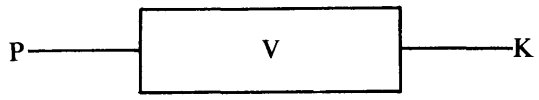
Then

$$K = J * \bar{M}^{0.0005,0} \quad (25)$$

or a simpler expression is

$$K = J * \bar{J}^{0.0005,0} \quad (26)$$

However, if the input K to the flip-flop comes from a 1 – KC clock V,



whose input is P, then

$$K = P * V^{0.0005,\,0.0005} \tag{27}$$

for the signal K, and

$$V = P * \bar{K} \tag{28}$$

for the 1–KC clock.

REFERENCES

1. Y. N. Chang and O. M. George, "Use of High-speed Digital Computers to Study Performance of Complex Switching Networks Incorporating Time Delays," *AIEE Transactions*, vol. 78, pt. I, pp. 982–987 (1960).

2. T. C. Preston and E. A. Estrine, "Spacecraft Electrical System Analysis," *IEEE Proceedings of International Conference on Aerospace Electrotechnology*, vol. AS2, no. 2, pp. 623–629 (1964).

3. S. H. Caldwell, *Switching Circuits and Logical Design*, John Wiley and Sons, New York, 1958.

4. H. M. Markowitz, B. Hausner and H. W. Karr, *SIMSCRIPT, a Simulation Programming Language*, Prentice-Hall, New York, 1963.

SIMULATION OF A MULTIPROCESSOR COMPUTER SYSTEM

Jesse H. Katz
International Business Machines Corporation, Scientific Center
Los Angeles, California

1. INTRODUCTION

Computer simulators have generally been constructed at one of two levels of detail: the instruction level or the bit-time (logic) level. Such simulators have been produced for many years now and their value is well established. By contrast, only minor attention has been given to simulating computer systems at a macroscopic level. One type of macro-level simulator has been reported recently by Hutchinson[1]; in his model the simulated system consists of an entire computation center, with the computer representing merely a component.

As computer systems grow increasingly complex the macro-level modeling of such systems becomes increasingly useful. By applying such a model one can predict the performance of the system on a prescribed job load, and/or evaluate the effect of various parameters on system performance.

In this paper we report on an experimental model that is applicable to a class of multiprocessor operating systems including IBM's Direct-Couple Operating System (DCS). The model has enabled us to evaluate the effect of selected hardware parameters, software parameters and environmental parameters on the performance of a DCS-type multiprocessor operating system. The principal measures of system performance produced by the model are statistics on turnaround time, throughput, equipment utilization, and job queues.

The paper is in eight sections. Section 2 discusses the main features of the existing Direct-Couple Operating System. Section 3 sketches the simulated system treated by the model. Section 4 gives an overview of the simulation system, which consists of two computer programs—the Job Generator and the Simulator. Section 5 discusses the Job Generator, an auxiliary program which generates the properties of specific sets of jobs that are fed to the Simulator. Section 6, the principal section of the paper, discusses the Simulator itself. Section 7 discusses the analysis supporting the specification of the Overhead Analyzer, an important component of the Simulator. And Section 8 summarizes the main findings.

2. MAIN FEATURES OF THE DIRECT-COUPLE OPERATING SYSTEM

A main purpose of the multiprocessor model is to evaluate the Direct-Couple Operating System. Thus DCS, as actually implemented, is of central importance in the model. DCS is described in varying levels of detail in appropriate documents of the IBM Systems Reference Library.[2-4] In this section we discuss its main features. The section is offered as background for subsequent sections of the paper.

The development of DCS is a natural result of two major trends in computer development. One is the trend towards multiprocessing computer sys-

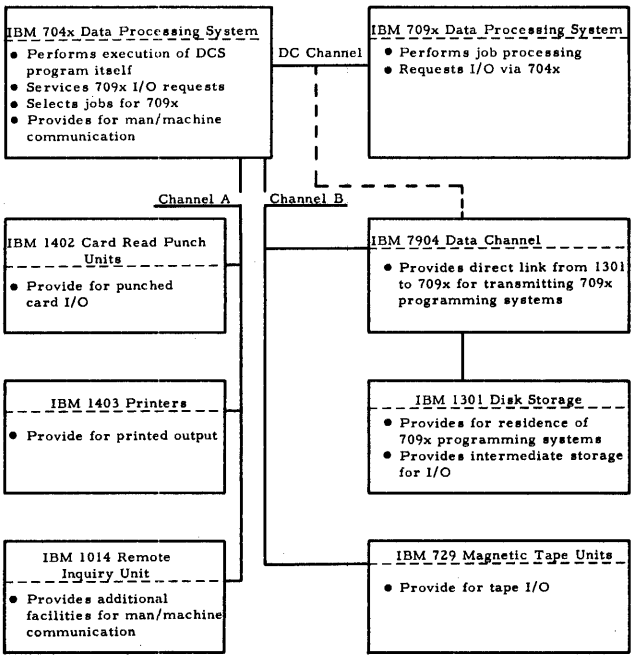


Figure 1. DCS machine configuration.

tems. The other is the trend towards increasingly automatic operating systems. Figure 1 shows the general machine configuration of DCS and the principal functions of the various equipments.

The main advantages of DCS are that 1) it minimizes the need for operator intervention (and generally overlaps such intervention with useful computation) and 2) it processes several jobs in parallel. The parallel processing of jobs makes it possible for the system to time-share its various equipments to a greater extent than is realizable with serial processing.

In general, any given job which DCS processes goes through three *phases*: a preprocessing phase, a processing phase and a postprocessing phase. Each phase consists of one or more *stages*. The preprocessing phase consists of the input and setup stages; the processing phase consists of the execution stage; and the postprocessing phase consists of the breakdown, punch, print and purge stages. An additional stage, the utility stage, is not associated with any of the three phases; it occurs in lieu of the execution stage. The phases and stages, and their principal functions, are shown in Figs. 2, 3 and 4.

With regard to modeling, three properties of DCS stand out in importance.

1. *Commutator Control.* The main loop of the DCS control program consists of a commutator, i.e., a sequence of gates (or switches) which relinquishes control to various parts of the DCS con-

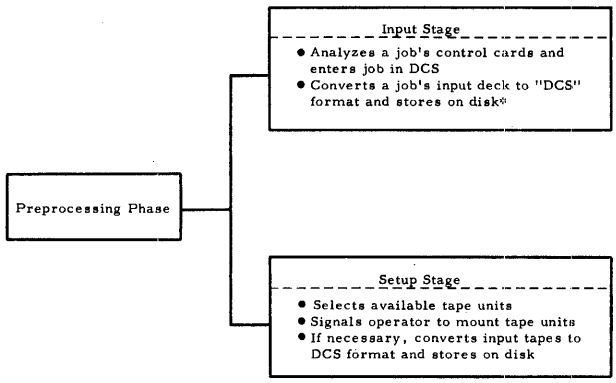


Figure 2. DCS: preprocessing phase.

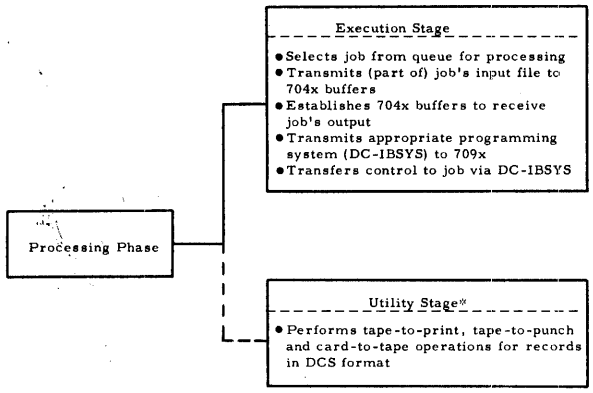


Figure 3. DCS: processing phase.

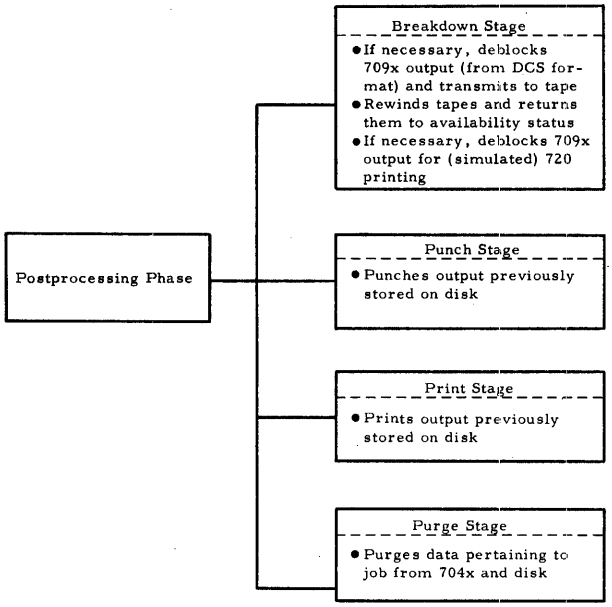


Figure 4. DCS: postprocessing phase.

trol program for *only very short bursts of time.** Therefore, in a strictly imitative model, the model clock would advance in microscopic increments. This renders a strictly imitative approach incompatible with a macro-level model.

2. *Parallel Processing.* The fact that DCS processes jobs in parallel means that the model must view all jobs in the system concurrently. This requirement has led to the system-state approach described in Section 6.

3. *Facility Sharing.* For efficiency purposes certain facilities within a multiprocessor system are generally shared by several jobs. An important instance of facility-sharing in DCS is the sharing of the disk by programs residing on the 709x and 704x. Whenever programs on both computers make competitive demands for the disk, one program is delayed while the other is serviced. Thus, the sharing of facilities is a major contributor to overhead† occurring in a multiprocessor system. The treatment of overhead in the model is of basic importance and is handled by the Overhead Analyzer.

3. THE SIMULATED SYSTEM

The extent of the simulated system is indicated in Fig. 5.

An individual job may be submitted by a programmer at a remote station or a remote terminal; it is at this point in time that simulation of the job begins. Simulation of the job continues until the time the job is returned to the originating station or terminal. Thus, the model is able to give simulated results on turnaround time. The gross events for a job submitted at a station are the following:

- A messenger picks up the job at the station and transmits it to keypunching and/or "central in" of the computer system.
- The job is processed by the computer system proper and stacked at "central out" of the computer system.
- A messenger picks up the job at "central out" and delivers it to the originating station.

For a job submitted at a remote terminal, messenger pickup service is not required; messenger delivery service may or may not be required, depending on whether or not off-terminal output is generated.

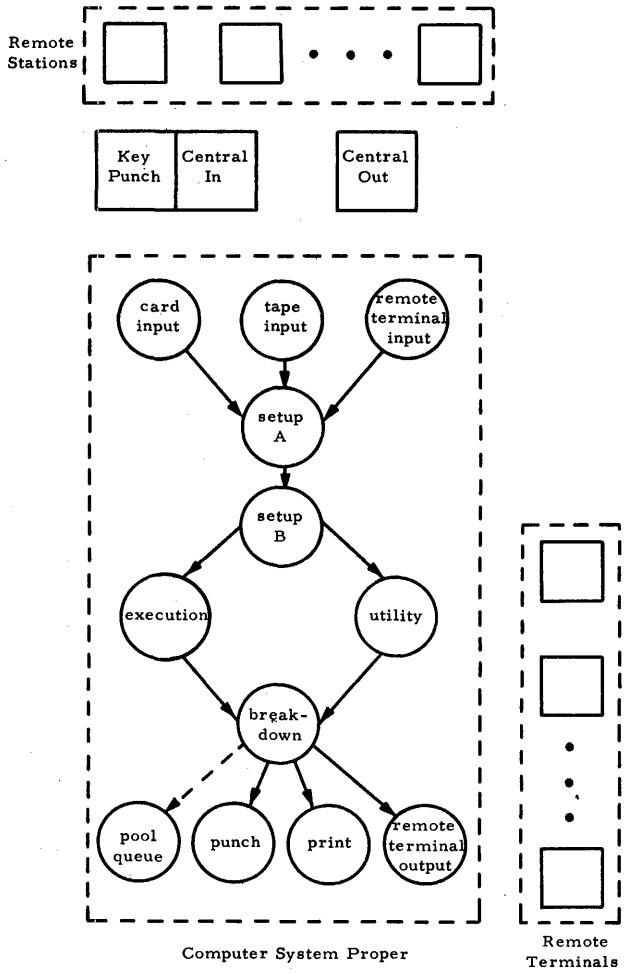


Figure 5. Simulated system.

An individual job is simulated at a more detailed level during its passage through the computer system proper. Here, a job is simulated at the "stage" level. Thus, the model is able to give simulated results on throughput time, i.e., time through the computer system proper. Figure 5, Computer System Proper, shows the 12 stages included in the model. In a typical case a job might be processed by the following stages:

Card input stage—to read job's input onto disk;
Setup A stage—to mount an input tape for the job;
Setup B stage—to convert contents of input tape to an internal processing format;
Execution stage—to perform job's main execution (input from disk and output to disk);
Breakdown stage—to reconvert tape output and release tape;

*A multiprocessor control program is mechanized in this way in order to (try to) keep the various equipments in the system continually busy.

†Overhead, in a general sense, is defined as time not devoted to the performance of directly useful work. Overhead is defined in a numerical sense under "Updating the Matrix" in Section 6.

Punch stage—to punch output stored on disk; and
Print stage—to print output stored on disk.

The arrows shown in Fig. 5 indicate the *precedence relations* among stages. This concept is an important one in connection with a multiprocessor system and will be discussed in Section 6 (under "Software Parameters").

4. OVERVIEW OF THE SIMULATION SYSTEM

The multiprocessor simulation system (Fig. 6) consists of two computer programs: the Job Generator and the Simulator.

The Job Generator is an auxiliary program which generates the properties of jobs that are fed to the Simulator. The program accepts as input the statistical properties of the user's job *population.* Its output consists of parameters that characterize a set of specific jobs; the set represents a *sample* drawn from the user's population of jobs.

The Simulator accepts two general kinds of input: 1) the output produced by the Job Generator and 2) user-supplied input. The latter includes parameters that characterize the hardware system, the software system, and the environment of the computer system. Output from the Simulator consists of various statistics that give measures of system performance.

The Simulator consists of two parts: the Simulator Proper and the Overhead Analyzer. The Overhead Analyzer is subordinate to the Simulator Proper and services the Simulator Proper on demand. The Simulator Proper is virtually independent of the computer system configuration, the configuration-dependence being buried almost exclusively in the Overhead Analyzer.

5. THE JOB GENERATOR

An individual job fed to the Simulator is characterized by 21 parameters:

1. Job identification number.
2. Time-of-arrival in system.
3. Station or terminal at which job arrives.
4. Job priority, e.g., 0,1,2,..., with "zero" the lowest priority.
5. Maximum time in execution stage, as specified by programmer.*
6. Maximum number of lines of printed output, as specified by programmer.
7. Maximum number of cards of punched output, as specified by programmer.
8. Keypunching time.
9. Whether job enters computer system via cards or tape.
10. Whether I/O is direct mode or compatibility mode.†
11. Rate of I/O calls during execution stage.
12. Number of cards of input.
13. Number of characters of remote terminal input.

*Parameters (5), (6) and (7) are programmer-specified cutoff parameters.

†Two modes are available in DCS for data transmission between the 709x and 704x: direct and compatibility. Direct mode I/O conventions take full advantage of DCS facilities whereas compatibility mode I/O conventions do not; thus direct mode transmission is faster. Jobs written in FORTRAN IV, COBOL and MAP operate in the direct mode inasmuch as the IBM 7090/7094 IBJOB Processor operates on DCS in the direct mode. Other kinds of jobs written for the standard 709x Data Processing System operate on DCS in the compatibility mode.

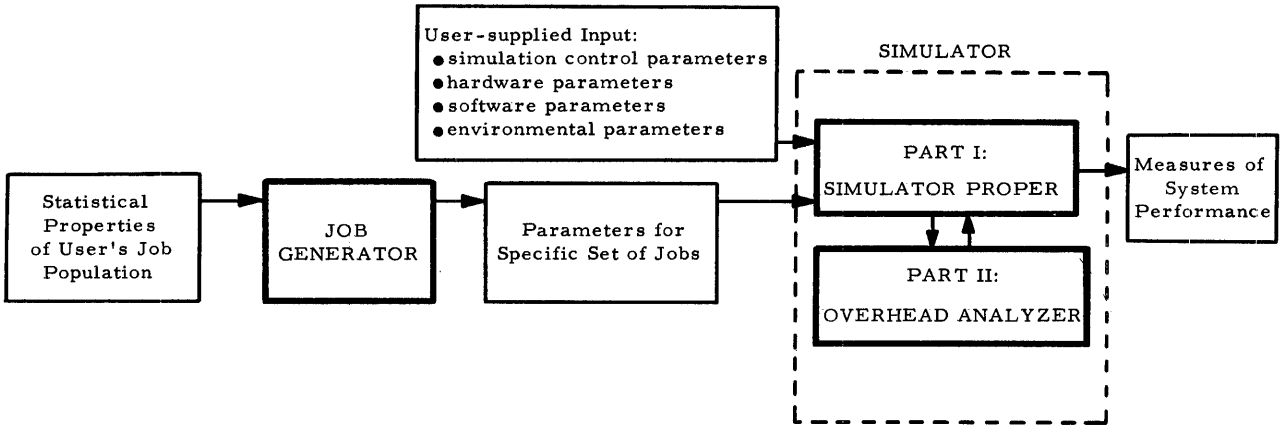


Figure 6. Simulation system.

14. Base time* for setup A stage.
15. Base time for setup B stage.
16. Base time for execution stage.
17. Base time for utility stage.
18. Base time for breakdown stage.
19. Number of cards of punched output.
20. Number of lines of printed output.
21. Number of characters of remote terminal output.

It is the function of the Job Generator to construct sets of job parameters that reflect the actual jobs of a particular user's installation. Thus, the input to the Job Generator consists of statistics on the user's job population. The input consists mostly of 1) frequency distributions for individual job parameters and 2) data specifying correlations among various parameters.

In order to help specify the generator an extensive analysis was made of a full month's actual data of a large aerospace company. The analysis included the computation of frequency distributions, means and standard deviations for all job variables, as well as the computation of scatter diagrams and correlation coefficients for various pairs of variables. In addition, statistical significance tests were made in order to insure that the data sample was sufficiently reliable.

6. THE SIMULATOR

In this section we describe the Simulator itself. The section consists of six parts. The first two discuss input and output of the Simulator respectively. The third part discusses the basic approach of the model, the system-state approach. Part four presents the model logic. The fifth part reviews some basic concepts in Simscript, the language in which the model is mechanized. And the final part describes the program organization of the Simulator.

Simulator Input

In addition to the sets of job parameters fed to the Simulator (Section 5), four other classes of parameters are required: simulation control parameters, hardware parameters, software parameters and environmental parameters.

Simulation Control Parameters. Parameters that control simulation include 1) time simulation begins, 2) time observation begins,† 3) time simulation ends, and 4) options governing simulation output.

Hardware Parameters. Hardware parameters include 1) number of read-punch units, 2) number and types of printers, and 3) number of remote terminals. Thus, hardware parameters serve to specify an equipment configuration.

Software Parameters. Software parameters enable us to study the effect of scheduling jobs within a multiprocessor system under various scheduling policies.

Fixed Versus Dynamic Stage Scheduling. One such parameter is fundamental and specifies the stage scheduling mode: fixed or dynamic. DCS, as implemented, schedules job stages in a fixed sequence, i.e., a given job has its stages executed in an unvarying sequence each time it is run. It is possible, however, for a multiprocessor control program to schedule job stages *dynamically*, i.e., in accordance with on-the-spot conditions within the system. The idea of dynamic scheduling in connection with parallel processing of jobs has been suggested by Leiner et al.[6]

With regard to dynamic scheduling the concept of *precedence relations* is basic. In Fig. 5, if two stages are connected by arrow (e.g., execution and breakdown), then one stage (execution) *must* precede the other (breakdown). However, if two stages are not connected by arrow (e.g., punch and print), then the order in which the stages are executed is immaterial. Under the dynamic scheduling option the model observes the stage precedence relations shown in Fig. 5, with the pool queue stage holding the queue of jobs scheduled for the punch, print, and remote terminal output stages.

Stage Queue Disciplines. Each stage in the simulated computer system has a job queue associated with it. For each job queue there is a *queue discipline* which specifies the basis on which jobs in the queue are ranked for service. The queue disciplines to be invoked at various multiprocessor stages are specified by means of a class of software parameters.

At the print stage, for instance, one might choose to service jobs in the queue 1) on the basis of priority and time-of-arrival in the computer system; 2) on the basis of priority, maximum number of lines of

*Base time for a multiprocessor stage is the processing time for that stage under conditions of "no overhead" in the multiprocessor system. In general, actual processing time for a stage is greater than base time due to the existence of overhead.

†In simulating a traffic system it is sometimes useful to consider the simulation period as consisting of an initialization period followed by an observation period. This enables the user to suppress the gathering of statistics while the system is building up into a more-or-less "steady state." The feature has been used, for example, in the simulation of an automobile traffic network.[5]

printed output, and time-of-arrival in the computer system; or 3) on some other user-specified basis. Each of these options is available by parameter, with option (3) requiring user-supplied code in the program.

Queue discipline options are also available at the execution, punch, terminal input and pool queue stages.

The implementation of queue discipline options in the model raises a question in Simscript. In the terminology of Simscript the problem of ranking jobs in a queue becomes the following: How can the entities belonging to a Simscript set be ranked on the basis of n attributes, given that Simscript provides machinery for ranking on the basis of one attribute only? A discussion of the problem is given in the Appendix.

Cutoff Parameters. A final class of software parameters specifies 1) the maximum time any job can spend in the execution stage; 2) the maximum number of lines of printed output for any job; and 3) the maximum number of cards of punched output for any job. These installation-specified cutoff parameters are used by the multiprocessor system in the event the programmer himself does not supply overriding cutoff parameters.

Environmental Parameters. Environmental parameters specify the environment within which the computer system operates. These parameters include the following: 1) the number of stations in the system, 2) the messenger pickup and delivery schedules at each station, and 3) the messenger transmission times from (to) each station to (from) the computer system.

Simulator Output

Simulator output reports are delivered at intermediate points of simulation as specified by user option and at the end of simulation. Each report contains two types of statistics: updating and cumulative. Updating statistics are those that are gathered since the issuance of the previous output report, while cumulative statistics are those that are gathered since the beginning of observation.

Before describing the items of output on a report, we define four *kinds* of statistics:

1. *Throughput time* for a job is defined as the time it takes for the job to pass through the multiprocessor computer system.
2. *Turnaround time* for a job is defined as the difference between the time the job is returned to a station and the time the job was submitted to the station.
3. Each job is assigned a sequence number when it enters the computer system. Similarly each job is assigned a sequence number when it exits from the computer system. The *computer system service displacement* for a job is defined as the computer system entrance number minus the computer system exit number. The *absolute computer system service displacement* is defined as the absolute value of the computer system service displacement.
4. Corresponding to (3), a job is also assigned sequence numbers when it enters the system at a station and when it leaves the system. This gives analogous definitions for *system service displacement* and *absolute system service displacement*.

The definitions in (3) and (4) make it possible to measure the extent to which the (computer) system deviates from being a first-in, first-out service facility.

A simulator output report contains ten classes of statistics:

1. Overall performance of the system—including the mean, high, low, range and standard deviation for each of the following: throughput time, turnaround time, absolute computer system service displacement, and absolute system service displacement.
2. Job queues at stations.
3. Job queues at the various computer system stages.
4. The activity/inactivity of the various computer system stages.
5. Computer system stage performance, including base time vs overhead time.
6. The activity/inactivity of computer system equipments.
7. Throughput time by priority.
8. Turnaround time by priority.
9. Turnaround time by station.
10. The distribution of jobs within the system.

Figures 7, 8, 9 and 10 illustrate selected sections of output in more detail. Figure 7 illustrates job queues at computer system stages. Figure 8 illustrates computer system stage performance. Figure 9 illustrates activity/inactivity of computer system equipments. And Fig. 10 illustrates throughput time by priority.

In addition to the aggregative statistics above, detailed statistics are collected on each individual job processed by the Simulator.

SECTION 3. STAGE QUEUES

SUBSECTION 3.1 UPDATING

STAGE	NO. OF JOBS IN QUEUE AT START OF REPORTING PERIOD	NO. OF JOBS IN QUEUE AT END OF REPORTING PERIOD	HIGH NO. OF JOBS IN QUEUE DURING REPORTING PERIOD	LOW NO. OF JOBS IN QUEUE DURING REPORTING PERIOD	MEAN NO. OF JOBS IN QUEUE DURING REPORTING PERIOD	TIME QUEUE EMPTY	TIME QUEUE NON-EMPTY	JOB-QUEUE HOURS
(1) CARD INPUT	0	0	26	0	2.7911	2.7561	1.2439	11.1645
(2) TAPE INPUT	0	0	0	0	0.	4.0000	0.	0.
(3) TERMINAL INPUT	0	0	0	0	0.	4.0000	0.	0.
(4) SETUP A	0	0	1	0	0.	4.0000	0.	0.
(5) SETUP B	0	0	0	0	0.	4.0000	0.	0.
(6) EXECUTION	0	26	32	0	8.9819	1.6276	2.3724	35.9276
(7) UTILITY	0	0	1	0	0.	4.0000	0.	0.
(8) BREAKDOWN	0	0	0	0	0.	4.0000	0.	0.
(9) POOL QUEUE	0	0	0	0	0.	4.0000	0.	0.
(10) PUNCH	0	0	4	0	0.0547	3.9076	0.0924	0.2188
(11) PRINT	0	0	2	0	0.0434	3.8839	0.1161	0.1738
(12) TERMINAL OUTPUT	0	0	0	0	0.	4.0000	0.	0.

Figure 7. Job queues at computer system stages.

SECTION 5. STAGE PERFORMANCE

SUBSECTION 5.1 UPDATING

STAGE	STAGE DIMENSION	BASE TIME (1)	OC SYSTEM OVERHEAD TIME (2)	STAGE EXECUTION TIME (3)=(1)+(2)	OVERHEAD FACTOR (4)=(3)/(1)	GROSS MULTIPROCESSING FACTOR (5)=(3)/SYS BUSY T	NET MULTIPROCESSING FACTOR (6)=(1)/SYS BUSY T
(1) CARD INPUT	1	1.3059	0.0200	1.3259	1.0153	0.49	0.49
(2) TAPE INPUT	1	0.	0.	0.	0.	0.	0.
(3) TERMINAL INPUT	1	0.	0.	0.	0.	0.	0.
(4) SETUP A	1	0.4178	0.	0.4178	1.0000	0.16	0.16
(5) SETUP B	1	0.	0.	0.	0.	0.	0.
(6) EXECUTION	1	2.5466	0.	2.5466	1.0000	0.95	0.95
(7) UTILITY	1	0.0849	0.	0.0849	1.0000	0.03	0.03
(8) BREAKDOWN	1	0.	0.	0.	0.	0.	0.
(10) PUNCH	1	0.3067	0.0599	0.3666	1.1955	0.14	0.11
(11) PRINT	3	3.0251	0.3463	3.3714	1.1145	1.26	1.13
(12) TERMINAL OUTPUT	1	0.	0.	0.	0.	0.	0.

Figure 8. Computer system stage performance.

SECTION 6. EQUIPMENT ACTIVITY/INACTIVITY

SUBSECTION 6.1 UPDATING

EQUIPMENT	DISTRIBUTION OF OBSERVATION TIME BY EQUIPMENT BUSY/IDLE				DISTRIBUTION OF SYSTEM BUSY TIME BY EQUIPMENT BUSY/IDLE			
	TIME		PERCENTAGE		TIME		PERCENTAGE	
	BUSY	IDLE	BUSY	IDLE	BUSY	IDLE	BUSY	IDLE
709X DATA PROCESSING SYSTEM	2.5466	1.4534	63.7	36.3	2.5466	0.1391	94.8	5.2
704X DATA PROCESSING SYSTEM	2.6857	1.3143	67.1	32.9	2.6857	0.	100.0	0.
CARD READER 1	1.3259	2.6741	33.1	66.9	1.3259	1.3598	49.4	50.6
CARD PUNCH 1	0.3666	3.6334	9.2	90.8	0.3666	2.3191	13.7	86.3
PRINTER 1	1.6554	2.3446	41.4	58.6	1.6554	1.0303	61.6	38.4
PRINTER 2	1.2446	2.7554	31.1	68.9	1.2446	1.4411	46.3	53.7
PRINTER 3	0.4714	3.5286	11.8	88.2	0.4714	2.2143	17.6	82.4
REMOTE TERMINAL 1	0.	4.0000	0.	100.0	0.	2.6857	0.	100.0

Figure 9. Activity/inactivity of computer system equipments.

SECTION 7. THROUGHPUT TIME BY PRIORITY

PRIORITY	NUMBER OF JOBS	THROUGHPUT TIME (MINUTES)				
		MEAN	HIGH	LOW	RANGE	STANDARD DEVIATION
0	81	30.79	78.63	0.55	78.08	21.52
1	3	10.96	18.70	3.13	15.57	6.36

Figure 10. Throughput time by priority.

The System-State Approach

In simulating a multiprocessor system at a macro-level we have rejected a strictly imitative approach and have constructed a model based upon the *system-state* approach. The basic concept in this approach is that the multiprocessor computer system is considered to "change state" whenever any job in the computer system completes any stage or whenever a new job enters the computer system. Each such change of state is accompanied by an advance in the model clock.

The system-state approach yields four important benefits:

1. The model clock is advanced in the largest possible increments, consistent with a faithful simulation. Consequently, the resulting model is at an appropriate level of abstraction and its running time is relatively short.

2. The system-state approach is conceptually straightforward whereas a strictly imitative approach would have been conceptually difficult. By definition, a strictly imitative approach is one in which the model logic resembles in very large measure the logic of the multiprocess or control program itself. Hence, a strictly imitative approach leads to model logic which tends to become as complex as multiprocessor control program logic.

3. In order to characterize multiprocessor overhead in the model, it suffices to measure actual overhead empirically, i.e., by observing the *effects* of overhead in an actual multiprocessor system. This kind of measurement has turned out to be feasible. The kind of measurement required in support of a strictly imitative model might not have been feasible. A strictly imitative model would have required an extensive analysis and timing of the multiprocessor control program itself.

4. The model is relatively independent of the computer system configuration being modeled. The model can be viewed as consisting of two components: a large-sized configuration-independent component and a small-sized configuration-dependent component. Since these components are practically distinct, the effect of making a change to the computer system configuration can be determined on the basis of making a corresponding change to the appropriate part of the configuration-dependent component of the model.

Model Logic

In the system-state approach the state of the multiprocessor computer system is represented by a 7-by-12 matrix (Fig. 11). The 12 columns of the matrix represent the 12 stages of the model. The movement of a job through the computer system is represented in the model by the movement of a job from column-to-column of the matrix.

Not all elements of the matrix are relevant. Some elements are never relevant; some are relevant for

		1	2	3	4	5	6	7	8	9	10	11	12
	stage / row vector	card input	tape input	terminal input	setup A	setup B	execution	utility	breakdown	pool queue	punch	print	terminal output
1	d = dimension	X	X	X	X	X	X	X	X	X	X	X	X
2	p = queue discipline	X	X	X	X	X	X	X	X	D	F	F	F
3	q = job queue	X	X	X	X	X	X	X	X	D	F	F	F
4	e = job execution	X	X	X	X	X	X	X	X	O	X	X	X
5	t = base time remaining	X	X	X	X	X	X	X	X	O	X	X	X
6	v = overhead accumulation	X	X	X	X	X	X	X	X	O	X	X	X
7	s = successor stages	X	X	X	X	X	X	X	X	D	X	X	D

Figure 11. System-state matrix.

fixed stage scheduling only; and some are relevant for dynamic stage scheduling only.

The System-State Matrix. The system-state matrix consists of seven 12-dimensional row vectors.

1. *Dimension Vector.* Let d_i denote the ith element of the dimension vector d. The value d_i specifies the number of jobs which the ith stage can process concurrently. We have

$$d = (a, 1, c, 1, 1, 1, 1, 1, 0, a, b, c)$$

where a = number of read-punch units, b = number of printers, and c = number of remote terminal units. Note that $d_9 = 0$ since no job can be processed by the pool queue stage which is merely a queue for punch, print, and remote terminal output under dynamic stage scheduling.

2. *Queue Discipline Vector.* Let p_i denote the ith element of the queue discipline vector p. The value p_i is a code which specifies the queue discipline to be invoked to rank the jobs in the queue belonging to the ith stage.

A discussion of queue disciplines is given above (under "Software Parameters").

3. *Job Queue Vector.* Let q_i denote the ith element of the job queue vector q.

For $i \neq 9$, q_i identifies the set of jobs waiting for ith stage execution. Thus q_i is a vector. The job identification numbers constituting this vector are ordered according to p_i, the queue discipline for ith stage execution.

For $i = 9$, q_i identifies the set of jobs waiting for 10th, 11th or 12th stage execution. Again q_i is a vector. This vector is relevant only in the case of dynamic stage scheduling. The job identification numbers constituting this vector are ordered according to p_9.

4. *Job Execution Vector.* Let e_i denote the ith element of the job execution vector e.

Then e_i, $i \neq 9$, is itself a vector whose jth element is denoted $e_{i,j}$, $j = 1, \ldots, d_i$. If $e_{i,j} > 0$, then $e_{i,j}$ is the identification number of the job being executed in the jth position* of the ith stage. If $e_{i,j} = 0$, then no job is being executed in the jth position of the ith stage.

The value e_9 is null.

5. *Base Time Remaining Vector.* Let t_i denote the ith element of the base time remaining vector t.

Then t_i, $i \neq 9$, is itself a vector whose jth element is denoted $t_{i,j}$. The value $t_{i,j}$ specifies the base time remaining for the job in the jth position of the ith stage.

The value t_9 is null.

6. *Overhead Accumulation Vector.* Let v_i denote the ith element of the overhead accumulation vector v.

Then v_i, $i \neq 9$, is itself a vector whose jth element is denoted $v_{i,j}$. The value $v_{i,j}$ specifies the overhead time which has accumulated for the job in the jth position of the ith stage.

The value v_9 is null.

7. *Successor Stages Vector.* Let s_i denote the ith element of the successor stages vector s.

Then s_i is itself a vector whose jth element is denoted $s_{i,j}$. The value $s_{i,j}$ is in turn a vector that specifies the successor stages for the job in the jth position of the ith stage.

Updating the Matrix. The essential steps in updating the system-state matrix are the following:

1. *Overhead Factor Vector.* In accordance with relevant properties of the system-state currently existing, the subroutine called the Overhead Analyzer computes the overhead for each job being processed. Relevant properties include 1) the particular stages that are active; 2) the number of jobs that are active in each of those stages; and 3) the input/output properties of active jobs. The overhead factor for the job in the jth position of the ith stage is designated $f_{i,j} \geq 1$.

An example of an overhead factor is the following: Assume "printer 2" has a maximum rate of 1100 lpm and assume that due to overhead existing in the current system-state its actual rate is 1000 lpm. Then $f_{11,2} = 1.1$.

The analysis supporting the specification of the Overhead Analyzer is discussed in Section 7.

2. *Potential Advancement of Model Clock.* The potential advancement in the model clock is

$$w = \min_{i,j} \{t_{i,j} f_{i,j}\},\ e_{i,j} > 0,\ j = 1, \ldots, d_i,\ i = 1, \ldots, 12$$

No job in the system will complete its current stage until this amount of time passes.

3. *Actual Advancement of Model Clock.* The actual advancement in the model clock is

$$r = \min \{w, z\}$$

where z is the amount of time remaining before the next job enters the computer system.

*The number of "positions" of the i^{th} stage equals d_i.

4. *The New Base Time Remaining Vector.* Let

$$t^{(g)} = (t_i^{(g)}) = ((t_{i,j}^{(g)}))$$

designate the value of the t vector following execution of the gth system-state. And let

$$e^{(g)} = (e_i^{(g)}) = ((e_{i,j}^{(g)}))$$

designate the value of the e vector following execution of the gth system-state. Then the value of the t vector following execution of the $(g + 1)$th system-state is

$$t^{(g+1)} = (t_i^{(g+1)}) = ((t_{i,j}^{(g+1)}))$$

where

$$t_{i,j}^{(g+1)} = \begin{cases} 0, & e_{i,j}^{(g)} = 0 \\ t_{i,j}^{(g)} - \dfrac{r}{f_{i,j}}, & e_{i,j}^{(g)} > 0 \end{cases}$$

5. *The New Overhead Accumulation Vector.* Let

$$v^{(g)} = (v_i^{(g)}) = ((v_{i,j}^{(g)}))$$

designate the value of the v vector following execution of the gth system-state. Then the value of the v vector following execution of the $(g + 1)$th system-state is

$$v^{(g+1)} = (v_i^{(g+1)}) = ((v_{i,j}^{(g+1)}))$$

where

$$v_{i,j}^{(g+1)} = v_{i,j}^{(g)} + r - t_{i,j}^{(g)} + t_{i,j}^{(g+1)}$$

6. *Inter-Stage Movement of Jobs.* Any job whose stage is complete is moved from its current stage to its new stage queue. As many jobs as possible are moved from stage queues to stage execution.

Review of Simscript Concepts

The Simulator has been mechanized using the Simscript language and hence is structured within the general framework provided by Simscript. The reader is referred to Dimsdale and Markowitz[7] for a description of Simscript or to Markowitz et al[8] for a complete reference manual on the language. In this section we review by example the basic Simscript concepts, namely, the concepts of temporary entity, permanent entity, attribute, set, exogenous event and endogenous event. Entities, attributes and sets depict the status of the simulated system, and events cause changes to the status.

An example of a type of temporary entity is *job*. Each job fed to the Simulator is an entity; it is temporary since in general the job is not present in the system during the full course of simulation.

An example of a type of permanent entity is *printer*. Each printer in the simulated computer system is an entity; it is permanent since it exists in the system for the full course of simulation.

Examples of attributes are "number of cards of input" and "time printer busy"; the former is an attribute of *job* while the latter is an attribute of *printer*.

An example of a type of Simscript set is *stage queue*. Each of the 12 stage queues is a set. The members of each set are the jobs in the queue.

An exogenous event is an event caused by forces outside the boundary of the simulated system. An example is the event ENTER, which marks the entrance of a job in the system.

An endogenous event is an event caused by preceding events within the boundary of the simulated system. An example is the event DONE, which marks the completion of a job on the computer.

Program Organization

The Simulator consists of two phases that are executed serially. Phase I performs the simulation and writes output data on disk, and Phase II delivers the output reports constructed from this data.

Data Description. The data description of the Simulator is expressed by means of 2 types of temporary entities and their attributes, 10 types of permanent entities and their attributes, and 6 types of sets.

Temporary Entities. A temporary entity JOB exists for each job in the simulated system. This entity is described by a total of 46 attributes. A temporary entity BATCH exists for each batch of jobs being carried 1) from station to keypunching and/or "central in" and 2) from "central out" to station. This entity is described by a total of three attributes.

Permanent Entities. A permanent entity STAGE (with 29 attributes) exists for each of the 12 stages in the computer system. A permanent entity STATION (with 27 attributes) exists for each station in the system. A permanent entity READ/PUNCH (with four attributes) exists for each read/punch unit in the computer system. A permanent entity PRINTER (with three attributes) exists for each printer in the computer system. And a permanent entity PRIORITY (with 10 attributes) exists for each job priority level.

In addition to the above, four more permanent entities serve array-dimensioning functions.

Finally, the (implied) permanent entity SYSTEM is described by a total of 94 attributes.

Sets. A set STATION QUEUE is owned by each station in the system. A set KEYPUNCH QUEUE is owned by the SYSTEM. A set STAGE QUEUE is owned by each stage in the computer system. A set EXECUTION QUEUE is owned by the SYSTEM. A set CENTRAL OUT QUEUE is owned by each station in the system. And a set BATCH QUEUE is owned by each batch of jobs in the system.

The member entities of each of the above sets are JOB's.

Program Logic. The logic of the Simulator is expressed by means of 2 exogenous event routines and 10 endogenous event routines. For supporting logic the event routines, in turn, call upon 28 subroutine subprograms, 3 function subprograms and 10 report subprograms.

Exogenous Event Routines. The routine GO performs program initialization. This routine is the first in the program to be executed and is executed once only. The routine ENTER is executed each time a job is entered in the simulated system.

Endogenous Event Routines. The routine LOOK is executed at the beginning of the observation period. This routine performs initialization for statistics gathering. The routine STOP is executed at the conclusion of the simulation period. This routine terminates Phase I of the Simulator and calls report-writing Phase II. The routine STAT is executed each time a report is to be issued. This routine generates output data for the report. The routine TO is executed each time a messenger picks up a batch of jobs at a station for transmittal to keypunching and/or "central in" of the computer system. The routine KEY is executed each time a batch of jobs arrives at keypunching. The routine ON is executed whenever there arrives at the computer system 1) a batch of jobs direct from a station; 2) an individual job via keypunching; or 3) an individual job via a remote terminal. The routine DONE is executed each time the computer system completes processing of a job. The routine FROM is executed each time a messenger picks up a batch of jobs at "central out" for transmittal to a station. The routine EXIT is executed each time a job exits from the simulated system. And the routine STEP is executed at the conclusion of each system-state. This routine carries out the logic indicated under "Model Logic" above.

7. ANALYSIS OF MULTIPROCESSOR OVERHEAD

The specification of the Overhead Analyzer represented a major phase of the modeling process. The specification was based on an intensive, empirical analysis of the occurrence of overhead in an actual multiprocessor system (DCS).

Modification to DCS Operating System

In order to measure actual DCS overhead, the DCS Operating System was modified so that it produces a binary tape containing a sequence of "time-stamps." The time-stamps are created during DCS operation and stored in 460-word data buffers, just like other output. The resulting time-stamp data provides a profile of actual system-states and a profile of individual jobs passing through the computer system.

Prior to modifying DCS we consulted with DCS experts on the question of what effect the collection of time-stamp data would have on actual DCS operation. Their judgment was that DCS operation would be affected in only a negligible way and that the act of observing DCS "from the inside" would not affect significantly our observation results.* This judgment was shown to be correct by an actual experiment carried out following modification of the system. The experiment consisted of running a set of jobs twice—under standard DCS and under modified DCS. The running times differed by 15 hundredths of 1 percent—specifically 3 seconds in some 34 minutes.

Time-Stamp Data

Entries on the time-stamp tape are of three types: state identifiers, events and I/O counts.

State Identifiers. Each occurrence of a change-of-state in DCS causes a time-stamp entry identifying the new system-state.

Events. Each occurrence of an *event* in DCS causes an event time-stamp entry. An event time-stamp consists of four components: 1) type of event, 2) buffer saturation indicator, 3) job number identifying the job associated with the event, and 4) time of occurrence. The principal types of events time-stamped are the following:

*An exception to this is the occurrence of buffer saturation; the collection of time-stamps induces buffer saturation somewhat earlier than normal.

- Reading one card from the ith reader, $i = 1,2,\ldots$
- Printing one line on the ith printer, $i = 1,2,\ldots$
- Punching one card on the ith punch unit, $i = 1,2,\ldots$
- Typing one character.
- Reading one card image from tape.
- Beginning/ending of DCS purge stage.
- Beginning/ending of system load.
- Beginning/ending of library load.
- Beginning/ending of dump.

I/O Counts. Each occurrence of an end-of-system-state on DCS causes seven time-stamp entries giving counts of input/output activity that occurred during the preceding system-state:

- Number of 709x reads.
- Number of 709x writes.
- Number of 709x non-data selects.
- Number of 709x input buffer loads.
- Number of 709x output buffer loads.
- Number of 704x input buffer loads.
- Number of 704x output buffer loads.

Analysis of Time-Stamp Data

Using routines that print and plot time-stamp data and that compute overhead factors for specified system-states, we have been able to construct appropriate statistical distributions for inclusion in the Overhead Analyzer of the model.

Table 1 illustrates four sets of overhead factors collected in 5-second intervals from a 7094/7044 installation. Each of the four columns represents a cumulative frequency distribution. The meaning of a typical entry in this table, e.g., entry 45 in column 2, is as follows: Consider the case where jobs on the 7094 are in the compatibility mode and issue I/O calls at a rate of less than 4-per-second, and compute overhead factors for the 1100 line-per-minute printer every 5 seconds. Then 45 percent of these overhead factors are 1.04 or less.

Table 1. Illustrative Overhead Factors

Overhead Factor	Cumulative Percentage			
	Reader	Printer (1100 lpm)		
	Compatibility			Direct
		<4 I/O calls per sec	>10 I/O calls per sec	
	(1)	(2)	(3)	(4)
1.01	64	12	0	4
1.02	76	26	2	10
1.03	78	36	3	22
1.04	80	45	9	26
1.05	87	49	14	30
1.08	89	59	19	36
1.15	99+	75	27	52
1.30		84	42	68
1.50		91	61	84
2.00		98	77	93
2.50		99+	91	97
3.00			97	99+
3.50			99+	

8. SUMMARY

With the advent of multiprocessor computer systems the prediction of computer system performance on a prescribed job load has become a problem of considerable complexity. This paper has described a model whose principal purpose is to ease this problem. A second important purpose of the model is to determine the effect of varying basic system parameters—hardware, software and environmental.

The model is at a macroscopic level, i.e., it attempts a relatively high degree of abstraction of the real system. This level of simulation has been made possible in connection with a multiprocessor system as a result of using the system-state approach, the main idea in the model. With simulation at a macro-level the running time of the program is attractively short. For a sample of the runs completed to date the ratio of real time to simulated time is 195. That is, a typical 16-hour workload can be simulated in less than 5 minutes. The 5 minutes here refers to 7094 time using a 7094/7044 Direct-Couple System to host the simulation.

Following are some representative questions to which the model has helped provide answers:

- For a given equipment configuration and a specified job load, what improvement in throughput can be achieved using dynamic stage scheduling rather than fixed stage scheduling?
- If all jobs submitted from Station 1 are assigned priority level 9 (highest priority) rather than their currently assigned priorities, what change will result in the mean throughput time (and high throughput time) for jobs submitted from each of the individual stations?

- If a third printer is added to a given two-printer equipment configuration, what change will result in the mean number (and high number) of jobs in the print stage queue? What change will result in printer equipment utilization? Answer questions two ways—assuming third printer 600 lpm and 1100 lpm.
- For a given equipment configuration and a specified job load, suppose the queue discipline for the execution stage is changed from 1) priority and time-of-arrival to 2) priority, maximum time in execution stage, and time-of-arrival. What change will result in the mean throughput time (and high throughput time)? What change will result in the mean (high) absolute computer system service displacement?
- Suppose messenger service is improved by adding one messenger and by making prescribed changes in the messenger schedule. What change will result in the turnaround time at each station?

An indication of the level of effort of the multiprocessor simulation project is the amount of programming involved. The Simulator itself consists of some 2650 source cards in Simscript. The Job Generator consists of some 375 source cards, also in Simscript. The modification to DCS consists of some 650 source cards in MAP. And the routines that analyze the time-stamp tape produced by the modified DCS consist of some 2175 source cards, in FORTRAN and Autocoder.

ACKNOWLEDGMENTS

I am grateful to R. A. Rock and L. A. Verret for their collaboration on the multiprocessor simulation project, to H. Jacobs for his consulting services on the project, and to B. Dimsdale for his counsel and encouragement.

Appendix

RANKING A SET ON n ATTRIBUTES

Simscript provides automatic machinery for ranking the entities of a set on the basis of *one* attribute. It is sometimes necessary, however, to rank the entities of a set on the basis of n attributes. In such a case one can employ a function that maps n attribute values into a "composite attribute value" and rank the set on the basis of the composite attribute.

Consider, for example, the following problem:

Let n equal the number of attributes on which the ranking is to be based.

Let the ith attribute "outrank" in importance the $(i + 1)$th attribute, $i = 1, \ldots, n - 1$.

Let x_i denote the value of the ith attribute; assume x_i is positive integer-valued, with its maximum m_i; i.e., $x_i = 1, 2, \ldots, m_i$.

Then the n attribute values $(x_1, \ldots, x_n)$ can be mapped into an appropriate composite attribute value by means of the function

$$f(x_1, \ldots, x_n) = z_n + \sum_{i=1}^{n-1} z_i \prod_{j=i+1}^{n} m_j$$

where $z_i = x_i$ if ith attribute is ranked "high" in Simscript sense $(i = 1, \ldots, n)$, and $z_i = m_i - x_i$ if ith attribute is ranked "low" in Simscript sense.

REFERENCES

1. G. K. Hutchinson, "A Computer Center Simulation Project," *Comm. ACM,* vol. 8, no. 9, pp. 559–568 (1965).

2. IBM 7090/7040 Direct-Couple Operating System: Operator's Guide, IBM Systems Reference Library, C28-6384.

3. ———: Programmer's Guide, ibid, C28-6382.

4. ———: System Programmer's Guide, ibid, C28-6383.

5. J. H. Katz, "Simulation of a Traffic Network," *Comm. ACM,* vol. 6, no. 8, pp. 480–486 (1963).

6. A. L. Leiner et al, "Organizing a Network of Computers to Meet Deadlines," *Proceedings of the EJCC,* 1957, pp. 115–128.

7. B. Dimsdale and H. M. Markowitz, "A Description of the Simscript Language," *IBM Systems Journal,* vol. 3, no. 1, pp. 57–67 (1964).

8. H. M. Markowitz, B. Hausner and H. W. Karr, *Simscript: A Simulation Programming Language,* Prentice-Hall, Englewood Cliffs, N. J., 1963.

MARKOVIAN MODELS AND NUMERICAL ANALYSIS OF COMPUTER SYSTEM BEHAVIOR*

Victor L. Wallace
Systems Engineering Laboratory
The University of Michigan, Ann Arbor, Michigan
and
Richard S. Rosenberg
Logic of Computers Group
The University of Michigan, Ann Arbor, Michigan

INTRODUCTION

The advent of multiple-access computing, the increasing variety of processors in systems, and the growing use of multiprocessing and multiprogramming in computing systems has put many new burdens on the computing system designer-planner-programmer. The selection of suitable system structure and programming structure, as well as the selection of scheduling rules, requires a much more detailed and precise understanding of the stochastic behavior of system traffic than has been required in the past. In short, the "architect" of system hardware and software is finding a need for more and more insight into the behavior of computers as networks of queues and processors.

As a rule, his chief tool for obtaining this insight has been by the use of Monte Carlo simulations. However, as the systems gain in complexity, and as system design becomes more sensitive to the effects of congestion, these simulations become either too expensive or their estimations of probability too imprecise to be viable as tools for exploring system behavior in depth. Accuracy can be obtained only at the expense of very large samples. Exploration means even further calculation as parameter values and structures are changed and solution repeated. Even if everything has been done to reduce the system to the simplest model having the desired properties, the calculations are still likely to be extensive (hence expensive).

In this paper we will discuss an approach to the solution of computing system congestion which is very often an attractive alternative to simulation for a system designer or "architect." The approach is based on the use of finite-state Markov chains as models for the system, followed by a numerical solution of a set of algebraic equations for the equilibrium probabilities of those Markov chains.† It will be shown that, through the use of an efficient program for the accurate solution of large problems of the above type, much less computation time is needed than would be needed to simulate the same system.

This technique also shows promise of providing a procedure which is well suited to use on-line in a man-machine interactive mode. In such a mode, a

*This work was sponsored by the Rome Air Development Center, Rome, N.Y., under Contract No. AF 30(602)-3558.

†Following Churchman,[1] we note that since our procedure does not involve "sampling" of the model, it is not a simulation technique at all.

user of the procedure should be able to explore systems freely, and without excessive delays, while solutions to the problems he poses are prepared for him. He should also receive precise, reproducible answers which can be rationally compared with the results of other analyses and other systems. The procedure described here comes close to these goals in many respects.

MARKOV CHAIN MODELS

It should first be pointed out that the mathematical models known as Markov chains represent a quite broad and useful class of stochastic models for computer systems. Indeed, most models usually represented by networks of queues and processes can be approximated quite closely by some model derived from a Markov chain. However, the usual problem encountered in the use of these models is one of size; the number of states in the chain representing the system can easily exceed all reasonable bounds. By going to numerical methods, rather than the analytical methods typical of queueing theory, it is possible to deal with much larger Markov chains and hence to make substantial use of their generality.

Now, in the most common cases we regard the system being modeled as an interconnection of *queues* and *processes*,‡ with a prescribed stochastic flow of tasks among them. In this context, any processing capability which can be occupied by at most one task at a time can be considered a *process*. Thus an arithmetic processor, a segment of memory, a data channel, a stored program, a console, or even an operator or user can be considered processes. Also, in this context, a *queue* is any list or collection of uncompleted tasks whose routes are stochastically indistinguishable. There will usually be an *integer variable* associated with each queue or group of processes: e.g., the number of tasks waiting in that queue, or the number of processes in that group currently occupied by tasks. Also associated with each process will be a random time variable representing the duration of time that a task will occupy the process. At the end of that interval the values of some of the integer variables will change, due to the motion of the task from the process to its next process. Many things can happen to this flow: processes may be blocked by other processes, preemptions may occur, priorities may be assigned. Whatever happens it should be represented in the model. However, in order to be solved it must be described in terms of a mathematical model which is capable of solution.

As a practical matter, the most useful model for the purpose is the Markov chain, which will be described in the following sentences. For the sake of generality, and in order not to lose the ability to represent blocking, preemptions, priorities, and other complications, this discussion is relatively abstract and general.

The process of creating a Markovian model whose characteristics approximate a given computer system having any complexity in its rules of behavior is, broadly speaking, a part of queueing theory, and a thorough presentation of that process will not be attempted here. It suits our purposes merely to indicate the nature of the Markovian restrictions, and so to give assurance that Markov chains can be often applied as models. A paper by Smith,[2] appearing in another session of this conference, presents a discussion of several such models which represent aspects of a time-shared computer system, along with conclusions derived from RQA-1 analysis. That paper will serve to illustrate the next remarks more concretely. Two previous applications of Markovian models and of RQA-1 have also been discussed elsewhere,[3,4] and serve as good illustrations.

Consider the state of a system to be described by an n-component vector $\underline{x} = \{x_1, x_2, \ldots, x_n\}$. Let the components x_i be integer-valued and $0 \leq x_i \leq N_i$, where N_i is a known finite integer, for each $i = 1, 2, \ldots, n$. Let the value of the state $\underline{x}$ at any particular time t be represented by a random variable $\underline{x}_t$ so that $\{\underline{x}_t,\ 0 \leq t < \infty\}$ represents a continuous-time stochastic process. Since the value of $\underline{x}_t$ will vary with time by distinct jumps, the time intervals between successive jumps can be designated by a sequence of random variables $\{\tau_1, \tau_2, \ldots\}$.

Under certain conditions, the process $\underline{x}_t$ can be represented by a Markov chain. Let

$$t_i = \sum_{k=1}^{i} \tau_k, \qquad i = 1, 2, \ldots \tag{1}$$

and

$$t_0 = \theta^+ \tag{2}$$

so that the t_i, $i = 1, 2, \ldots$, represent time values immediately after occurrence of a jump. Then let the following be true:

(1) For every pair of states $(\underline{l}, \underline{m})$ representing a jump from $\underline{l}$ to $\underline{m}$ which is possi-

‡In queueing theory, the term server would be more common.

ble (probability $\neq 0$), the sequence $\{\tau_1, \tau_2, \ldots\}$ is a family of conditionally independent[5] random variables: given $\underline{x}_{t_{i-1}} = \underline{l}, \underline{x}_{t_i} = \underline{m}$.

(2) For each i and each fixed pair of states $(\underline{l}, \underline{m})$ in the above set

$$pr\{\tau_i \leq \tau' \mid \underline{x}_{t_{i-1}} = \underline{l}, \underline{x}_{t_i} = \underline{m}\} = 1 - \epsilon^{-\nu_{\underline{l},\underline{m}}\tau'} \quad (3)$$

where $\nu_{\underline{l},\underline{m}}$ is a positive constant.

Under these circumstances $\underline{x}_t$ is a continuous-time finite-state Markov chain.

If we interpret the state variables as representing values of queue lengths, numbers of occupied processes of a particular type, etc., and if we interpret the times t_i as the times just after arrivals occur or processes complete, then what this implies is that the intervals of time between arrivals, and the intervals of time during which a task occupies a process, must be

1. Independent of all other inter-arrival or occupancy times, and
2. Exponentially distributed random variables when the state at the beginning of the interval, and the consequences of ending the interval, are known.

However, since state variables may represent any integer variable related to the system modeled, the latter interpretation is sometimes unnecessarily restrictive. It should especially be noted that frequently a model which is not Markovian in a particular defined state space *will* be Markovian if several additional variables are added. One usually attempts to choose as state variables x_t the smallest set of variables for which $\underline{x}_t$ is Markovian.

The requirement that all processing intervals must be exponentially distributed, in the sense of Eq. (3), is often too severe a restriction. Fortunately, there are several recourses available. First, there is a considerable range of derived distributions which can be constructed by appropriate interconnection of "exponential" processes.[6] In other words, a "non-exponential" process can be replaced by several artifical exponential processes. The cost of this artifice is the addition of more state variables, and thus an increase in the complexity of the model. Figure 1 (adapted from a figure of Morse[6]) shows several of these derived distributions in order to illustrate some of the variety of distributions available by this means. If we represent, schematically, an exponential process by the symbol of Fig. 2, then the

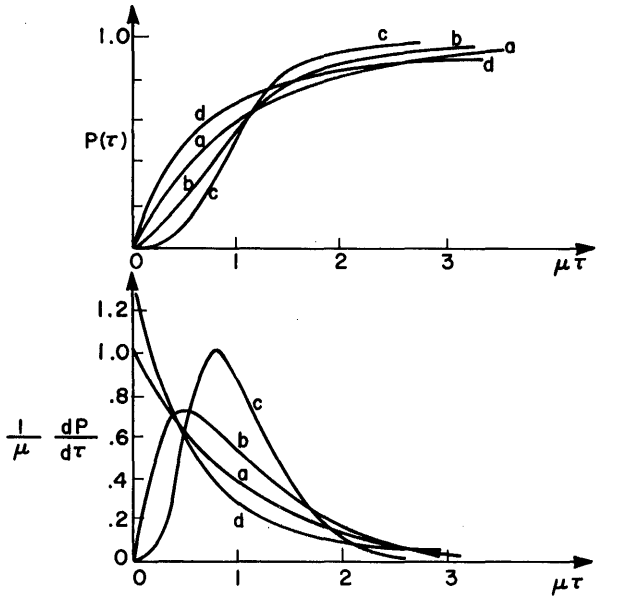


Figure 1. Distribution functions and density functions derived from the exponential. The mean of each distribution is $1/\mu$. (a) Exponential, parameter μ. (b) Special second order Erlang, parameter μ. (c) Special fifth order Erlang, parameter μ. (d) Hyperexponential, second order $\mu_1 = \mu/2$, $\mu_2 = 2\mu$, $\alpha_1 = \alpha_2 = 1/2$.

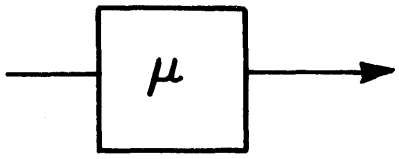


Figure 2. Schematic representation of an exponential process with mean service time $1/\mu$.

Erlang and Hyperexponential processes can be represented by Figs. 3 and 4 respectively.

Secondly, if only one of the processors does not have an exponentially distributed processing interval, the entire system can be transformed into a related discrete-time Markov chain by a process known as imbedding.[7] Once the discrete-time imbedded chain has been solved, one proceeds (usually in a straightforward manner) to find the related solution of the original process. One important class of processes for which this approach

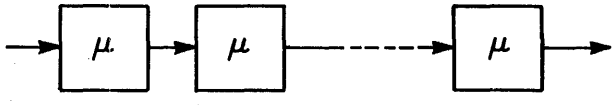


Figure 3. Schematic diagram modeling an nth order Erlang process.

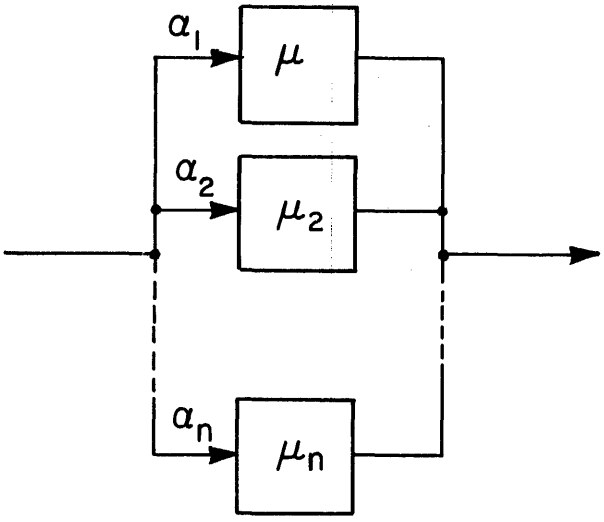


Figure 4. Schematic diagram of an nth order hyper-exponential process.

is highly useful is the class of semi-Markov processes.[8]

This survey of modeling techniques was necessarily brief, and intended merely to indicate that a great deal of flexibility in modeling is available if one can treat either continuous-time or discrete-time Markov chains which have a large number of states. Again, for examples of the power of the models Refs. 2, 3, and 4 are recommended. They are, in fact, relatively simple models; many more complex models can and have been treated effectively.

EQUILIBRIUM JOINT PROBABILITY DISTRIBUTIONS

The prime objective of most analyses of queueing systems is the evaluation of the equilibrium[9] probabilities of state. Since the state is described by the vector $\{x_1, x_2, \ldots, x_n\}$, the probability distribution of the state will be a multivariate distribution, and the state probabilities will be joint probabilities for the variables $x_1, x_2, \ldots, x_n$. Once these joint probabilities are known many other probabilistic measures are readily established as marginal distributions, expectations, or simple functions of these. Through-put rates, processor utilization efficiency, expected waiting times, distributions of queue lengths, distributions of the number of processes occupied, and probability of "busy signal" are but a few of these which are readily computed from the equilibrium probabilities of state.

In order to simplify what follows, we will refer to the states without regard to their vector character. In other words, we will refer to a state as a positive integer i which is the result of a function $i = i(\underline{x})$ which assigns an integer value to each distinct vector state which occurs with a nonzero probability. (Since each dimension of every $\underline{x}$ is finite, the set of values of i will also be finite). The vector interpretation is readily recovered after the calculation of the "probability of state i" is completed. Appropriate marginal distributions, expectations, etc., can thus still be computed. By this device, we can treat the state as a finite integer, and the vector-valued Markov chain $\{\underline{x}_t: 0 \leq t < \infty\}$ by a one-dimensional continuous-time finite-state Markov chain $\{i_t: 0 \leq t < \infty\}$. Correspondingly, if it is a discrete-time vector-valued Markov chain $\{\underline{x}_k: k = 0, 1, \ldots\}$ which must be solved, then it can be treated as a one-dimensional discrete-time Markov chain.

THE RECURSIVE QUEUE ANALYZER

The Recursive Queue Analyzer,[10] RQA-1, is a computer program designed to evaluate the equilibrium joint probability distributions of the state variables in very large, finite Markovian queueing systems. It has been designed to facilitate the analysis of both discrete- and continuous-time Markov chains having as many as 5000 states. The primary design goal has been to provide a computation fast enough to encourage experimentation with models in the study of system design. This has been achieved through efficient use of available (32K) high-speed storage in the computer (an IBM 7090), and through careful program design. The program was written in the MAD language, with selective use of the UMAP assembly language.

For a continuous-time Markov chain with a finite state space, one can always write an equation for the equilibrium probabilities in the form

$$\underline{\pi} Q = 0 \tag{4}$$

where $\underline{\pi}$ is a vector whose ith element is the equilibrium probability that the system is in state i, and Q is a matrix of constants called the transition intensity matrix of the chain. Q is descriptive of the system model.

For a discrete-time Markov chain with a finite state space, one can equivalently write an equation for the equilibrium probabilities in the form

$$\underline{\pi} A = \underline{\pi} \tag{5}$$

where $\underline{\pi}$ has the same significance, and A is a matrix of constants called the transition matrix of the chain.

The RQA-1 employs an iterative procedure to determine the solution $\underline{\pi}$ to Eqs. (5) and (6). The procedure is a straightforward power-iteration procedure,[11] so that if $\underline{\pi}_k$ is the kth iterate,

$$\underline{\pi}_{k+1} = \underline{\pi}_k G \tag{6}$$

is the $(k+1)$th iterate. The matrix G may be either equal to the matrix $\Delta Q + I$, (Δ a scalar), or the matrix $\xi A + (1-\xi)I$, (ξ a scalar), depending on whether Eq. (4) or (5) is to be solved. The Δ and ξ are chosen so as to guarantee efficient convergence to a solution of (4) or (5).

Clearly, a 5000 degree matrix when stored as a two-index array requires 25,000,000 locations of storage, which is unreasonable for a "fast" program. However, both A and Q are generally sparse matrices (have mostly zero-valued elements) and will usually have a high degree of repetition of equal element values. Hence a scheme of storage which lists location information along with value information is a necessary starting point.* The repetitiveness is partially a result of a "block structure" imposed on the matrices A or Q by a choice of a well-behaved mapping function $i(\underline{x})$, and partially a result of the fact that the probabilities of transition from a state $\underline{l}$ to a state $\underline{m}$ are often constant functions of one or more of the coordinates of $\underline{l}$, at least over some range of values. Both of these effects are often imperfect, but still useful. (If they were perfect, they would have to have been the result of a process having independent projections, and the matrices A or Q would be Kronecker sums of the matrices of the projection processes.)

In the program a set of four vectors, together called a transition table, are constructed which implicitly define the matrix. Let us call them α, β, γ, and B and denote their ith elements by α_i, β_i, γ_i, and B_i respectively. The quadruple $(\alpha_i, \beta_i, \gamma_i, B_i)$ specifies one or more elements of a matrix in the following manner:

The value of the element is α_i and its matrix coordinates are (β_i, γ_i). Due to the repetition usually found in the matrices, the value α_i may occur in other locations of the matrix with coordinates $(\beta_i + r\delta,\ \gamma_i + r\delta)$, where δ is a constant (fixed throughout the transition table) and r takes values $0, 1, 2, \ldots, (B_i - \beta_i)/\delta$. In other words, the quadruple $(\alpha_i, \beta_i, \gamma_i, B_i)$ specifies the occurrence in the matrix of elements with value α_i at coordinates (β_i, γ_i), $(\beta_i + \delta,\ \gamma_i + \delta), \ldots, (B_i, \gamma_i + (B_i - \beta_i))$. Thus, the quadruple $(\alpha_i, \beta_i, \gamma_i, B_i)$ might represent the matrix

$$\begin{array}{ll} & \begin{array}{cccc} \gamma_i & \gamma_i+\delta & \gamma_i+B_i-\beta_i & \\ \downarrow & \downarrow & \downarrow & \end{array} \\ \begin{array}{l} \\ \beta_i \to \\ \\ \beta_i+\delta \to \\ \\ B_i \to \end{array} & \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ \begin{smallmatrix} 0&0&0&0\\0&0&0&0\\0&\alpha_i&0&0\\0&0&0&0 \end{smallmatrix} & 0 & 0 & 0 \\ 0 & \begin{smallmatrix} 0&0&0&0\\0&0&0&0\\0&\alpha_i&0&0\\0&0&0&0 \end{smallmatrix} & 0 & 0 \\ 0 & 0 & \begin{smallmatrix} 0&0&0&0\\0&0&0&0\\0&\alpha_i&0&0\\0&0&0&0 \end{smallmatrix} & 0 \end{array}\right] \end{array} \tag{7}$$

and the matrices represented by the other quadruples can be considered to be *added* to it to form A, or Q. If no repetition of the value α_i occurs, the value of B_i will be equal to the value of β_i.

For this matrix storage scheme, it is possible to carry out a vector-matrix product very efficiently. Let π_k^i denote the ith component of the kth iterate. Then each iteration (Eq. (6)) is carried out in the following sequence:

1. The vector to contain $\underline{\pi}_{k+1}$ is initially zeroed.
2. Set $i = 1$, $j = 0$.
3. Multiply α_i by $\pi_k^{\beta_i + j\delta}$ and accumulate into $\pi_{k+1}^{\gamma_i + j\delta}$.
4. Repeat step (3) for $j = 1, 2, \ldots,$ until $(\beta_i + j\delta)$ is greater than B_i.
5. Reset $j = 0$, and repeat steps (3) and (4) for $i = 2, 3, \ldots,$ until all quadruples have been treated.

SPEED OF SOLUTION—A COMPARISON

For the simple power-iteration used in the RQA-1 program, the number of multiplications required per iteration is equal to the number of nonzero elements in the matrix (A or Q). This is exactly

*The storage scheme below, and the procedure for carrying out the iteration when using this storage scheme, were suggested to the authors by Prof. R. V. Evans, of the Case Institute of Technology (private correspondence).

equal to the number of distinct pairs of states $(\underline{l}, \underline{m})$ which represent possible starting and ending values, respectively, of a jump in the stochastic process $\{\underline{x}_t\}$ (or $\{\underline{x}_k\}$). Thus, if we average over all $\underline{l}$ the number of distinctly different values that can be reached in a single step from $\underline{l}$, and represent that average by the symbol E, then the number of multiplications per iteration is the product ES of the average "activity" E, and the number of states S.

In order to compare the speed of solution of RQA-1 with that of simulation in a continuous-time case, we will estimate the relative error η in the computation of the equilibrium probability of some event. Let this probability have value π, and be the sum of any number of limiting state probabilities. Let the absolute initial error, resulting from the choice of the initial iterate, be ϵ_0. Then, it has been shown[10] that the convergence error of the RQA-1 algorithm after k iterations is usually

$$O\left[\left(1 - \frac{|\gamma|}{\nu}\right)^k \epsilon_0\right] \tag{8}$$

where $|\gamma|$ is the nonzero eigenvalue of the matrix Q having smallest modulus, ν is the rate of occurrence of jumps averaged over all states, and it is known that $\frac{|\gamma|}{\nu} < 1$. Thus, the number of iterations required to reduce the relative error to the order $O(\eta)$ is

$$I \approx \frac{|\log(\eta/\eta_0)|}{\log\left(1 - \frac{|\gamma|}{\nu}\right)} \tag{9}$$

where $\eta_0 = \epsilon_0/\pi$. Further, the number of multiplications required is of the order of

$$M = \frac{ES\,|\log(\eta/\eta_0)|}{\log\left(1 - \frac{|\gamma|}{\nu}\right)} \tag{10}$$

Of course, other operations are also required, but RQA-1 holds the iteration time to about twice the multiplication time, and all tasks other than iteration and output do not significantly increase this computation time.

These figures are consistent with experience on the IBM 7090. Generally, for 100-state problems such as that reported by Fife and Rosenberg,[3] about 30 iterations per second were obtained, with complete solutions (within 0.0001) in about two seconds. For 1000-state problems, three iterations per second is typical, with solution times on the order of 20 seconds.

In contrast, we now estimate the number of random number generations required for simulation of the same models. It will be assumed that the limiting probability π is estimated by calculating the percentage of total time that the system is found to be in the state i. Then the standard deviation s of the estimate of π can be approximated by the expression

$$s \approx \frac{2\pi(1 - \pi)}{|\gamma|\,T} \tag{11}$$

where T is the duration of the simulation (T is in the time units of the system, as is also $1/\gamma$). For the absolute convergence error, $\eta\pi$, to be within two standard deviations of the estimate a duration of simulation of

$$T \approx \frac{8(1 - \pi)}{\pi\,|\gamma|\,\eta^2} \tag{12}$$

is required. The number of random numbers generated (assuming one per jump) would need to be

$$R \approx \frac{8\nu(1 - \pi)}{\pi\,|\gamma|\,\eta^2} \tag{13}$$

It should be observed that the typical time to generate a random number is much greater than a "multiply-time," and that present simulators often take much more time for housekeeping than for actual generation of the random numbers. Thus a simple comparison of R with M is biased strongly in favor of simulation. Nevertheless, we proceed to make only a simple comparison. The ratio of R/M is approximated by

$$\frac{R}{M} \approx \frac{8(1 - \pi)}{ES\,\pi\eta^2\,|\log(\eta/\eta_0)|} \cdot \frac{\nu \log\left(1 - \frac{|\gamma|}{\nu}\right)}{|\gamma|} \tag{14}$$

Typically $\frac{|\gamma|}{\nu} << 1$, and the second factor is thus approximately unity. Hence

$$\frac{R}{M} \approx \frac{8(1 - \pi)}{ES\,\pi\eta^2\,|\log(\eta/\eta_0)|} \tag{15}$$

Since the usual applications will be ones in which π is not close to unity, we can usually also neglect the $(1 - \pi)$ factor. The remaining function,

$$\frac{8}{ES\,\pi\eta^2\,|\log(\eta/\eta_0)|} \tag{16}$$

which approximates R/M, is plotted in Fig. 5 for $\eta_0 = 1$ (a fairly conservative choice). This figure graphically shows that the ratio R/M increases very

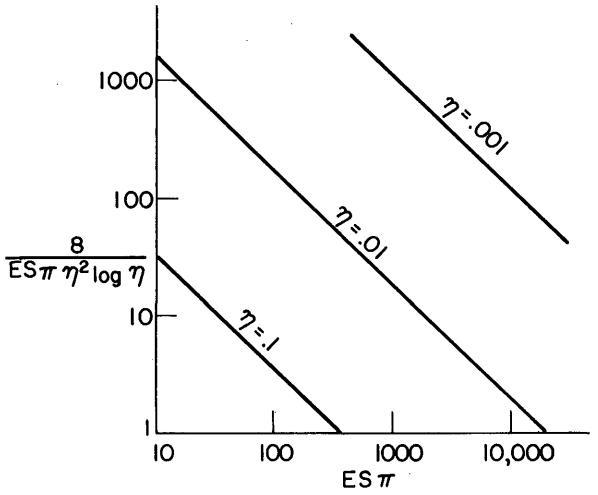


Figure 5. Illustrating the computational merit of the numerical techniques.

rapidly as more accuracy (smaller η) is required, and decreases as the number of states, the activity E, or π are increased. Thus for small enough η the number of random number generations required in the simulation can be very much larger than the number of multiplications required in the iterative process. This results from the fact that simulation error (from Eq. (11)) normally decreases as the square root of simulation time, while iteration error (from Eq. (9)) decreases exponentially with the number of iterations. Where repeatability and comparison of results are important, errors of the order of 0.001 are not at all unreasonable. In such a case, even in an extreme problem having $S = 5000$, $E = 20$, and $\pi = 0.1$, the iterative techniques will have an advantage of two orders of magnitude over simulation. Add to this the much greater housekeeping involved in simulation, and the advantage is dramatic.

A comparison for the discrete-time Markov chain solution would result in similar conclusions.

CONCLUSIONS

The purpose of the foregoing comparison was not to issue a call for everyone to abandon simulation for the analysis of computer systems. Rather, it was intended to point up a potential which should not be ignored. There are many difficulties incurred in the use of a program like the RQA-1 which need to be overcome before it will be universally applied. The process of modeling systems by Markov chains is a relatively sophisticated one, and often requires a great deal more "cleverness" than does a Monte Carlo approach using GPSS or Simscript. Secondly, the representation of the model in the form of a matrix in the RQA format is now a quite tedious process. (A coupling of an RQA-like procedure with a problem-oriented language can relieve this difficulty, and is currently under study.) Thirdly, although the Markovian models have much generality, there will always be problems which cannot be so modeled, and hence must be simulated (unless the expense is prohibitive).

On the other hand, even one order of magnitude improvement in the time required to solve a system congestion problem with precision can make a man-machine interactive exploration of system configurations by a system "architect" practical when it might otherwise have been impractical. Also, with the current provisions in RQA-1 for defining the transition matrices in literal form, so that parameters can be altered by a simple change of data at execution time, it is possible to obtain extensive sets of graphs describing functional relationships accurately and economically, as was done in Refs. 2, 3, and 4. Using these features, all of Smith's published results[2] required less than 4 minutes of IBM 7090 computation.

REFERENCES

1. C. W. Churchman, "An Analysis of the Concept of Simulation," *Symposium on Simulation Models: Methodology and Application to the Behavioral Sciences*, A. C. Hoggatt and F. E. Balderston, eds., South-Western Publishing Co., Cincinnati, 1963, pp. 1–12.

2. J. L. Smith, "An Analysis of Time-Sharing Computer Systems Using Markov Models," this volume.

3. D. W. Fife and R. S. Rosenberg, "Queueing in a Memory-Shared Computer," *Proc. of the 19th Nat. Conf. A.C.M.*, Philadelphia, 1964.

4. —— and J. L. Smith, "Transmission Capacity of Disk Storage Systems with Concurrent Arm Positioning," *IEEE Trans. on Electronic Computers*, vol. EC-14, no. 4, pp. 575–582, (Aug. 1965).

5. M. Loeve, *Probability Theory*, 3d ed., Van Nostrand, 1962, pp. 351–353.

6. P. M. Morse, *Queues Inventories and Maintenance*, Wiley, 1958, Chap. 5.

7. D. G. Kendall, "Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of Imbedded Markov Chains," *Ann. Math. Stat.*, vol. 24 (1953).

8. W. L. Smith, "Regenerative Stochastic Pro-

cesses," *Proc. Roy. Soc.* (*London*), Ser. A, vol. 232, p. 6 (1955).

9. E. Parzen, *Stochastic Processes*, Holden-Day, 1962, pp. 247–253.

10. V. L. Wallace and R. S. Rosenberg, "RQA-1, The recursive Queue Analyzer," Systems Engineering Laboratory Technical Report No. 2, University of Michigan, Ann Arbor.

11. J. Todd, *Survey of Numerical Analysis*, McGraw-Hill, 1962, pp. 196–197.

SMPS—A TOOLBOX FOR MILITARY COMMUNICATIONS STAFFS

Kathe Jacoby, Diana Fackenthal and Arno Cassel
Franklin Institute Research Laboratories
Philadelphia, Pennsylvania

INTRODUCTION

Many papers oriented to the computer user deal with programming languages. These languages may be either flexible or oriented toward a particular problem field, such as military information retrieval or simulation; however, they are languages requiring the user to learn vocabulary, grammar, punctuation, and spelling to translate his problem into the specific language. This is not easy and generally requires considerable practice.

An officer on the communications staff of a military headquarters does not have time to study a language and learn how to express himself in it. In addition, he does not have the experience of an industrial engineer who is accustomed to flow-charting the operations needed to accomplish a function. Nevertheless, he needs to evaluate the effectiveness of his present methods and procedures and level of staffing under conditions which would occur when the workload might suddenly change because of world or local military or political events. He also needs to be able to determine whether any changes in methods, procedures, or staffing will improve the total response of the system.

The prime criterion for evaluation of a communication system is message transit time. Within this criterion are subcriteria to be chosen by the headquarters involved, which may specify:

> The maximum transit time for messages of a specific class shall be less than T minutes.
>
> The percentage of messages of a specific class with transit time less than T minutes shall be greater than P percent.

Transit time through a system depends on two factors: processing time and waiting time. Processing time can be determined without the use of computers by observing the required time to perform specific tasks and by summing this time over all the tasks to be performed on a specific message. Waiting time is either batching or queuing time. Batching time is the time an operator waits after completing one task on a message before delivering it to the next task or operator, so that the first operator can continue performing the same task on a number of messages; this time can be estimated. Queuing time can be mathematically estimated when only a few queuing points are involved. However, when many dynamically interacting queues must be considered, Monte Carlo simulation techniques must be used to gather information about the formation of queues and the delays caused by queuing. This requires digital-computer simulation.

The Franklin Institute Research Laboratories (FIRL) has developed two tools for officers on the communication staff of a military headquarters to use for system evaluation; these tools were developed as part of a study for the Department of the Army and the Defense Communications Agency to improve message processing operations within a headquarters.[1] One tool is a method called Auto-

matic Flow Process Analysis (AFPA) which allows personnel without any flow-charting or system-analysis experience to develop accurate flow charts by carrying out a set of procedures.[2] The second is Simplified Message Processing Simulation (SMPS), with which the same personnel can prepare a simulation model and message samples by following a set of simply stated procedures; SMPS does not require personnel to learn any programming language.

With SMPS, members of a military communications staff can evaluate a message-processing system under dynamic conditions without requiring the services of personnel experienced in computer technology or programming. The SMPS toolbox contains building blocks and a framework with which a model of a message-processing system can be built.

COMMUNICATIONS STAFF NEEDS

The communications staff at a military headquarters needs to be continually aware of the capabilities and effectiveness of their current message-processing systems, not only with respect to current traffic but also with respect to crisis conditions which may occur. Figure 1 shows an overview of the activities within a Message Communications Terminal office (communications center and staff message control) at a military headquarters. Within the limits of military regulations and command structure, this staff is able to suggest changes to improve system operation. However, changes should not be implemented unless there is assurance that the total system operation will be improved; therefore, methods for evaluation are required. Because of the differences in needs, regulations, and traffic at different headquarters, only the staff at the individual headquarters (rather than a higher agency) can best evaluate its own systems. The likelihood that these operational staff personnel have programming background or inclination is very small.

TOOL 1, AFPA

AFPA permits the non-system analyst to construct an accurate flow chart of the operation of his system. In the message-processing case for which AFPA was designed, the message passes along the flow of the chart through the tasks performed in the boxes of the flow chart. A task is called an event and specifies what personnel and equipment are involved (such as a communications-center receive operator or a Xerox machine), what is done (such as tearing the message from the teletype monitor), and how long the event takes (such as 30 seconds).

TOOL 2, SMPS

Figure 2 shows a small fragment of an AFPA flow chart. A detailed simulation including all of

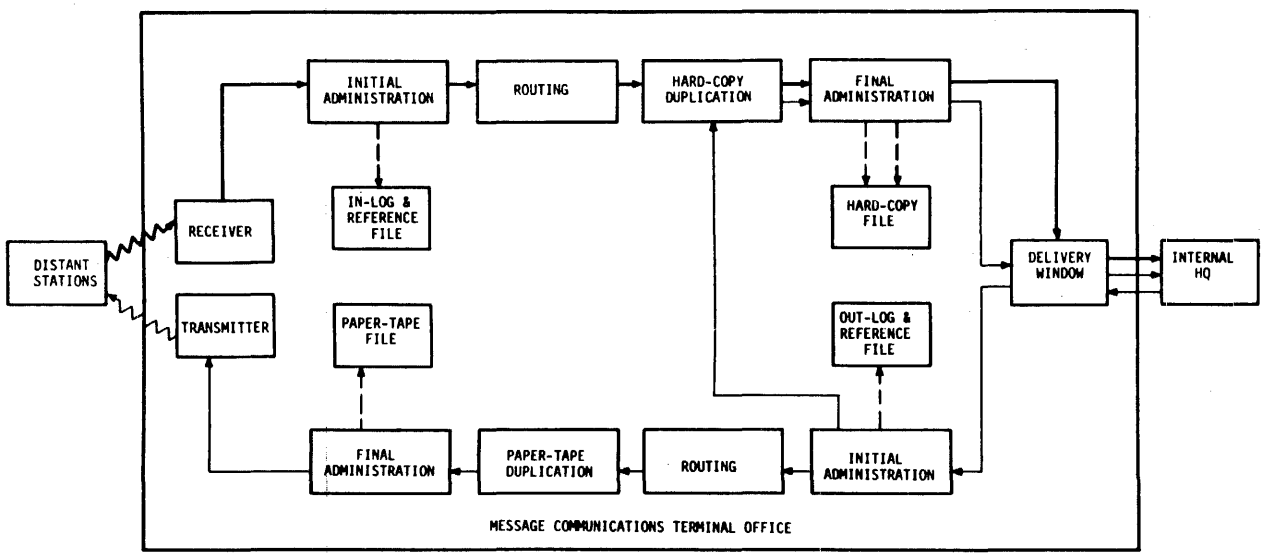


Figure 1. Overview of activities within a message communications terminal office.

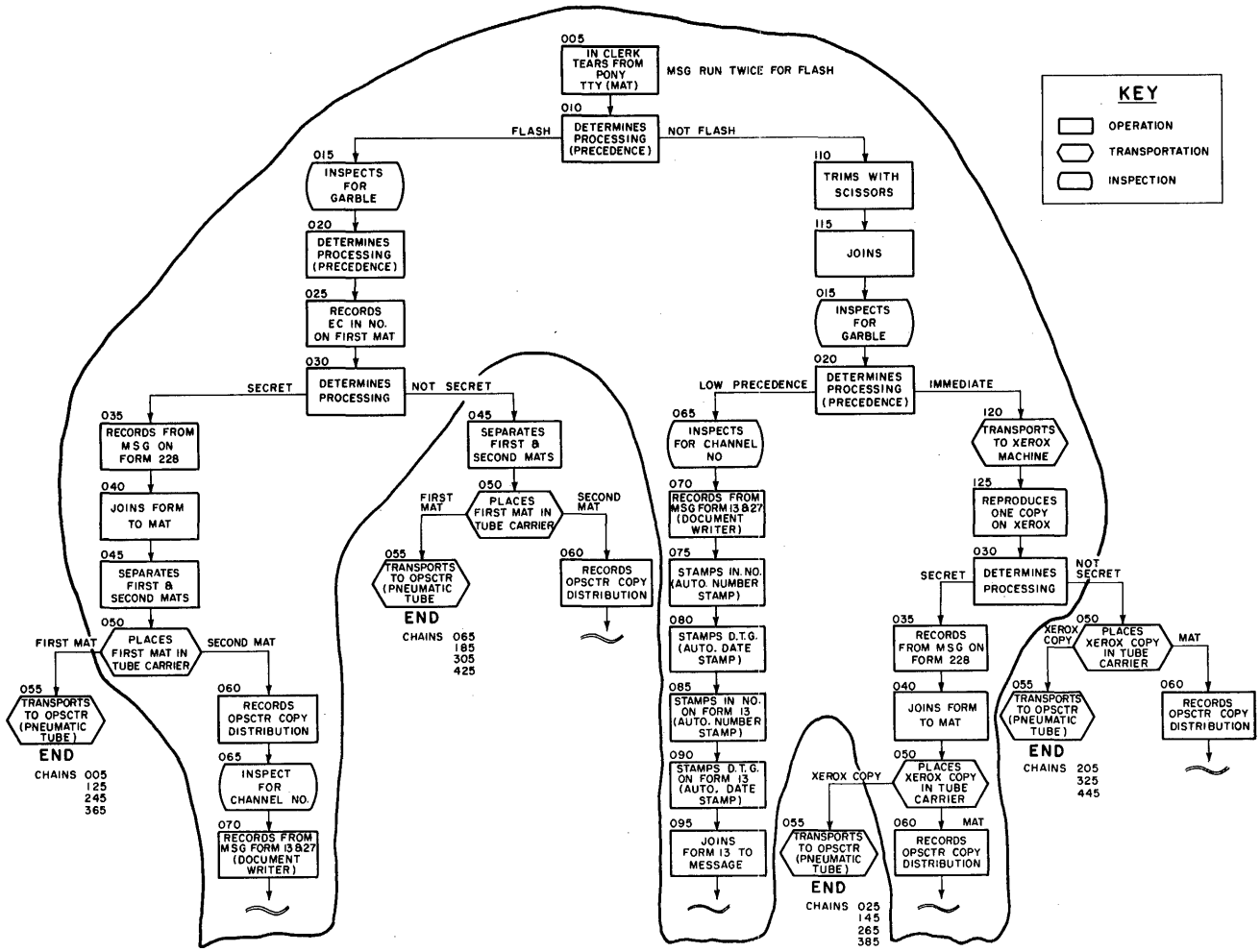


Figure 2. Fragment of an AFPA flow chart.

the events on an entire AFPA flow chart, would take many hours to construct; however, Simplified Message-Processing Simulation permits the AFPA events to be grouped into broader tasks which may be matched directly to the SMPS building blocks. Thus, a simulation model may be assembled quickly and easily. The outlined area of Fig. 3 is the SMPS simplification of the outlined portion of the AFPA flow chart fragment shown in Fig. 2.

A technical report, "SMPS—Simplified Message-Processing Simulation,"[3] instructs the user how to construct simplified flow charts from the AFPA flow charts, how to fill out task-description worksheets from the simplified flow chart, and how to match the SMPS building blocks to the tasks defined.

When the SMPS building blocks are matched to the tasks, the simulation model is essentially complete. The SMPS report also describes how the input messages for the simulation may be prepared from a real traffic sample or from statistically generated messages.

What SMPS Is

SMPS is a language derived from the macro-assembler capabilities of IBM's GPSS II. The building blocks of SMPS include a set of GPSS variables defined in terms of the parameters of a GPSS transaction, a set of functions for the generation of GPSS parameters from a deck of cards generated independently to describe a message sample, a few other GPSS system variables, and a set of GPSS macro instructions. SMPS relies heavily on the development of DMPS (Detailed Message-Processing Simulation[1]) for the method of parameter construction and assignment.

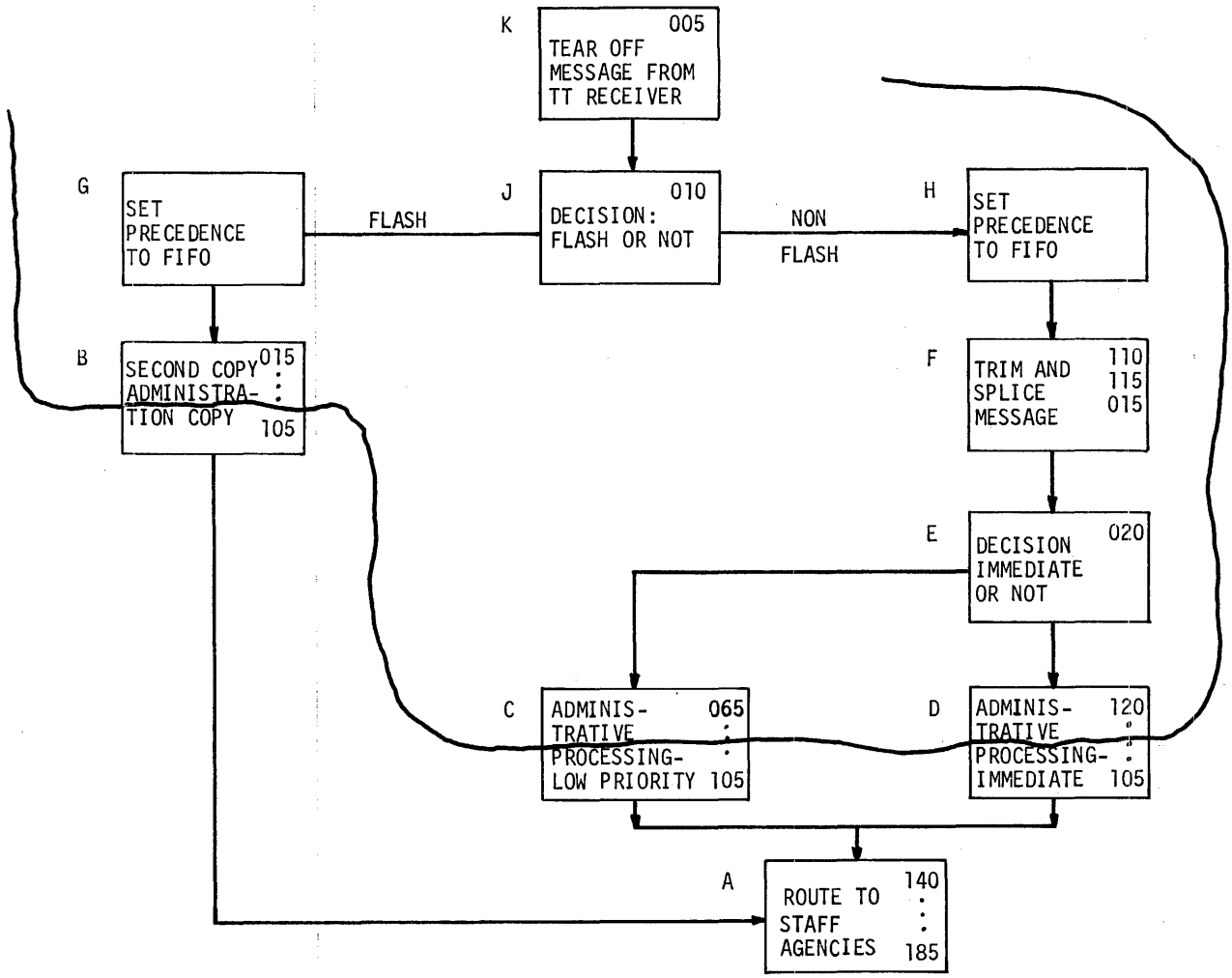


Figure 3. Simplified flow chart fragment.

Use of SMPS

The simplified flow chart is derived from the AFPA flow chart of chains of events, which is in tree form. First, uninterrupted tasks are identified, and identical chains of tasks are merged. The first worksheet describes personnel and equipment (Fig. 4). The second worksheet describes the tasks in the simplified flow charts in terms of personnel and equipment required, next tasks to be performed under what conditions, and processing time expected, either in numbers or as a formula in terms of message characteristics (Fig. 5). The next task is the selection of SMPS building blocks. In the simplest cases, a SMPS building block matches each task. However, if the task is complex or unusual, it may be necessary to divide it into several simpler tasks to find a match.

In the case of communications-terminal processing for which SMPS was designed, the usual tasks are decision-making, logging, routing, poking (tape cutting), tape reproduction, offline encryption/decryption, inspection, transmission, filing, reference lookup, transportation or delivery between major staff areas of a headquarters, typing, reproduction, collation, distribution of copies, and additional administrative functions.

The SMPS building blocks, called modules, are divided into six categories. The first category contains general-purpose modules which involve queue number, personnel or equipment identifica-

Personnel	Associated Code	Unique Code	No. Identical	Equipment	Associated Code	Unique Code	No. Identical
SMC Inclerk		2	1	Xerox		4	1
SMC Message Controller		3	1	Multilith		5	2
Multilith Operator	5	-	1				

Figure 4. Worksheet 1, personnel and equipment.

tion, next module, and time factors; the time factors are for processing or batching. The second category contains decision modules for usual decisions. The third category contains modules which represent transportation and contain facilities to record transit times within the GPSS simulation and for external statistical analysis. The fourth category contains a set of modules to permit delivery at regular intervals. The fifth group contains the modules used to control the flow of the three types of messages into the model. The first type of message is the sample concerning which transit times are to be measured to evaluate system effectiveness; this is the group which is specified on a card deck generated independently of the model. The second type concerns service and similar messages which are not being directly evaluated but which occupy both personnel and equipment within the communications facility; these are an integral part of the model. The third type of message does not represent actual traffic but is used to account for any other activities, such as breaks for personnel or downtime for equipment, which would impede the processing of significant message traffic by occupying personnel or equipment and making them unavailable. The sixth group contains flexible modules which allow most unusual tasks to be performed without requiring knowledge of GPSS.

Figure 6 contains the description of a few modules.

Headquarters: A	Staff Agency: SMC	Incoming ✓ Outgoing

Task ID Code	Task Description	Personnel and Equipment Needed	Processing Time		Next Task	Queue Discipline	Output Batched
			AFPA Event	Sec.			
K	Tear off message from TT Receiver	SMC Inclerk	005	33	J	FIFO	No
J	Decision flash or not	SMC Inclerk	010	1	GG if flash H if not flash	FIFO	No
G	Set queue discipline to Prec FIFO	-	-	0	B	None	-
B	Administrative processing flash message	SMC Inclerk	015...105	41	A	Prec FIFO	No
A	Routing of Messages SMC	Message Controller	140...185	109	Off chart	Prec FIFO	No
H	Set queue discipline to Prec FIFO	-	-	0	F	None	-
F	Trim and splice message	SMC Inclerk	110...015	30	E	Prec FIFO	No
E	Decision immediate or not	SMC Inclerk	020	1	D if immediate C if not	Prec FIFO	No
D	Administrative processing immediate message	SMC Inclerk Xerox copier	120...105	89	A	Prec FIFO	No
C	Administrative processing low precedence message	SMC Inclerk	065...105	24	A	Prec FIFO	Yes

Figure 5. Worksheet 2, task definition.

Name	No. of Variables	Module Description	Meaning of Variables							
			1	2	3	4	5	6	7	8
A1	5	General One individual or piece of equipment takes a message from a numbered queue and processes it for a time specified by two time factors which describe either a rectangular distribution of time or specifies a formula for time. The message is then passed to the next task.	Queue No.	Identity of equipment or personnel	Next Task	Time Factor 1	Time Factor 2			
B1	7	General Same as A1 except two additional time factors are available for batching delay.	Queue No.	Identity of equipment or personnel	Next Task	Time Factor 1	Time Factor 2	Time Factor 3	Time Factor 4	
B2	8	General Same as B1 except that additional equipment or personnel is required.	Queue No.	Identity of first equipment or personnel	Identity of second equipment or personnel	Next Task	Time Factor 1	Time Factor 2	Time Factor 3	Time Factor 4

Figure 6. Selection of available modules.

Headquarters: A		Staff Agency: SMC	Incoming ✓ / Outgoing	
Task ID Code	Module Identity	Variable No. of Module	Interpretation of Variable for use in this Task	Code
K	A1	1	Queue 1	1
		2	SMC inclerk	2
		3	Task J next	J
		4	Time factor 1 mean 33 sec	33
		5	Time factor 2 spread 3 sec	3
J	A17	1	Queue 2	2
		2	SMC inclerk	2
		3	Next task if flash = G	G
		4	Next task if not flash = H	H
G	F3	1	Task B next	B

Figure 7. Worksheet 3, module assignment.

Worksheet 3 (Fig. 7) aids in the matching of modules to tasks. The function-definition worksheet (Fig. 8) aids in the construction of functions to define processing times in terms of message characteristics.

The cards representing the significant sample are prepared by a computer program to a form acceptable by the simulation program; these cards contain the identification and significant characteristics of each message. The computer program (written in FORTRAN) also prints these characteristics of each message in English (Fig. 9). This printout includes time of arrival in the system as day, hour, and minute, as well as the simulator clock time for arrival in total seconds. It also includes the identification number, which indicates whether the message is incoming or outgoing (those numbered over 20,000 are incoming), precedence, classification, number of addresses, number of lines of text, number of communications channels required, number of pages, number of staff agencies on local distribution, number of local copies, whether off-line encryption or decryption is required, special security categories, or other special characteristics involved.

Two programs are available to prepare a message deck.

One program uses an actual message sample, in which case the message characteristics are determined by examining a message. These characteristics then are transcribed onto cards, which are used as data by this input-preparation program.

If statistical generation of messages is desired, an

Function Application	GPSS Function Identity (Time Factor 2)	GPSS Variable for Function	Variable Definition	Variable Definition in GPSS Language	Multiplier (Time Factor 1)
Pony circuit Time	FN10	V6	Number of lines	Primary variable	7 or 14
MCPU time	FN11	V20	135 + 60 (number of channels) + 140 x (number of pages)	K135 + K60 * V7 + K140 * V8	1
Multilith time	FN12	V21	(Number of copies)/2 x (number of pages) + 150 x (number of pages) + 10	V11/K2 * V8 + K150 * V8 + K10	1
Poking time	FN13	V22	67 + 4.3 x (number of addressees) + 9 x (number of lines)	K67 + K43 * V5/K10 + K9 * V6	1

Figure 8. Function-definition worksheet.

DAY	HR	MIN	TIME	IN/OUT	PREC.	CLASS.	----NUMBER OF ADD.	LINES	CHAN.	PAGE	AGEN.	COPY	CRYP	SPECAT	SPEC.ORIG/ADD
V12			V1	V2	V3	V4	V5	V6	V7	V8	V10	V11	V9	V13	V14
0	0	9	540	20001	IMMED.	CONFID	1	4	1	1	3	5	NO	0	0
0	0	18	1080	20002	IMMED.	CONFID	1	10	1	1	3	5	NO	0	0
0	0	20	1200	20003	IMMED.	CONFID	1	2	1	1	3	5	NO	0	0
0	1	18	4680	1	IMMED.	CONFID	4	29	4	2	10	31	NO	0	0
0	3	39	13140	20004	PRIOR.	CONFID	9	30	4	2	4	10	NO	0	0
0	8	41	31260	2	ROUT.	CONFID	1	2	1	1	6	18	NO	0	0
0	9	48	35280	3	ROUT.	SECRET	2	30	1	1	4	5	NO	0	0
0	9	49	35340	4	ROUT.	SECRET	3	5	1	1	4	5	NO	0	0
0	9	50	35400	5	ROUT.	CONFID	2	6	2	1	3	11	NO	0	0
0	9	58	35880	20005	ROUT.	EFTO	2	12	1	1	2	4	NO	0	0
0	10	4	36240	6	ROUT.	CONFID	2	6	1	1	3	10	NO	0	0
0	10	27	37620	20006	IMMED.	UNCLAS	11	22	7	1	1	1	NO	0	0
0	10	29	37740	20007	ROUT.	TS	3	40	1	2	2	8	YES	0	0
0	10	30	37800	20008	ROUT.	SECRET	3	14	3	1	3	9	NO	0	0
0	10	52	39120	7	ROUT.	CONFID	3	20	1	1	3	5	NO	0	0
0	10	53	39180	8	ROUT.	CONFID	1	2	1	1	3	4	NO	0	0
0	10	54	39240	9	ROUT.	CONFID	4	48	2	3	11	39	NO	0	0
0	10	55	39300	10	ROUT.	CONFID	5	27	1	2	4	11	NO	0	0
0	11	53	42780	20009	IMMED.	CONFID	2	7	2	1	1	1	NO	0	0
0	11	53	42780	20010	PRIOR.	EFTO	5	8	1	1	4	12	NO	0	0
0	11	54	42840	20011	ROUT.	EFTO	5	9	3	1	1	5	NO	0	0
0	11	56	42960	20012	PRIOR.	UNCLAS	2	32	1	2	4	13	NO	0	0
0	11	56	42960	20013	ROUT.	UNCLAS	17	11	8	1	2	7	NO	0	0
0	11	56	42960	20014	ROUT.	UNCLAS	1	12	1	1	2	8	NO	0	0
0	12	6	43560	20015	ROUT.	CONFID	3	13	1	1	1	4	NO	0	0
0	12	22	44520	20016	ROUT.	CONFID	12	14	7	1	3	3	NO	0	0
0	12	22	44520	20017	ROUT.	CONFID	3	15	1	1	1	5	NO	0	0
0	12	23	44580	20018	PRIOR.	CONFID	2	38	1	2	4	5	NO	0	0
0	12	45	45900	11	ROUT.	CONFID	1	17	1	1	3	17	NO	0	0
0	12	46	45960	12	ROUT.	CONFID	1	1	1	1	2	8	NO	0	0
0	12	50	46200	20019	PRIOR.	SECRET	16	50	4	3	5	9	NO	1	3

END OF LISTING OF INPUT CARDS

Figure 9. Listing of input messages provided by input programs.

alternative FORTRAN input-preparation program is available with which the message characteristics necessary for the run can be easily specified. A listing of these specifications and detailed diagnostic routines concerning card or logical errors is provided. The other outputs of this program are the same as those of the first input program.

Relatively few items in the printout of the simulation run are significant to this type of model. Hence, the volume of printout to examine is not excessive.

The most important question in evaluating a model of a message-processing system is, "How long does it take a message to get through the system?" This information is most meaningful in terms of the cumulative distribution function of the total transit time through the system; however, it may also be important to know the time through major subsystems, as well as the time for messages with special characteristics.

A major output of SMPS is a deck of cards, each of which contains all the characteristics of a message, a transit time either through the entire system or through a major portion of the system, and an identifier specifying the meaning of the transit time given. Thus, this card deck can be processed manually, by EAM equipment, or by computer to select the messages with the characteristics of interest and to determine the transit-time distributions for these characteristics.

The other output of SMPS is the printout produced by the GPSS program. The portions of the output significant to the user include the tables which give the fraction of total number of messages with transit times less than each increment of an accumulating time scale, and the queue statistics which indicate where bottlenecks occur.

RANGE OF APPLICABILITY OF THE PRESENT PACKAGE

Although this application is based on AFPA flow charts, the technique does not require that AFPA be used. The flow chart which describes system operation may be constructed independently; however, in this case, more skill may be required in defining the tasks of suitable size. The basic concept is that a task must be small enough that the personnel and equipment involved would not be interrupted to perform any service for any other message.

The basic structure of SMPS assumes that a message has certain properties which are recorded in the simulation representation of the message—namely,

the GPSS transaction parameters. Two properties are not fixed and may be defined at each headquarters; however, these properties may have, at most, 10 values. The characteristics chosen are ones most meaningful to a variety of military headquarters. Hence, although the processing examples carried out thus far involved military terminal processing, SMPS should be useful for any processing of military or nonmilitary messages. Although such properties as off-line encryption or security classification are not apt to be meaningful for nonmilitary applications, any properties defined can be ignored in a model. If the statistical input program is used, each specified characteristic must be examined to determine whether it can be ignored in creating the message sample.

In its current form, SMPS can be used to simulate any message-processing application where the transit time for a message and its flow through the processing steps depend only on the message characteristics defined in SMPS and on statistical variables.

APPLICABILITY OF TECHNIQUE FOR OTHER USES

The SMPS technique is not limited to dynamic analysis of message processing. Whenever a system can be looked on as consisting of processing units which can be described by a small number of characteristics and where both processing time and batching time depend on characteristics of these units alone, a set of building blocks and a structure similar to SMPS can easily be constructed in a very short time by personnel with programming experience.

Because most flow charts contain relatively few patterns of boxes and lines, it is possible to describe most systems by reusing a few modules with different variable values. For example, one general equipment- or personnel-use module can be used which includes as variables a queue number, three or four equipment/personnel identities, the next task, and several time factors. Two decision modules corresponding to two- and three-path branchings, will probably be sufficient. Decision modules have variables of relations (less than, equal to, greater than, for example), a number being tested by the relation, next task if true, and next task if false. A few special modules can be programmed to insert in the flow-process chart for priority assignments, tabulations, origination rates, and the like. With these types of modules, a model can easily be constructed.

ADVANTAGES OF THE SIMPLIFIED MESSAGE-PROCESSING SIMULATION

A "language" such as SMPS is easier to learn than a simulation or programming language: it has no grammar and little vocabulary. A model in SMPS can be constructed very quickly. Changes are readily made and alternatives are easily compared. Because the level of abstraction is high, the model is easily understood in terms of activities which occur and of what is required for the activities.

REFERENCES

1. A. Cassel et al, "Improved Message Processing (IMP)—An Analysis of Headquarters Message-Processing Operations," Technical Report 1-055, Franklin Institute Research Laboratories (Oct. 1965).

2. P. W. Maraist and A. Barskis, "Automated Flow Process Analysis (AFPA)—A Technique for Analysis of Headquarters Message Processing," ibid, no. 1-160 (Nov. 1965).

3. K. Jacoby and D. Fackenthal, "Simplified Message-Processing Simulation (SMPS)—A Technique for Analysis of Headquarters Message Processing," ibid, no. 1-161 (Nov. 1965).

4. General Purpose Systems Simulator II, Form B20-6346-1, International Business Machines Corporation (1963).

DIGITAL SIMULATION OF LARGE-SCALE SYSTEMS

Robert V. Jacobson
Advanced Systems Department, Space and Information Systems Division
Raytheon Company, Sudbury, Massachusetts

Over the past decade systems analysis teams have repeatedly demonstrated the feasibility of using general purpose digital computers to simulate the operation of large-scale real-world systems. BAG, DECAP, INCA, STAGE, TEFORM and TEMPER are all representative examples. However, the process of developing and using these system simulations has not always been entirely satisfying to the ultimate users. The purpose of this paper is to examine the process of simulating systems, and so to suggest some causes of dissatisfaction and their remedies. Because of the diversity of usage, it seems to be important to define the key words to be used.

System Model is used to mean the interrelationships and logic which describe the system adequately for the task at hand.

A **Simulation** is a mechanism based on a model which operates "like" the system. That is to say, a simulation is an operating version of the model.

A **Digital Simulation** is a simulation in which the system quantities are represented by digits, and so is most easily implemented on a digital computer.

This usage follows M. R. Lackner's paper "Digital Simulation and System Theory."[1] Note that the term computer simulation under these definitions would mean a simulation *of* a computer. Other writers[2] have used "computer simulation" in the same sense that "digital simulation" is used here, but this author believes that clarity suffers as a result.

The task flow diagram (Fig. 1) shows how a typical case moves from the problem statement to the final analysis. The figure shows five different

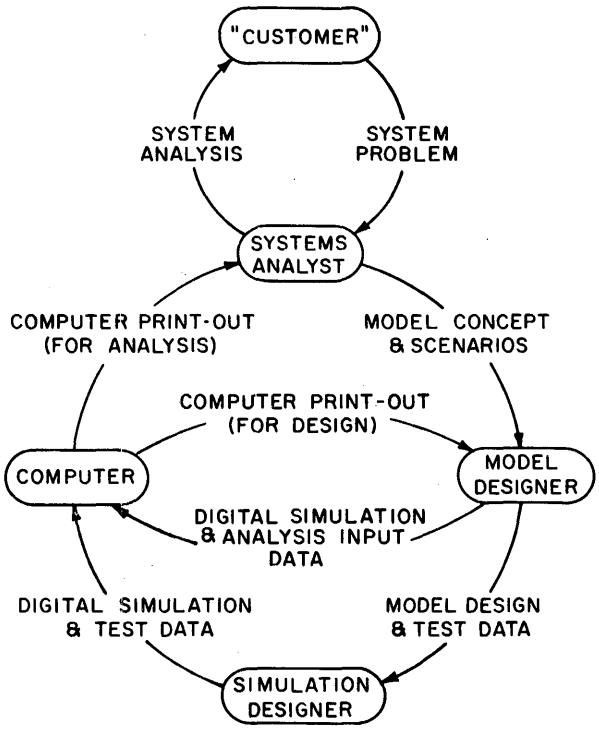


Figure 1. Task flow diagram of model/simulation construction and use.

people (or organizations) in the "loop;" while in fact a smaller number of individual people may be involved, the division of tasks into these five categories seems to be a useful one.

The flow begins with the "customer" who states the problem to be analyzed to the systems analyst. (The customer is defined as the person or organization with a principal mission other than the processes to be described below.) Let us assume that the systems analyst determines that the problem can best be solved through the use of a digital simulation and that he directs the model designer to design a system model. In many cases the systems analyst and the designer of the system model will be the same person or organization. The simulation designer, bearing in mind the scenarios which are to be analyzed, designs a simulation which is based on the model. The system analyst uses the output data generated by the digital simulation to draw conclusions about the operation of the real-world system, and reports them to the customer. Once a working computer program for the digital simulation has been constructed, the system analyst/model designer hopefully can deal directly with the computer with the help of the computer supporting staff.

With the above definitions in mind let us consider first the information flow from the system analyst to the model designer. The model designer would like to identify and define the variables and constants which the system analyst considers significant, and to find expressions which adequately describe their interrelationships. Next he wants to develop a logic diagram which represents the flow (often with time) of events or processes in the system, and which ties together the interrelationships. Two problems often arise here. The system analyst would like to have "every" real-world variable included in the model, and would like relationships to be "completely accurate."* In fact, the systems analyst probably doesn't know quantitatively the contribution made to the accuracy of the model by each of the parameters and variables which he can identify, or a complete representation of their interrelationships. It is the author's view that the model designer can best serve the system analyst by urging the initial selection of variables and relationships which will most simply (rather than accurately) describe the real-world system, with the long-range objective that, as understanding of the operation of the model grows, complexity can be introduced. As an example of initial simplicity if one were to simulate detection of a target by a radar, one might compute the effect of each of the dozens of quantities which enter into the detection of a target. On the other hand, one might *begin* with a model which says that for a given radar, detection *never* occurs beyond a given range, but at lesser ranges detection *always* occurs for all targets. Oversimplified? Perhaps, but indeed because received signal varies roughly as the fourth root of target range the approximation would not be significantly in error if, for example, one were modeling an air defense information processing system, and were not directly concerned with the sensors, in this case the radar, but rather with such things as data storage and correlation, data link saturation, and displays. The fact that a large aircraft was "detected" at 75 miles instead of 80 miles as it would be in the real-world may contribute far less to the inaccuracies of the model than other assumptions which had been made by default so to speak, rather than explicitly. That is to say that the model designer may overlook factors having a far greater effect on accuracy.

Since the very purpose of constructing the model and simulation is system analysis, it is implicit that the contribution to similitude, or accuracy of each of the model's elements is not known quantitatively at the beginning of the design effort. If the model designer can restrain the systems analyst's desire to model "everything," and rather model as little as possible initially, the result will be that the complete structure of the model will emerge at the earliest possible date and then can be quickly converted into a digital simulation. The systems analyst, and the model designer are now in a position to test the significance of each of the elements of their model in a systematic way. Continuing the example given above, they might like to make detection range a yes–no function of the target range, modified by a linear function of the nominal target cross section. If this elaboration yields a significantly different result, they might go a step further and try a model in which target cross section was a function of both nominal target cross section and target attitude relative to the radar.

The important point is that as early as possible in the design cycle they have a model (and a computer simulation of it) which operates and generates output. They have been forced to think through the entire system, as a result have gained a better under-

*The words "large-scale" are included in the title to exclude from this discussion systems which can in fact be completely simulated such as savings bank records or an airlines reservation system.

standing of the system operation, and so have more accurately identified the really significant system parameters and variables. The level of detail to which the model will then be expended is far more likely to be uniform, and both have the assurance that something will come from their efforts, however much it may fall short of their original aspirations. Equally important, the systems analyst will have a clear image of the model's structure and so will be better able to evaluate its output. These points may be summarized as:

The Model Designer's

Role: To convert the significant elements of the real-world system into a unified mathematical/logical model.

Objective: To maximize the utility of the mathematical model.

Guidelines: Evolutionary model design to achieve a uniform level of detail, and systematic evaluation of the model design.

The problems which the simulation designer must solve revolve around the conflict between the generality of the model and the explicit character of computers. The model's logic and expressions must be stated in an explicit way, input data must be of a stated form and content, and the format of the output data must be described in advance. On the other hand, the model designer can be expected to want to make changes during the design process, and each change will cost time and money. The simulation designer will be of greatest service to the model designer and systems analyst if he accepts this fact of life, and keeps in mind the thoughts suggested below.

The Simulation Designer's

Role: To convert the mathematical model into a useful computer program.

Objectives: To maximize machine independence and to simplify the process of changing the model/simulation.

Guidelines: To serve the problem at hand, not the computer.

The simulation designer should first take steps to minimize the impact of the particular computer to be used on the problem to be solved. That is to say the computer should serve rather than dominate the problem. Secondly through forward-looking design techniques, the simulation designer can often facilitate the changes which will inevitably be sought by the model designer after he has experimented with the first primative versions of the simulation. The resulting computer program should have the following characteristics:

- Inherent adaptability.
- Complete labeling of output.
- Careful source program record keeping.
- User-oriented input and output, and operating documents.
- Graphical outputs as appropriate.
- Machine independence.

As an example of adaptability the computer program may call for a list to be scanned. In a FORTRAN program this would probably be done with a DO-LOOP. If there is some uncertainty about the list size, the DO-LOOP upper limit can be an input parameter, so that it can be easily and universally changed if necessary. Likewise if a number of WRITE formats use a common list of labels which are subject to change, it might be better to input the labels rather than store them in the individual format statements. The important point is not so much these primitive examples themselves as the design objective of simplifying changes.

Obviously the simulation designer should be alert to the effect of computer limitations on simulation design, and so model design. The size of memory core storage is the most obvious current limitation, but running time, turn-around time, and input/output device selection are also significant. Depending upon his personal background the model designer may need little or considerable guidance from the simulation designer. However, the latter should resist the temptation to overwhelm the model designer by detailing the prohibitions placed on the model design, but rather seek to minimize them. The impossible cannot be achieved, but imaginative thinking can often reveal clever solutions to the problem at hand.

In many cases, the simulation design process will consume time and money comparable to if not greater than that devoted to operating the completed simulation. This fact focuses attention on the need for careful, systematic simulation design procedures. For example as a general rule the output should include an appropriate heading which adequately identifies the computer run. Adequate identification might include identification of the input base data used, the particular version of the simulation used (since it may be in a state of flux), the date, and some statement of the objectives of the run. Equally important is the proper and consistent use of such a heading. In the rush to meet a com-

puter run submission deadline while debugging, there is a temptation to bypass the process of updating the heading data. However, this is one place where haste does indeed make waste, and sooner or later the time and money expended on at least one computer run of a series is lost because the printout has lost its identity. At best the run must be repeated; at worst wrong conclusions are drawn and additional runs wasted.

It is equally important to keep careful records of changes to the program. It should always be possible to associate a given set of printouts with the specific program that produced it. This permits the model designer to track the evolving model design with the output of the simulation. Not uncommonly a change in the model (and its reflection as a change in the simulation) will produce an unexpectedly negative effect and the model designer will want to rescind the change. If he has failed to mark changes systematically, the simulation designer may have difficulty in retracing his steps.

A technique has been evolved at Raytheon for using the field 73–80 of the standard FORTRAN punch card to record changes. Field 73–78 is coded with the name of the subroutine, for example SAMPLE. Field 79–80 holds the serial number of the change. Assuming subroutine SAMPLE were included in the first attempt at compilation, field 73–80 for all punched cards would show SAMPLE 01. If no more changes were made until say the fifth batch of compilations, the new and changed cards would be identified as SAMPLE 05. Furthermore the simulation designer inserts a comment card at the head of SAMPLE which briefly identifies the 05 changes to SAMPLE and assists the model designer in controlling the growth of the simulation. When subroutine SAMPLE has been completed the final punched card deck can have a short subroutine identifier, i.e., SAMP, and sequential serial numbers inserted in field 77–80 by a standard utility program.

Finally the simulation designer makes a major contribution to the value of the model simulation by providing customer-oriented input and output formats, and straightforward and well-documented operating procedures. If he has done his job well, the completed simulation can be operated by the model designer directly as suggested by Fig. 1. Hopefully the input format matches the normal practices of the system analyst. If he is accustomed to thinking of a quantity in nautical miles, he obviously should not be required to input it in meters. Likewise output data should conform to and should be labeled in his terms, not computer program symbols. During the design and test of a simulation a variety of output formats will likely be developed for debugging. Since the specific data which the system analyst will want to see will vary with the purpose of the specific run, it is useful to be able to suppress specific output formats through the setting of control parameters. The analyst can concentrate on the subject of interest, and I/O device charges are minimized. Lastly, recognizing that one picture often is worth ten thousand words the simulation designer should be alert for situations in which graphical output would be useful to the systems analyst. Some languages such as DYNAMO[3] specifically include graphical output. Generalized programs have been developed which will produce graphs on a line printer.[4,5] Many computer systems now feature X-Y plotters, but conventional line printers can be used in a graphical mode, and can be assumed to be available at almost all computer installations.

Figure 2 is a plot of aircraft and decoy positions relative to a surface-to-air missile (SAM) site generated by the DECAP model. Notice that all labels are designed for easy reading. Distances are shown in nautical miles East-West, and North-South of the site. The program was designed so that the SAM site is automatically located in the appropriate quadrant of the map, and the scales are adjusted to match. In the case illustrated the program sensed that the cloud of targets was roughly South-West of the SAM site and so it located the SAM site in the North-West quadrant of the map. The symbols used for targets are defined at the right. The game and run are identified, and the specific time and location depicted is noted. Figure 3 is a plot of cumulative kill probability as a function of reentry vehicles used for each of four different attacks against a point target. It is part of the DACE model developed by Raytheon using the Bolt, Beranek, and Newman, Incorporated remote-access, time-share system, TELCOMP. The basic plotting function is a part of the system software. The simulation designer specified the headings, and scales, and modified the variables to be plotted to match the specifications of the PLOT function. Specifically the variable to be plotted may range from -1 to $+1$. In the case illustrated the variable to be plotted, cumulative kill probability, ranges from 0 to $+1$. The Instructions to transform and plot the variable take the following form:

```
PK (A, B) = (PK (A, B)  -  0.5) *2
PLOT      PK (A, B)      ON NUM
```

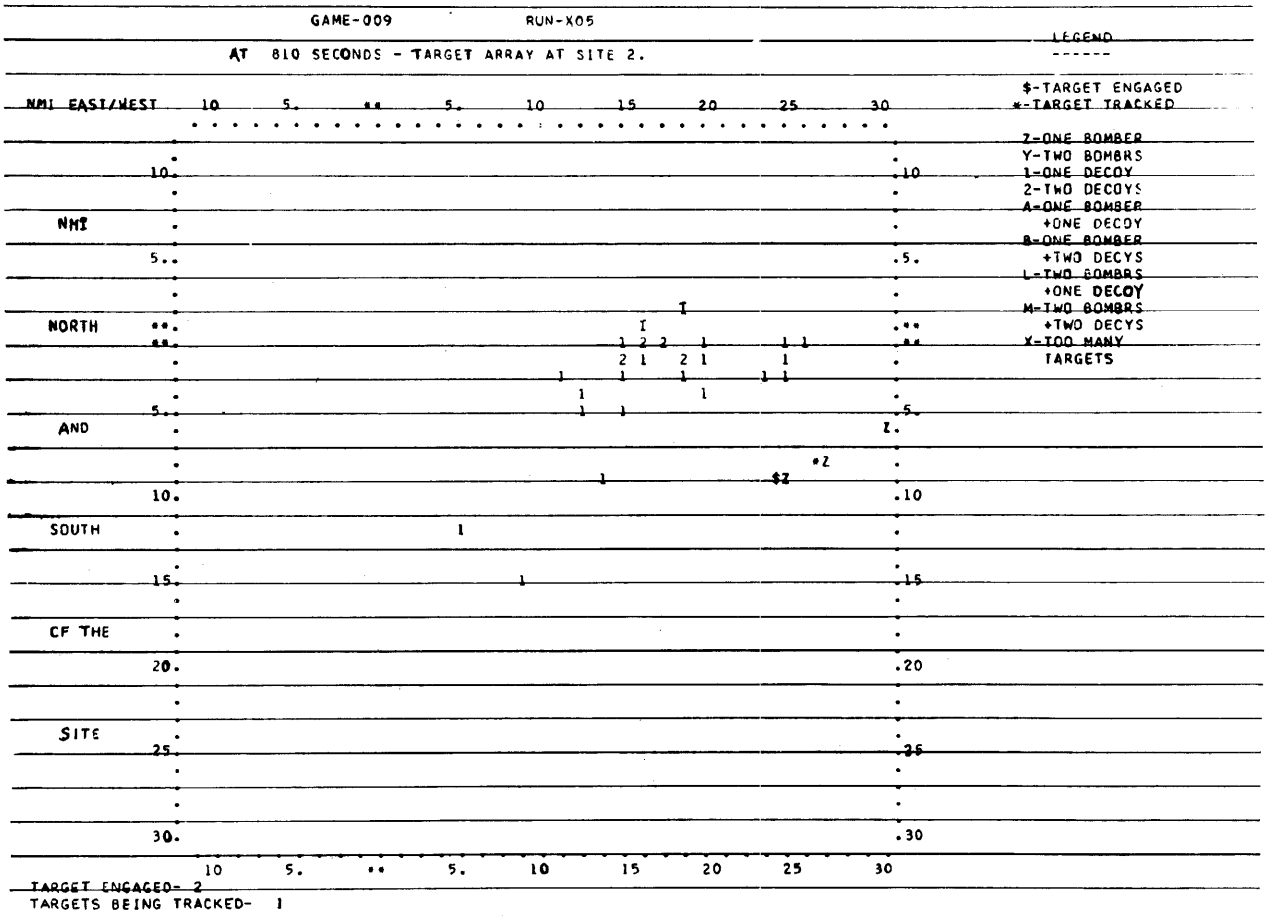


Figure 2. A computer-generated map to show disposition of aircraft, and decoys.

(A and B define the four different cases being studied and NUM is the number of reentry vehicles yielding the plotted kill probability.) The advantage to the simulation designer of the generalized instruction, PLOT, is that he can concentrate his attention on the problems of the specific simulation. Figure 3 also illustrates a valuable feature of on-line computing. The system analyst monitors the output from the simulation. When he obtains the desired results, he can interrupt the computation and go on to the next case.

The same effect can be achieved on batch computers by storing successive values of a variable on a scratch tape with appropriate tags. When the run is completed, the executive portion of the simulation can make use of an auxiliary program to provide appropriate headings and scales and to construct a plot of the variable as a function of the independent variable with which each value was tagged. TEFORM, a central war model, computes counterforce and countervalue potential ratios at each event cycle time and stores the values. These are brought back at the end of a game and converted into a time plot which follows the event initiated printout.

Since the above implicitly has been stated in terms of the current family of high-speed batch processing computers, one might ask what changes the new remote-access, time-sharing computers will have. The author's own limited experience suggests that the model designer will appear to absorb the simulation designer's job in large part because he will communicate directly with the computer directly through his remote console. In fact, it is likely that the simulation designer's function will remain but in a new form. First his role in a specific task will become consultive. In looking over the model designer's shoulder either literally, or figuratively, he will spot program needs which can be best met with techniques unfamiliar to the model designer, and suggest them. More important, however, will be the

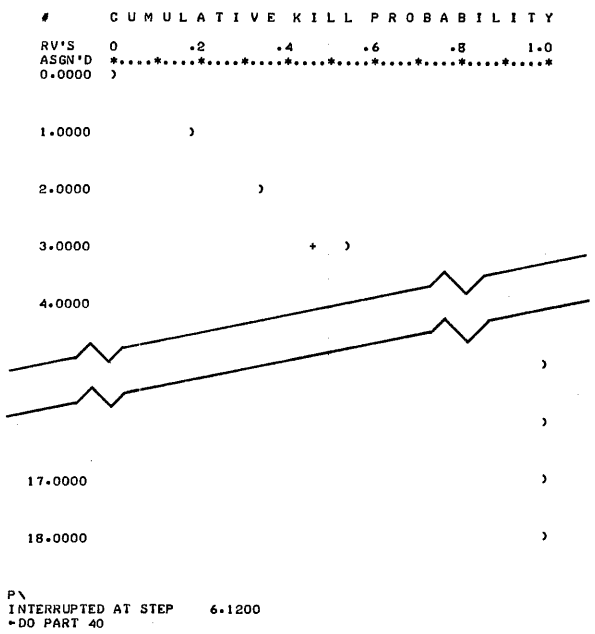


Figure 3. A graph generated by a computer in real time.

simulation designer's contributions to the programming language. Being able to anticipate at least to some extent the things the model designer is going to do, he can seek out ways to adapt the programming language to the needs. The other area in which the simulation designer can contribute is in applications of mass memories. If the computer system is capable of storing the input data and results from a large number of previous runs, the system analyst may want to be able to compare current runs with earlier runs. He may want to be able to repeat an earlier run with only one or few changes, and compare the results. Since the limitations on data links are not likely to be eased quickly, the remote-access, time-sharing system is probably going to be characterized by a low data transfer rate for some time to come. As a consequence the systems analyst will appreciate a programming language which enables him to state his problem succinctly, and to receive only the answer desired without being distracted by unwanted output. The simulation designer will be challenged in the decade ahead to bring to the systems analyst and the model designer the full power of on-line computing coupled to a mass memory.

To summarize, system simulations are susceptible to two broad defects, lack of credibility, and lack of accessibility. The first is within the control of the model designer. If he begins his design with the maximum simplicity rather than complexity, he can strive for uniformity of detail, and at the same time give the systems analyst a clear quantitative measure of the effect of departures from "complete accuracy." By imaginative design of input and output formats, and operating procedures, the simulation designer can give the systems analyst a computer program that is easy to operate, and adaptable to the problem at hand. In the last analysis all of the above comes down to being customer-oriented. Each individual in the loop serves best when he adopts the viewpoint of the person preceding him.

REFERENCES

1. M. R. Lackner, "Digital Simulation and System Theory," Document No. SP-1612, System Development Corp. (Apr. 6, 1964).

2. M. Greenberger, "A New Methodology for Computer Simulation," Document No. MAC-TR-13, MIT Project MAC.

3. A. L. Pugh III, *Dynamo User's Manual*, 2d ed., MIT Press.

4. R. G. West and J. R. Reynolds, "FORTRAN Programs for Plotting Two Dimensional Graphs," Document No. NMC-TM-65-31, U.S. Naval Missile Center (June 21, 1965).

5. G. H. Grace, "Application of Empirical Methods to Computer-Based System Design," Document No. SP-1952, System Development Corporation (June 1, 1965).

DSL/90—A DIGITAL SIMULATION PROGRAM FOR CONTINUOUS SYSTEM MODELING

W. M. Syn
Systems Development Division
and
Robert N. Linebarger
Data Processing Division
IBM Corporation, San Jose, California

INTRODUCTION

Computer simulation has been used for some time in the analysis and design of dynamic systems. With recent advancements in computer performance, the field of dynamic simulation—long the exclusive domain of the analog computer—has begun to utilize digital methods. No less than a score of digital simulation programs have appeared since R. G. Selfridge's pioneering effort in 1955; and the number is ever-increasing. These programs offer a convenient method of simulating continuous system dynamics employing well-known and easy-to-use analog computer programming techniques. The common starting point for such simulation is the conventional analog block diagram, and the common approach is the breakdown of the mathematical system model into its component parts or functional blocks. These blocks, having a near one-to-one correspondence with analog computing elements such as integrators, summers, limiters, etc., usually appear as subroutines within the simulator program. Using one of the simulation packages, "programming" involves no more than merely interconnecting the functional blocks by a sequence of connection statements according to the rules laid down by the input language. This interconnecting of blocks is analogous to the wiring of the patchboard on an analog computer. Therefore, these digital-analog simulation programs combine the best features of the analog and digital computers: the flexibility of block connection structure of the former and the accuracy and reliability of the latter.

DSL/90 is a new digital simulation package for the 7090 family of computers. The program is available from the SHARE library (IWDSL No. 3358). Its development, from drawing board to production code, was guided by the following broad objectives:

- To incorporate within it all the desirable and proven features of its predecessors;
- To make this useful technique of digital simulation attractive to a group of users who are not analog-computer-oriented, yet retain the large following of analog programmers who are devoted to the building-block approach to system analysis;
- To provide a "continuous system simulator" program that is applicable to a broad range of continuous system analysis and not restrained by conventional digital-analog simulator techniques.

Some of the DSL/90 features are:

- A library of DSL system blocks such as integrator, limiter, summer, etc.;
- A simple nonprocedural applications-oriented input language specifying the rules for connecting the library blocks together;
- An input routine which permits quick and easy parameter entry and data changes;
- Complete print output routines including a graphical output facility;
- Choice of numerical integration routines with or without error bounds using centralized or noncentralized integration schemes;
- Automatic sequencing of input language statements (this is called "sorting" in programs such as ASTRAL and MIDAS);
- Facility to add to the DSL/90 library any user-defined blocks in the form of subroutines (FORTRAN, MAP or binary decks);
- Intermixing of DSL and FORTRAN language statements;
- Repeatability of language statements (macro-generation);
- Dynamic storage of data.

Although DSL/90's input language statements are block-oriented, they are not restricted solely to block notation. DSL/90 permits an intermixing of its input language statements (henceforth called DSL statements) and FORTRAN IV statements. Thus, the power of FORTRAN is made available to the problem solver. One far-reaching implication of this language feature is that simulation "programming" may begin anywhere from the analog block diagram formulation of the problem to the higher-level mathematical model in the form of ordinary differential equations.

OPERATIONAL FEATURES

Basic Language Features

The DSL/90 language statements may be classified into three general categories: 1) structure or connection statements which define the interconnection of the functional blocks, 2) data statements which permit the entry of alphanumeric information, and 3) simulation control statements.

The Connection Statements. In the DSL/90 input language, the basic functional block is characterized by an output (outputs) that is functionally related to one or more inputs. Parameter names and initial conditions, if any, are also included in the statement which has the following general form:

Outputs = Block name (Initial conditions, Parameters, Inputs)

Below are examples of basic DSL connection or structure statements:

1. OUTNAM = SQRT (TEMP)

In the block diagram representation (Fig. 1), SQRT is the name of the functional block. It has a single input called TEMP and the output is given the name OUTNAM.

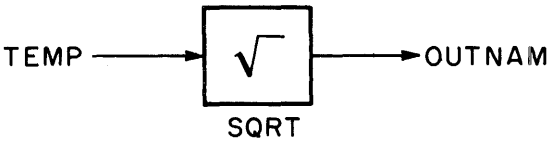


Figure 1.

2. Y = INTGRL (IC2, YDOT)

Figure 2 represents the block INTGRL which is the basic DSL/90 integrator block. IC2 and YDOT are its initial condition and input name respectively.

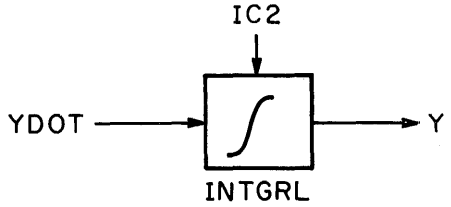


Figure 2.

3. OUT1, OUT2 = VALVE (LEVEL, INHI, INMED, INLO)

Figure 3 illustrates a user-supplied functional block named VALVE with two outputs OUT1 and OUT2. LEVEL is a unique parameter name se-

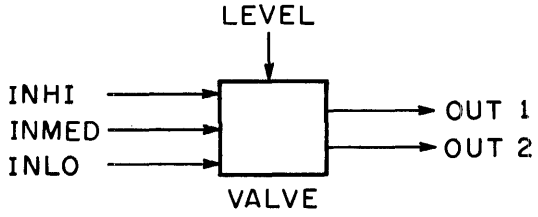


Figure 3.

lected by the user, and INHI, INMED and INLO are the names of the three input variables to the block.

From the above illustrations, it should be evident that a functional block in the DSL/90 language is completely specified by the unique names assigned to the inputs and outputs of each block. The user is free to select names meaningful to his process simulation, the only restriction being that a name consists of no more than 6 alphanumeric characters, the first of which is alphabetic. User-supplied blocks may have any name following the same restriction above. However, the names of standard blocks supplied as part of the DSL/90 simulation package are preassigned. DSL/90 provides an extensive library of functional blocks which are listed in Table 1.

The above format for characterizing functional blocks in DSL/90 is consistently adhered to. However, there are these exceptions: the basic operations of multiplying, dividing, summing and subtracting are replaced by the operators *,/, + and −, respectively. To this list of operators we add ** for exponentiation. Let us illustrate one of these operations by simulating a multiplier output (Fig. 4),

$$\text{OUT} = A \cdot B.$$

Table 1. Functional Description of Standard DSL/90 Blocks

	GENERAL FORM	FUNCTION
**	Y = INTGRL (IC, X) Y(0) = IC INTEGRATOR	$Y = \int_0^t X\,dt + IC$ EQUIVALENT LAPLACE TRANSFORM: $\frac{1}{S}$
*	Y = MODINT (IC, P_1, P_2, X) MODE-CONTROLLED INTEGRATOR	$Y = \int_0^t X\,dt + IC$ $P_1 = 1, P_2 = 0$ $Y = IC$ $P_1 = 0, P_2 = 1$ Y = LAST OUTPUT $P_1 = 0, P_2 = 0$
*	Y = REALPL (IC, P, X) Y(0) = IC 1ST ORDER SYSTEM (REAL POLE)	$P\dot{Y} + Y = X$ EQUIVALENT LAPLACE TRANSFORM: $\frac{1}{PS+1}$
*	Y = LEDLAG (IC, P_1, P_2, X) Y(0) = IC LEAD-LAG	$P_2\dot{Y} + Y = P_1\dot{X} + X$ EQUIVALENT LAPLACE TRANSFORM: $\frac{P_1S+1}{P_2S+1}$
*	Y = CMPXPL (IC_1, IC_2, P_1, P_2, X) Y(0) = IC_1 $\dot{Y}(0) = IC_2$ 2ND ORDER SYSTEM (COMPLEX POLE)	$\ddot{Y} + 2P_1P_2\dot{Y} + P_2^2Y = X$ EQUIVALENT LAPLACE TRANSFORM: $\frac{1}{S^2+2P_1P_2S+P_2^2}$
	Y = DERIV (IC, X) Y(0) = IC DERIVATIVE	$Y = \frac{dx}{dt}$ QUADRATIC INTERPOLATION EQUIVALENT LAPLACE TRANSFORM: S
	Y = DELAY (N, P, X) P = TOTAL DELAY IN TERMS OF INDEPENDENT VAR. N = MAX NO. OF POINTS DELAY DEAD TIME (DELAY)	$Y(t) = X(t-P)$ $t \geq P$ $Y = 0$ $t < P$ EQUIVALENT LAPLACE TRANSFORM: e^{-PS}
	Y = ZHOLD (P, X) Y(0) = 0 ZERO-ORDER HOLD	$Y = X$ $P = 1$ Y = LAST OUTPUT $P = 0$ EQUIVALENT LAPLACE TRANSFORM: $\frac{1}{S}(1-e^{-st})$
	Y = IMPL (IC, ERROR, FUNCT) IMPLICIT FUNCTION	$Y = IC$ $t = 0$ FIRST ENTRY $Y = FUNCT(Y)$ $t \geq 0$ $\lvert Y - FUNCT(Y)\rvert \leq ERROR \cdot \lvert Y\rvert$

SWITCHING FUNCTIONS

Y = FCNSW (P, X_1, X_2, X_3) FUNCTION SWITCH	Y = X_1 P < 0 Y = X_2 P = 0 Y = X_3 P > 0
Y = INSW (P, X_1, X_2) INPUT SWITCH (RELAY)	Y = X_1 P < 0 Y = X_2 P ≥ 0
Y_1, Y_2 = OUTSW (P, X) OUTPUT SWITCH	Y_1 = X, Y_2 = 0 P < 0 Y_1 = 0, Y_2 = X P ≥ 0
Y = COMPAR (X_1, X_2) COMPARATOR	Y = 0 $X_1 < X_2$ Y = 1 $X_1 \geq X_2$
Y = RST (P_1, P_2, P_3) RST FLIP-FLOP	Y = 0 $P_1 > 0$ Y = 1 $P_2 > 0$, ($P_1 \leq 0$) Y = 0 $P_3 > 0$, $Y_{n-1} = 1$, ($P_2 \leq 0$, $P_1 \leq 0$) Y = 1 $P_3 > 0$, $Y_{n-1} = 0$, " "

* THESE FOUR BLOCKS EXIST AS BUILT-IN MACROS WITHIN DSL. IN-LINE CODE REPRESENTING AN EQUIVALENT INTEGRATOR CIRCUIT IS GENERATED FOR EACH USE TO PERMIT THE USE OF CENTRALIZED INTEGRATION SCHEMES WITHIN THE BLOCKS.

** INTGRL MUST BE THE RIGHTMOST TERM FOR EACH LEVEL OF USAGE. IF X IS A SINGLE VARIABLE NAME THEN IT MUST BE UNIQUE WITHIN THE PROBLEM. IC MUST ALSO BE UNIQUE. (-IC IS NOT VALID). A LITERAL MAY BE USED FOR IC. ALSO SEE SECT. 5-1.

We have decided not to use OUT = MULT (A, B), but simply OUT = A*B. Let us summarize these ideas by considering a solution to Mathieu's equation:

$$\ddot{y} + (1 + A \cos t)\, y = 0 \qquad \dot{y}(0) = 0, y(0) = Y0$$

As the DSL connection statements for this circuit follow a near one-to-one correspondence with the functional blocks in Fig. 5, they may be written as:

```
FCN    = A * COS (TIME)
MULT   = FCN * Y
Y2DOT  = -Y - MULT
YDOT   = INTGRL (0., Y2DOT)
Y      = INTGRL (Y0, YDOT)
```

(Note that TIME is a DSL system name representing the independent variable of integration. It may easily be renamed by the user.)

Observe that the DSL statements in the above example are also FORTRAN arithmetic statements, and the right-hand portions of the statements are merely FORTRAN expressions. Therefore, as such, their complexity is restricted only by the rules that govern arithmetic expressions in the FORTRAN language.

Furthermore, these expressions can serve as inputs to any functional block, regardless of whether it is a DSL/90 or user-supplied block. For example, the first three DSL structure statements in the problem above may be written as one statement,

```
Y2DOT = -Y - A * COS (TIME) * Y;
```

or perhaps as

```
Y2DOT = -Y * (1. + A * COS (TIME)).
```

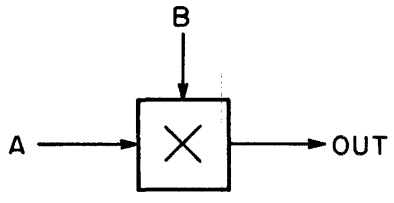


Figure 4.

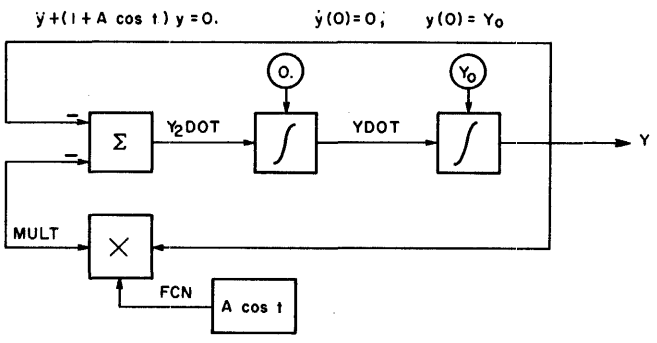
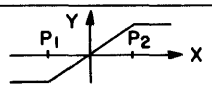
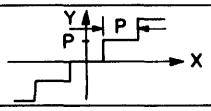
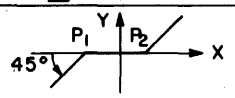
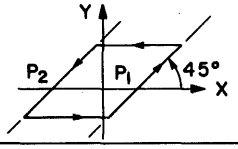
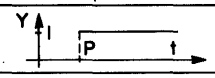
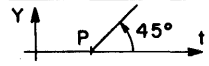
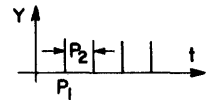
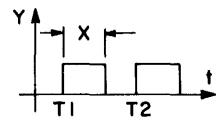
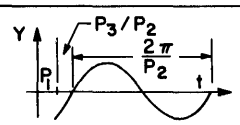
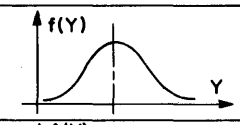
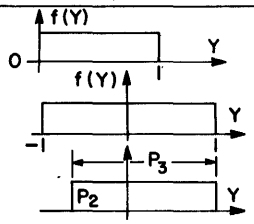


Figure 5.

FUNCTION GENERATORS

GENERAL FORM	FUNCTION
Y=AFGEN (FUNCT, X) ARBITRARY LINEAR FUNCTION GENERATOR	Y=FUNCT(X) $X_0 \leq X \leq X_n$ LINEAR INTERPOLATION Y=FUNCT(X_0) $X < X_0$ Y=FUNCT(X_n) $X > X_n$
Y=NLFGEN (FUNCT, X) NON-LINEAR FUNCTION GENERATOR	Y=FUNCT(X) $X_0 \leq X \leq X_n$ QUADRATIC INTERPOLATION (LA GRANGE) Y=FUNCT(X_0) $X < X_0$ Y=FUNCT(X_n) $X > X_n$
Y=LIMIT (P_1, P_2, X) LIMITER	Y=P_1 $X < P_1$ Y=P_2 $X > P_2$ Y=X $P_1 \leq X \leq P_2$ 
Y=QNTZR (P, X) QUANTIZER	Y=kP $(k-1/2)P < X \leq (k+1/2)P$ $k = 0, \pm 1, \pm 2, \pm 3, \ldots$ 
Y=DEADSP (P_1, P_2, X) DEAD SPACE	Y=0 $P_1 \leq X \leq P_2$ Y=X−P_2 $X > P_2$ Y=X−P_1 $X < P_1$ 
Y=HSTRSS (IC, P_1, P_2, X) Y(0)=IC HYSTERESIS LOOP	Y=X−P_1 $(X-X_{n-1}) > 0$ AND $Y_{n-1} \leq (X-P_1)$ Y=X−P_2 $(X-X_{n-1}) < 0$ AND $Y_{n-1} \geq (X-P_2)$ OTHERWISE Y=LAST OUTPUT 
Y=STEP (P) STEP FUNCTION	Y=0 $t < P$ Y=1 $t \geq P$ 
Y=RAMP (P) RAMP FUNCTION	Y=0 $t < P$ Y=t−P $t \geq P$ 
Y=IMPULSE (P_1,P_2) IMPULSE GENERATOR	Y=0 $t < P_1$ Y=1 $(t-P_1) = kP_2$ Y=0 $(t-P_1) \neq kP_2$ $k = 0, 1, 2, 3, \ldots$ 
Y=PULSE (P, X) PULSE GENERATOR WITH P AS TRIGGER	Y=0 INITIAL Y=1 $T_k \leq t < (T_k + X)$ Y=0 OTHERWISE $k = 1, 2, 3, \ldots$ T_k=t OF PULSE k, P_k 
Y=SINE (P_1, P_2, P_3) P_2=FREQUENCY IN RADIANS/SEC. P_3=PHASE SHIFT IN RADIANS TRIGONOMETRIC SINE WAVE WITH AMPLITUDE, PHASE, AND DELAY	Y=0 $t < P_1$ Y=SIN $[P_2 \cdot (t-P_1) + P_3]$ $t \geq P_1$ 
Y=NORMAL (P_1, P_2, P_3) NOISE GENERATOR (NORMAL DISTRIBUTION)	Y=GAUSSIAN DISTRIBUTION WITH MEAN, P_2, AND STANDARD DEVIATION, P_3 (P_1=ANY ODD INTEGER) 
Y=UNZRPI (P_1) Y=UNMIPI (P_1) Y=UNATOB (P_1, P_2, P_3) NOISE GENERATOR (UNIFORM DISTRIBUTION)	Y=UNIFORM DISTRIBUTION 0 TO 1 (P_1=ANY ODD INTEGER) Y=UNIFORM DISTRIBUTION, −1 TO +1 Y=UNIFORM DISTRIBUTION, P_2 TO P_2+P_3 

In addition, if the output, YDOT, of the first integrator is not a variable of interest, the two integrators may be "nested" as follows:

```
Y = INTGRL (Y0, INTGRL (0., Y2DOT)).
```

Finally, if the variable Y is the only one whose output is desired, the problem may be described by a single DSL connection statement, namely,

```
Y = INTGRL (Y0, INTGRL (0., – Y *
    (1. + A * COS (TIME)))).
```

The Data Statements. The subject of data entry was given prime consideration during the development of language features of DSL/90. The end result is free-form and symbolic specification of parameter values and initial conditions following a card identifier label which is punched left-adjusted in the first six columns of a data card. For example,

```
Cols  1–6      7–72
      PARAM    A = 0.5, PAR1 = 62.4,
                 PAR2 = 3.215 E + 4
      INCON    ICI = 0.2, XDOT = 1.3
      CONST    C1C = 7.3, C2C = 100.,
                 T = 46.25,
               EPSILN = 1.0 – 05
```

The identifying labels begin in column one. The data items, separated by commas, may be placed anywhere in columns 7–72. Blanks are ignored. Three consecutive decimal points at the end of any statement indicate that it is to be continued on the next card. Continuation may begin anywhere in columns 1–72. Data statements may be intermingled with connection statements.

The Control Statements. The statements may be conveniently grouped into three types:

1. Problem output control statements include print and plot requirements, title information and labeling of graphs, such as:

```
PRINT    .01, Y, Y2DOT
PREPAR   .005, Y, Y2DOT
GRAPH    8., 6., TIME, Y, Y2DOT
LABEL    SOLUTION OF MATHIEU'S
           EQUATION
RANGE    DELT, X
```

The above cards will cause the printing of TIME, Y, and Y2DOT at intervals of 0.01 units of time, and preparation of TIME, Y, and Y2DOT for graphing at intervals of 0.005 units of time. A single 8 × 6-inch graph properly labeled as directed, will be made with Y and Y2DOT plotted vs TIME. The maximum and minimum values attained by DELT and X will be printed at the end of the run.

2. Problem execution control statements are used to set error bounds and step size for integration routines, prescribe run cutoff conditions, and to specify other pertinent run information. Typical examples are

```
CONTRL DELT = .05, FINTIM = 2.0
ABSERR YDOT = 1.0 E – 5, Y = 5.0 E – 4.
```

The simulation will be executed from 0 to 2.0 with an integration interval of 0.05. The error bounds on YDOT and Y will be held at 1.0×10^{-5} and 5.0×10^{-4}, respectively. The latter bound will be applied to all other unspecified integrator outputs.

3. System control statements provide the user with a number of options, the most important ones being choice of integration methods, bypassing the sequencing routine, and renaming of system variables. They also include an END card which signifies the end of a logical set of data cards, and a STOP card which ends the computer run.

For example:

```
CONTIN
INTEG     MILNE
NOSORT
RENAME    TIME = X, DELT = DELX
FINISH    DIST = 0.
```

These cards cause continuation of the simulation from the last calculated point, selection of the Milne 5th-order integration scheme, exercise of the no-sort option, renaming of two systems variables, and termination of the run when the value of DIST reaches zero.

All data and control cards, with the exception of the END and STOP cards and certain logical groups of cards (such as continuation statements) may be intermixed with DSL structure statements and may appear in any order. Proper statement order is determined by an internal sort based on correct information flow. Table 2 shows a complete list of DSL/90 data and control statements. Returning to Mathieu's equation, a complete DSL/90 program for $\ddot{y} + (1 + A \cos t)\, y = 0$ may be written as follows:

```
1–6       7–72
TITLE     SOLUTION OF MATHIEU'S
            EQUATION
          Y2DOT = –Y*(1.0 + A * COS
          (TIME))
PARAM     A = 0.5
          Y = INTGRL (Y0, INTGRL (0.,
          Y2DOT))
INCON     Y0 = 20.0
INTEG     MILNE
```

TABLE 2 Summary of DSL/90 Data Statement Formats

Label	Function (By Example)
COL. 1-6	7-72
PROBLEM DATA INPUT:	
PARAM	TAU = 25., PAR = 3.158E3, C4 = 2.0 E-5
CONST	CON1 = 45.3, PI = 3.14159, K = 3
INCON	IC1 = 20., A = 50.2, IC3 = 0
AFGEN	FCN = 3., 25., 5.2, 26.4, 6.0, 24., 7.5, 21.3
NLFGEN	FY3 = 0., 850., 5., 1245., 8., 1.574E3, 12.4, 2.4E03
TABLE	PAR1(8) = 4.5, INPUT(1-4) = 2., 2*8.6, 3.52E3
PROBLEM OUTPUT CONTROL:	
PRINT	0.1, X, XDOT, VELOC
TITLE	MASS, SPRING, DAMPER SYSTEM IN DSL/90
PREPAR	.05, X, Y, XDOT
GRAPH	10., 8., TIME, X, XDOT
LABEL	MASS, SPRING, DAMPER SYSTEM - 6/1/65
RANGE	X, XDOT, VELOC, DELT
PROBLEM EXECUTION CONTROL:	
CONTRL	DELT = .002, FINTIM = 8.0, DELMIN = 1.0E-10
FINISH	DIST = 0., ALT = 5000.
RELERR	X = 1.E-4, XDOT = 5.E-5
ABSERR	X = 1.E-3, XDOT = 1.E-4
CONTIN	
INTEG	MILNE
RESET	GRAPH, PRINT
DSL/90 TRANSLATOR PSEUDO-OPERATIONS:	
RENAME	TIME = DISPL, DELT = DELTX
INTGER	K, GO
MEMORY	INT(4), DELAY (100)
STORAG	IC(6), PARAM (10)
DECK	
SORT	
NOSORT	
PROCED	X = FCN (A, B, PAR5, IC3)
⋮	
ENDPRO	
MACRO	OUT = FCN2 (IC1, R, T, X)
⋮	
ENDMAC	
END	
STOP	

```
CONTRL   DELT = .02, FINTIM = 2.0
ABSERR   Y2DOT = 1.0E-5, Y = 2.0 E-5
PRINT    0.05, Y, Y2DOT
END
STOP
```

It should be apparent by now that the DSL input language is block-oriented, symbolic, and free-form. The use of FORTRAN is not limited to arithmetic statements. All FORTRAN library functions such as SQRT, SIN, COS, etc., are available. Under the rules which are clearly defined within DSL/90, a large subset of FORTRAN becomes available to the simulation user without sacrificing the ease of block notation programming. What this means to the engineer who is unskilled in FORTRAN programming is simply this: he can still perform his process simulation with a simple language, following a step-by-step building block approach. As he becomes more proficient, his programming becomes correspondingly more efficient and he may want to include elementary FORTRAN language features in his connection statements. Still later, as the complexity of his problem increases, he may use to advantage the more powerful features of DSL and FORTRAN.

Advanced Language Features

There are a number of other DSL/90 language features which are especially useful for the simulation of large or complex problems. We shall examine several of these.

Procedural Statements. Recall that the order in which DSL statements are entered is unimportant because connection statements are separated from the rest and sequenced (or "sorted") by the DSL processor (unless a "no-sort" option is exercised). In other words, the DSL/90 language may be considered as nonprocedural. In contrast, FORTRAN is a procedural language since FORTRAN statements are executed in the order in which they are written. Frequently, in a complex process simulation, it is desirable to introduce procedural statements within the simulation program. The purpose may be to control signal flow in certain portions of the program, or perhaps to compute a large number of parameter values once and only once. DSL/90 uses a pair of pseudo-operations, PROCED and ENDPRO, punched in columns 1–6, to designate the beginning and end of a block of procedural statements (they may be DSL or FORTRAN statements). Input and output names may be specified on the PROCED card to allow the procedural statements to be sorted as a block relative to other DSL statements. For example:

```
PROCED   TEMP = BLOCKA (TEST, IN)
         IF (TEST) 10, 10, 20
    10   TEMP = LIMIT (PAR1, PAR2, IN)
         GO TO 30
    20   TEMP = IN + TEST
    30   CONTINUE
ENDPRO
```

During the sequencing of DSL statements, the above procedural statements will be treated as a single functional block with output TEMP and inputs TEST and IN, as illustrated in Fig. 6. The order of the statements within the procedural block remains unchanged.

Macro-Generation. Pseudo-operations MACRO and ENDMAC, which are punched in columns

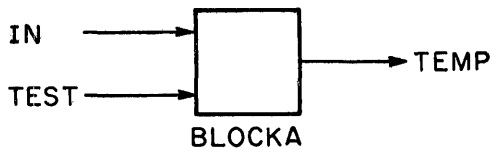


Figure 6.

1–6, are used to define a macro block. One may think of a macro as a repeatable procedural block with parameter variations. This is best illustrated by example. The following statements constitute a macro-definition:

```
1-6       7-72
MACRO     OUT = FILTER (V1, V2, K, IN)
          V1 = (IN - V2)/K
          V2 = INTGRL (0., V1)
          OUT = V2 + 0.5*V1
ENDMAC
```

During the definition of the macro, no language statements are produced. The name of this macro, FILTER, must be unique. However, the output name OUT and the input names, V1, V2, K, and IN, are dummy symbols which will be replaced by the actual names specified at the time when the macro is used. The subsequent appearance of the statement

```
LINE 1 = FILTER (A1, A2, TAU, XIN)
```

will cause the following three statements to be generated in-line:

```
A1      = (XIN - A2)/TAU
A2      = INTGRL (0., A1)
LINE1   = A2 + 0.5*A1
```

Just as in the case of the procedural block, these statements will be sequenced as a single functional block with LINE1 as output and A1, A2, TAU and XIN as inputs (see Fig. 7). The statements within the block are not sorted. Both DSL and FORTRAN statements may appear within a macro.

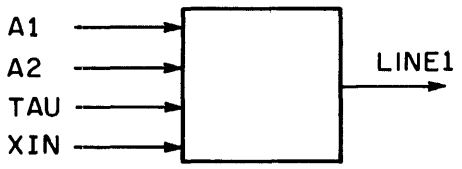


Figure 7.

Implicit Function Block. DSL/90 provides an implicit function block called IMPL for the solution of an implicit equation $f(y) = 0$ expressed in the form of $y = f(y)$. Clearly some iterative technique must be employed. These iterations must be performed within each integration interval until a convergence criterion is satisfied. The program for IMPL uses the direct iteration method developed by Wegstein. If there is no convergence after some preassigned maximum number of iterations, the simulation of the problem is terminated with appropriate diagnostic printout.

To use the implicit function block, one writes the DSL statement,

```
Y = IMPL (YO, ERROR, FOFY)
```

followed by the set of DSL or FORTRAN (or both) statements evaluating FOFY. Y, YO, ERROR and FOFY are symbolic names selected by the user. The DSL/90 system then sets up the necessary iterative loop. Let us illustrate by solving the implicit equation

$$y = \frac{C \cdot (e^y - 1)}{e^y} \qquad \text{(C is some constant)}$$

One simply writes:

```
Y    = IMPL (YO, ERROR, FOFY)
A    = EXP(Y)
FOFY = C* (A - 1.0) / A
```

The DSL/90 translator will automatically generate the following statements:

```
30001Y = IMPL (YO, ERROR, FOFY)
     IF (NALARM .LE.0) GO TO 30002
     A      = EXP (Y)
     FOFY = C* (A - 1.0) / A
     GO TO 30001
30002 CONTINUE
```

Note that three statements, and only those three, are added to the ones written by the user. The first time the IMPL routine is entered, NALARM is set to one, and Y is given the initial guess Y0. After each calculation of $f(y)$, program flow returns to the IMPL subroutine where the convergence criterion is tested. If satisfied, NALARM is set equal to zero and y assumes the most recently calculated value of $f(y)$. Otherwise the iteration continues.

User-Supplied Functional Blocks. Although DSL/90 provides an extensive library of operational blocks, there are occasions when special blocks are required to simulate specific process elements. These special blocks are programmed by the user as subroutines either in FORTRAN or MAP and simply added to the data at the time the simulation run is made. The user may treat these special blocks like all other DSL library blocks, interconnecting them to build a complex system model.

As an example of the use of special blocks, consider the modeling of the analog-to-digital converter shown as a nonlinear stepwise quantization in Fig. 8. If no such general block existed in the DSL library, it would be difficult to construct such a characteristic from the standard blocks available. How-

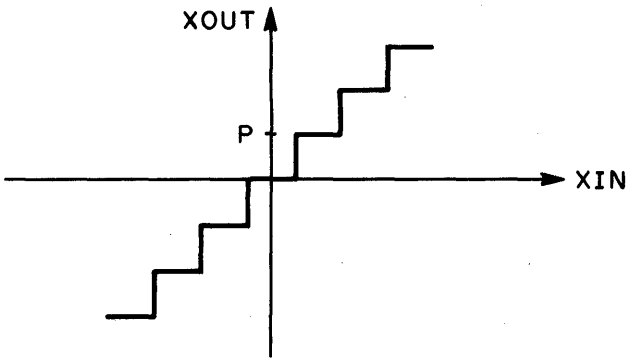


Figure 8.

ever, the quantization effect is easily modeled by the following FORTRAN statements:

```
FUNCTION QNTZR (P, XIN)
QNT = AINT (0.5 + ABS (XIN)/P)
QNTZR = SIGN (P*QNT, XIN)
RETURN
END
```

The parameter named P containing the value of the quanta step size is the only parameter supplied to the QNTZR block. This value of P is entered into the simulation program in exactly the same way as any other DSL parameter—on a PARAM card. Note also that the two blocks AINT (for truncation) and SIGN (for transfer of sign) are standard subroutines of the FORTRAN library. The above FORTRAN subprogram for the quantizer may be entered directly with the data cards for the simulation run, or as an alternative, it may be compiled independently and the resulting machine language deck (binary deck) added to the data deck. This functional block may even be added to the permanent DSL library by simply loading it on the library tape. In fact this was the case with the QNTZR block when we found it to be sufficiently useful to warrant a place in the DSL library. The ease with which a difficult nonlinearity has been modeled in a few lines of FORTRAN coding is quite apparent and typifies the flexibility of DSL/90 for handling nonlinear functions and special blocks.

Arbitrary Functions. DSL/90 provides two functional blocks, AFGEN and NLFGEN, for handling arbitrary functions of one variable. The x, y coordinates of the function points are entered sequentially following an identifying label and the symbolic name of the function, e.g.:

```
1-6      7-72
AFGEN FC1 = -10.2, 2.3, -5.6, 6.4, 1.0, 5.9, etc.
```

Although the total number of data storage locations is necessarily fixed by machine size, there is no restriction on the number of points one may use to define any function. The only requirement is that the x coordinates in the sequence $x_1, y_1, x_2, y_2, \ldots$ are monotonically increasing. Any number of arbitrary functions may be defined, identified only by their symbolic names assigned by the user. As an example, the DSL statement Y3 = AFGEN (FC1, XIN) will refer to the function called FC1. AFGEN provides linear interpolation between consecutive points, while NLFGEN uses a second-order Lagrange interpolation formula.

Tabular Data. This feature of DSL/90 allows blocks of data to be transmitted to the UPDATE subroutine in tabular form. In the construction of a special block, the user may have to consider sets of initial conditions, history and input parameters. This DSL/90 feature will eliminate the need for a lengthy subroutine argument string. To illustrate, suppose we wish to build a special block called SPEC which requires two initial conditions and 10 parameters. We begin by writing the following two DSL statements:

```
1-6      7-72
STORAG   IC(2), PAR(10)
TABLE    IC(1) = 2.0, IC(2) = 0.0, PAR(1)
           = 4., PAR(2-10) = 9*1.5
```

The first statement instructs the DSL/90 system to assign a total of 12 locations—2 for the array IC and 10 for PAR. The second statement illustrates the manner in which numeric values are entered into these reserved locations. Now, when we subsequently use a statement such as

```
YOUT = SPEC (IC, PAR, XINPUT)
```

DSL/90 system will replace the names IC and PAR with the addresses of the first locations of the arrays IC and PAR respectively. Obviously, the user when programming his subroutine SPEC must realize that the first two arguments in SPEC are location pointers to his arrays. His subroutine could begin with the following:

```
FUNCTION SPEC (LOCIC, LOCPAR, XIN)
COMMON/CURVAL/C(1)
I = LOCIC
J = LOCPAR
```

CURVAL is the labeled common where the current values of all variables are stored, and I and J are indices referencing the first initial conditions IC and parameter values PAR.

System Features

DSL/90 System Organization. The DSL/90 Operating System is separated into two major functions: language translation and model simulation. Each function operates independently under standard IBSYS control but as one continuous single-pass operating system. The transition is made by having the translator develop on an IBSYS scratch tape all the elements of a standard IBSYS job as well as the representation of the model to be simulated. This tape is then switched in as the standard IBSYS input for compilation and execution to complete the simulation. Diagnostics are printed if errors are found in translation or simulation. Elements which may appear as input to the translator are: 1) DSL/90 problem-oriented language sentences to describe the model, 2) data input to the model for parameter values and control of the simulation and output, 3) binary and BCD subroutines and functions supplied by the user for the simulation, and 4) appropriate controls to load binary or BCD subroutines and functions from a library tape. The entire system may be placed at any level of a standard batched IBSYS run. Three additional tape drives are required—two auxiliary and one for plotting.

DSL/90 may be run as an independent program or it may be used as a subprogram of a conventional FORTRAN program for control purposes.

Sort. A nonprocedural input language such as DSL/90 transfers the responsibility of establishing the execution sequence from the user to the program. To accomplish this DSL/90 alters the sequence of input statements according to the rule: an operational element (or statement) is properly sequenced if all its inputs are available either as input parameters or initial conditions or as previously computed values in the current iteration cycle. Unspecified algebraic loops are identified and, if any, the run is halted. The result of this sequencing operation is a properly organized FORTRAN IV subprogram.

Main Program Control. DSL/90 provides for calling the simulation routines from a MAIN program specified by the user. Hence the actual digital simulation may be placed under control of a FORTRAN routine compiled at execution time. This feature allows for testing of response conditions, matching boundary values, and dynamic alteration of parameters, initial conditions, or run control data between parameter studies.

Centralized Integration. By use of the block name, INTGRL, a user may specify that centralized integration is desired. The translator sets up statements so as to compute all inputs to the integrators but bypass computation of outputs until the end of the iteration cycle. At this time, all integrator outputs are updated simultaneously. A choice can be made between the 5th-order Milne Predictor-Corrector, 4th-order Runge-Kutta, Simpson's Trapezoidal, or Rectangular Integration methods. The first three allow the integration interval to be adjusted by the system to meet a specified error criterion, a factor which allows it to take large or small steps depending on the rate of change of one or more variables. There is provision in DSL/90 for the user to supply his own integration scheme, which may or may not be centralized.

Dynamic Storage Allocation. Data in DSL/90 is stored in a single vector including current values of structure variables and table values for function generators, integration history, error bounds, STORAG variables, etc. The storage is allocated dynamically (i.e., at execution time) according to what portions of the simulator are used and how many integrators, tables, and structure variables are in the simulation model. Standard DSL/90 blocks are loaded only if used.

APPLICATIONS

Having illustrated operational features of the DSL/90 digital simulation program, we will now draw upon the previous introduction to show how DSL/90 has been flexibly applied to simulation problems. Three specific simulations will be considered: 1) a biomedical block notation problem involving a respiratory servomechanism; 2) a process analysis problem involving the simulation of heat transfer dynamics of a recirculating furnace used in the glass industry; and 3) the simulation of the flight dynamics of a portion of the SATURN V booster rocket.

DSL/90 provides special programming features such as different integration methods, sorting, special blocks, etc., which make it attractive to the user for continuous system simulation. Several of these features will be illustrated in the examples to follow.

Application No. 1—Respiratory Servo Simulation

This problem involves evaluating the response of a proposed model for respiratory control of CO_2

partial pressure in the venous and arterial blood streams of a human. De Fares et al performed the original study on an analog computer and represented the basic CO_2 control mechanism in respiration by the three-compartment model shown in Fig. 9. Using the original study as a guide, this first example will illustrate the ease of handling conventional analog simulation problems using DSL/90.

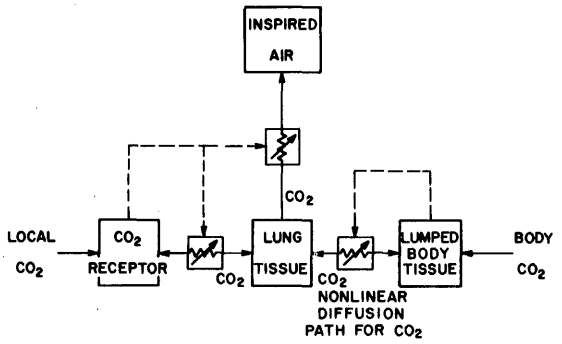


Figure 9. CO_2 control model.

The CO_2 control system operates as follows: The alveolar tissue in the lung serves as an exit sink for CO_2 production and possesses both CO_2 capacity and conductance characteristics. In a similar manner, body tissue can be considered as having an equivalent CO_2 capacitance and conductance. CO_2 produced by the body is partially stored in the local body tissue, raising the local body tissue partial pressure of CO_2. The CO_2 produced is simultaneously diffused through the tissue and picked up by the blood stream (venous path). The CO_2 is then carried to the lung and subsequently diffused to the alveolar tissue, raising its CO_2 partial pressure. Simultaneously, CO_2 is produced in the region of a receptor (CO_2 detector) in the medulla. This CO_2 is similarly diffused and carried to the alveolar tissue through the venous blood stream. It can be shown that the basic controlled variable in this system model is the partial pressure of CO_2 in the receptor tissue located in the medulla.

If CO_2-enriched air is also brought into the lungs, it simultaneously affects the CO_2 diffusion and buildup in the alveolar lung tissue. De Fares et al have shown that the partial pressure of CO_2 in the receptor can serve as an effective mechanism for controlling diffusion of CO_2 from the receptor and from inspired air. In this study, the CO_2 partial pressures of mixed venous blood flow and body tissue will be assumed equal. Similarly, the CO_2 partial pressures of arterial blood flow and alveolar lung tissue will be assumed equal.

By introducing disturbances in the CO_2 content of inspired air, the dynamics of such a control model may be studied. The objective of this model is to hold constant the partial pressure of the CO_2 in the receptor by controlling the diffusion conductance of CO_2 from the receptor area and of the inspired gas to the alveolar lung tissue. Thus, the CO_2 partial pressures of alveolar tissue and local body tissue will respond dynamically to changes in CO_2 content of the inspired air.

Network Model. Because of the dynamic analogies existing between the gas dynamics of the CO_2 diffusion model above and conventional circuit dynamics, it is convenient to represent the biological model by an equivalent circuit model. Figure 10 shows three capacitors tied together with variable nonlinear conductances, which represent the diffusion characteristics of the separate tissue/blood interface. The capacitors represent local tissue CO_2

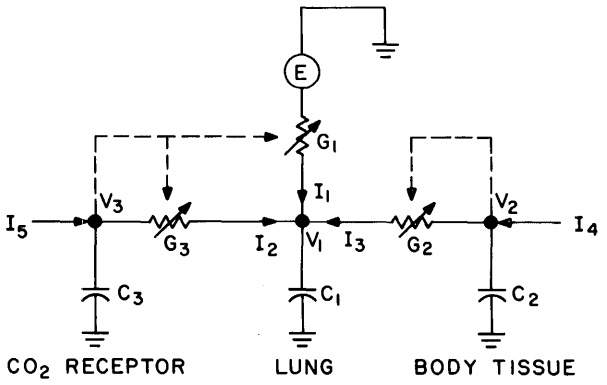


Figure 10. Equivalent network model.

capacity, and the voltages become the respective CO_2 partial pressures. The voltage source E represents the partial pressure of CO_2-enriched inspired air and is defined by the following relation:

$$E = Fi\,(B\text{-}47)$$

$$Fi = \%\ CO_2 \text{ content in inspired air}$$

where B = atmospheric pressure in mm Hg.

Table 3 lists the electrical network parameters and variables together with their physiological equivalents.

Digital-Analog Simulation. As a first example of DSL/90 application flexibility, conventional analog

Table 3. Electrical and Physiological Equivalents, Application No. 1

Elec.	Physiological	
Symbol	Quantity	*Units
G_1	CO_2 conductance-air to lung tissue	Liters (gas)/min/mm Hg (gas)
G_2	CO_2 conductance-body tissue to lung	Liters (CO_2)/min/mm Hg (CO_2)
G_3	CO_2 conductance-receptor to lung	Liters (CO_2)/min/mm Hg (CO_2)
C_1	Capacity of lung tissue	Liters (gas)/mm Hg (gas)
C_2	Capacity of body tissue	Liters (CO_2)/mm Hg (CO_2)
C_3	Capacity of receptor tissue	Liters (CO_2)/mm Hg (CO_2)
V_1	CO_2 partial pressure of lung tissue	mm Hg (CO_2)
V_2	CO_2 partial pressure of body tissue	mm Hg (CO_2)
V_3	CO_2 partial pressure of receptor tissue	mm Hg (CO_2)
E	Partial pressure of CO_2 in inspired air	mm Hg (CO_2)
I_4	Body CO_2 production	Liters (CO_2)/min
I_5	Receptor CO_2 production	Liters (CO_2)/min
I_1	CO_2 diffusion from inspired air to lung tissue	Liters (gas)/min
I_2	CO_2 diffusion from body tissue to lung tissue	Liters (CO_2)/min
I_3	CO_2 diffusion from receptor tissue to lung tissue	Liters (CO_2)/min

*Units are liters BTPS, m.m. Hg, minutes

block notation will be used to program the simulation. Figure 11 represents a DSL/90 digital-analog simulation block diagram of the network model shown in Fig. 10. Since DSL/90 operations are in floating-point arithmetic, no problem scaling is required and the parameters may be entered directly in terms of their conductances are given by the following relations:

$$G_1 = \psi_1 * V_3 - \Theta_1$$
$$G_2 = \psi_2 * V_2 - \Theta_2$$
$$G_3 = \psi_3 * V_3 - \Theta_3$$

where ψ is proportional to the slope of the experimentally determined steady-state cardiac output versus CO_2 partial pressure curves—liters (CO_2)/min/mm^{2}Hg (CO_2); and Θ = initial value of G, liters (CO_2)/min/mm Hg (CO_2).

Using data from respiratory experiments, the following parameters and initial values hold for the simulation:

$V_1(0)$	= 40.0	C_1	= 0.00344
$V_2(0)$	= 45.0	C_2	= 0.17
$V_3(0)$	= 45.0	C_3	= 0.0008
ψ_1	= 0.0038	Θ_1	= 0.1648
ψ_2	= 0.0025	Θ_2	= 0.0625
ψ_3	= 0.0002	Θ_3	= 0.0007
I_4	= 0.25	I_5	= 0.001

The DSL/90 statements which describe this simulator follow.

```
TITLE RESPIRATION SERVO PROBLEM - ANALOG MODE SOLUTION  6-1-65 RUN 1

      EIN=E*(1.-STEP(TDELAY))
      ADR2=EIN-V1
      G1=PSI1*V3-THETA1
      I1=G1*ADR2
      V1=INTGRL(V1IC,(I1+I2+I3)/C1)
      ADR4=V2-V1
      G2=PSI2*V2-THETA2
      I2=G2*ADR4
      V2=INTGRL(V2IC,(I4-I2)/C2)
      ADR7=V3-V1
      G3=PSI3*V3-THETA3
      I3=G3*ADR7
      V3=INTGRL(V3IC,(I5-I3)/C3)                    } Connection Statements

PARAM C1=0.00344, C2=0.17, C3=0.0008,...
      PSI1=0.0038, PSI2=0.0025, PSI3=0.00002,...
      THETA1=0.1648, THETA2=0.0625, THETA3= 0.0007, E=21.4
CONST I4=0.25, I5=0.001, TDELAY=20.0
INCON V1IC=40.0, V2IC=45.0, V3IC=45.0             } Parameters and IC's

CONTRL  FINTIM=36.0, DELT=0.05
RELERR  V1=0.001                                  } Run Control
INTEG MILNE

PRINT   0.1, V1, V2, V3, G1, G2, G3, I1, I2, I3
PREPAR 0.05, V1, V2, V3, G1, G2, G3, I1, I2, I3
GRAPH 6.0, 4.0, TIME, V1, V2, V3
LABEL PAR PRESS  3.0 PRCNT CO2  RUN 1  6-1-65
GRAPH 6.0, 4.0, TIME, G1, G2, G3
LABEL CONDUCTANCE  3.0 PRCNT CO2  RUN 1  6-1-65
GRAPH 6.0, 4.0, TIME, I1, I2, I3
LABEL CO2 DIFFUSION  3.0 PRCNT CO2  RUN 1  6-1-65  } Print and Plot Output

END
STOP
```

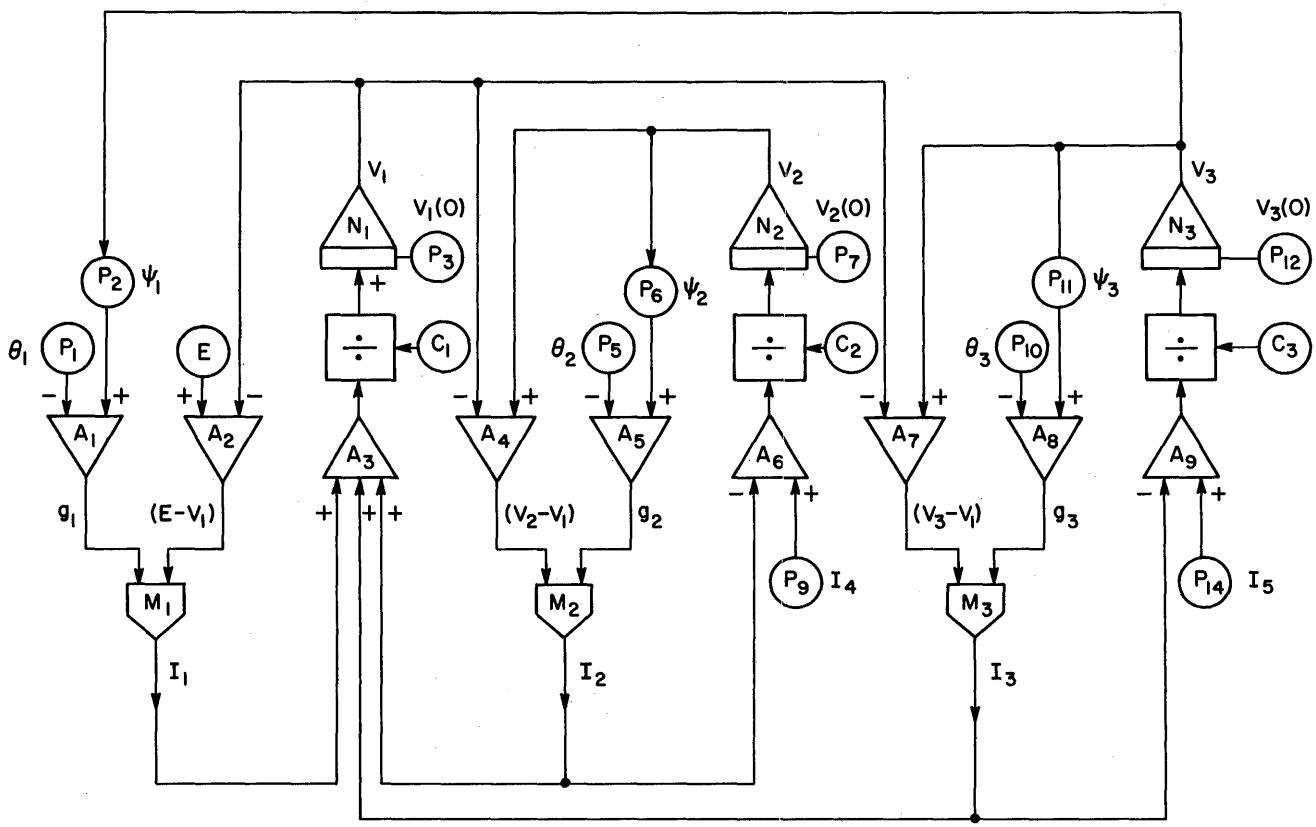


Figure 11. Digital-analog simulator block diagram.

Figures 12 and 13 show nonretouched DSL/90 plots of CO_2 partial pressures and tissue conductances. Inhaled air containing 3% CO_2 was assumed for 20 minutes followed by a 20-minute span of normal room air with no CO_2 content.

During the first 20 minutes, the receptor tissue (medulla), body tissue, and aveolar lung tissue all take up CO_2. The second 20-minute span shows the nonlinear response during purging of body CO_2.

Figure 14 shows part of the results printout and input data format.

After the initial runs were completed, a change in the G_3 conductance characteristic was suggested by medical research personnel. Instead of a linear relationship between G_3 and receptor CO_2 partial pressure, a smoothwise increasing empirical function as shown in Fig. 15 was substituted. To do

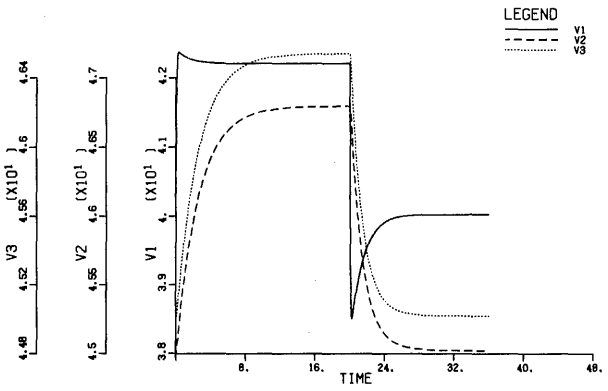


Figure 12. Par press 3.0% CO_2 run 1, 6-1-65.

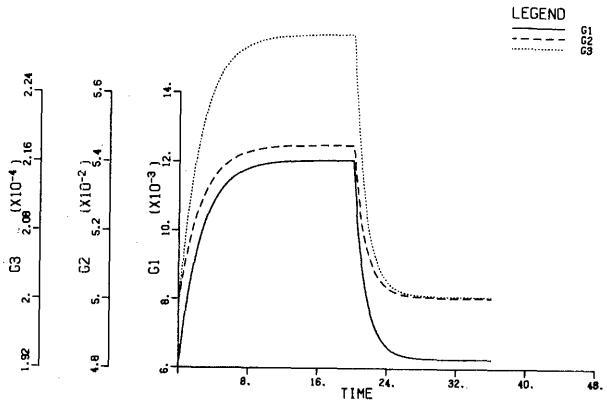


Figure 13. Conductance 3.0% CO_2 run 1, 6-1-65.

```
                *** DSL/90 SIMULATION DATA ***
TITLE RESPIRATORY SERVO PROBLEM - NETWORK MODE SOLUTION  6-1-65 RUN 4
PARAM C1=0.00344, C2=0.17, C3=0.0008,***
      PSI1=0.0038, PSI2=0.0025, PSI3=0.00002,***
      THETA1=0.1648, THETA2=0.0625, THETA3= 0.0007, E=21.4
CONST I4=0.25, I5=0.001, TDELAY=20.0
INCON V1IC=40.0, V2IC=45.0, V3IC=45.0
CONTRL  FINTIM=36.0, DELT=0.05
RELERR  V1=0.001
INTEG MILNE
PRINT   0.1, V1, V2, V3, G1, G2, G3
PREPAR 0.05, V1, V2, V3, G1, G2, G3
GRAPH 6.0, 4.0, TIME, V1, V2, V3
LABEL PAR PRESS  3.0 PRCNT CO2  RUN 4  6-1-65
GRAPH 6.0, 4.0, TIME, G1, G2, G3
LABEL CONDUCTANCE  3.0 PRCNT CO2  RUN 4  6-1-65
END
```

Figure 14a. DSL/90 simulation data.

this, it was necessary to redefine the G_3 conductance characteristic as the output of an arbitrary function generator block as follows:

$$G3 = AFGEN(F3, V3)$$

where the G_3 characteristic is given in a sequence of X and F(X) values.

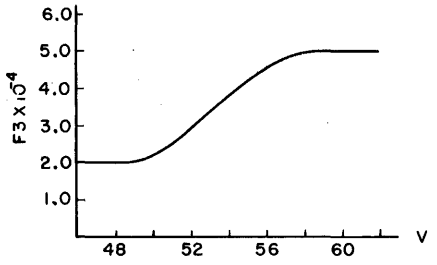
AFGEN F3 = 0.0, .0002, 48., .0002, 49., .00021,... 50., .00023, 51., .00027, 52., .00031, 53., .00035,... 54., .00039, 55., .00043, 56., .000465, 57., .00048,... 58., .00049, .59., .000495, 60., .0005, 80., .0005

In addition to the analog model approach shown here, two other methods were programmed in DSL/90 involving the network equations directly and

TIME	V1	V2	V3	G1	G2	G3
0.	4.0000E 01	4.5000E 01	4.5000E 01	6.2000E-03	5.0000E-02	2.0000E-04
10.000E-02	4.1937E 01	4.5036E 01	4.5030E 01	6.3147E-03	5.0089E-02	2.0060E-04
2.000E-01	4.2319E 01	4.5097E 01	4.5082E 01	6.5132E-03	5.0242E-02	2.0165E-04
3.000E-01	4.2369E 01	4.5162E 01	4.5138E 01	6.7232E-03	5.0404E-02	2.0275E-04
4.000E-01	4.2366E 01	4.5225E 01	4.5192E 01	6.9284E-03	5.0562E-02	2.0383E-04
5.000E-01	4.2359E 01	4.5286E 01	4.5244E 01	7.1262E-03	5.0714E-02	2.0487E-04
6.000E-01	4.2350E 01	4.5344E 01	4.5294E 01	7.3168E-03	5.0861E-02	2.0588E-04
7.000E-01	4.2340E 01	4.5401E 01	4.5342E 01	7.5003E-03	5.1002E-02	2.0684E-04
8.000E-01	4.2331E 01	4.5455E 01	4.5389E 01	7.6769E-03	5.1137E-02	2.0777E-04
9.000E-01	4.2323E 01	4.5507E 01	4.5433E 01	7.8468E-03	5.1267E-02	2.0867E-04
10.000E-01	4.2315E 01	4.5557E 01	4.5476E 01	8.0103E-03	5.1392E-02	2.0953E-04
1.100E 00	4.2309E 01	4.5605E 01	4.5518E 01	8.1676E-03	5.1513E-02	2.1036E-04
1.200E 00	4.2301E 01	4.5651E 01	4.5558E 01	8.3189E-03	5.1628E-02	2.1115E-04
1.300E 00	4.2295E 01	4.5696E 01	4.5596E 01	8.4644E-03	5.1739E-02	2.1192E-04
1.400E 00	4.2289E 01	4.5738E 01	4.5633E 01	8.6043E-03	5.1846E-02	2.1265E-04
1.500E 00	4.2284E 01	4.5779E 01	4.5668E 01	8.7389E-03	5.1949E-02	2.1336E-04
1.600E 00	4.2279E 01	4.5819E 01	4.5702E 01	8.8684E-03	5.2047E-02	2.1404E-04
1.700E 00	4.2274E 01	4.5857E 01	4.5735E 01	8.9928E-03	5.2142E-02	2.1470E-04
1.800E 00	4.2270E 01	4.5893E 01	4.5766E 01	9.1125E-03	5.2233E-02	2.1533E-04
1.900E 00	4.2265E 01	4.5928E 01	4.5797E 01	9.2276E-03	5.2321E-02	2.1593E-04
2.000E 00	4.2262E 01	4.5962E 01	4.5826E 01	9.3382E-03	5.2405E-02	2.1652E-04
[illegible]	4.2258E 01	4.5994E 01	4.58[illegible]	[illegible]5E-03	5.2486E-02	2.1708E-04
[illegible]	[illegible]5E 01	4.6025E 01	[illegible]	[illegible]	5.2563E-02	2.1[illegible]
[illegible]	[illegible]	[illegible]	[illegible]5019E 01	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	01 4.5019E 01	6.2[illegible]	[illegible]	[illegible]0038E-04
3.500E 01	[illegible]	4.5023E 01	4.5019E 01	6.2715E-03	[illegible]2	2.0038E-04
3.510E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2715E-03	5.0057E-02	2.0038E-04
3.520E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2714E-03	5.0057E-02	2.0038E-04
3.530E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2714E-03	5.0057E-02	2.0038E-04
3.540E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2714E-03	5.0057E-02	2.0038E-04
3.550E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2713E-03	5.0057E-02	2.0038E-04
3.560E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2713E-03	5.0057E-02	2.0038E-04
3.570E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2713E-03	5.0057E-02	2.0038E-04
3.580E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2713E-03	5.0057E-02	2.0038E-04
3.590E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2712E-03	5.0057E-02	2.0037E-04
3.600E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2712E-03	5.0057E-02	2.0037E-04

DSL/90 SIMULATION TIME = 13.892 SECONDS

Figure 14b. Respiratory servo problem—network mode solution.



V	F3
0.0	.0002
48	.0002
49	.00021
50	.00023
51	.00027
52	.00031
53	.00035
54	.00039
55	.00043
56	.000465
57	.00048
58	.00049
59	.000495
60	.0005
80	.0005

Figure 15. G_3 conductance characteristic.

fundamental compartment models. This last approach has proven particularly attractive since the biomedical user can directly program his own simulation problem without learning an artifax tool such as analog computer notation, network analysis, or FORTRAN programming. These techniques result in a major reduction in the user time required from initial problem coding to achieving final results. In addition, complete printouts and digital plots are available for each problem run, considerably simplifying the simulation documentation problem.

Application No. 2—Glass Tank Recirculating Furnace

This second example involves the analysis of the heat transfer dynamics of a recirculating furnace used for preheating combustion air on a glass tank. The problem illustrates the ease of using generalized block notation in DSL/90 for performing continuous system simulations. In this case, the example was drawn from the industrial process control field. The technique, however, is broadly applicable to any continuous system analysis problem.

As shown in Fig. 16, air is forced through a large preheating chamber, called a checker, filled with bricks cross-stacked to allow passage of the air around the brick surface, thereby preheating the cold air from the brick. The preheated air is then mixed with fuel, fired, and the resultant flame front melts the glass material in the tank. The hot combustion gases are forced through another checker, heating up the cold brick, and finally forced out the stack. After a period of time, usually about 15 minutes, the flow direction valve is reversed so that the cold checker that had been heated by the hot gases now becomes the preheating checker for the cold incoming air. Similarly, the previous hot checker that had been cooled by the cold input air now receives hot combustion gases which heat it up

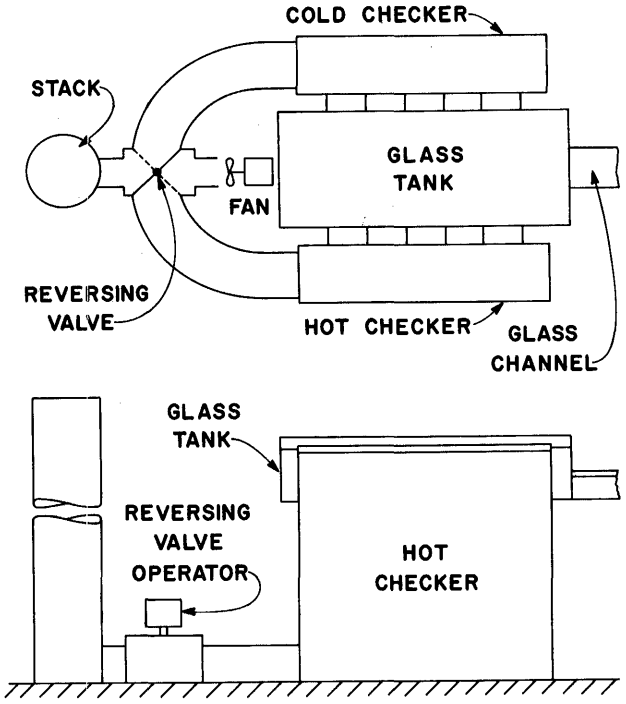


Figure 16. Schematic diagram—reversing furnace.

again. The object of the simulation is to study the heat transfer dynamics of the recirculation furnace during the heating and cooling cycles induced by air flow reversals.

The first step was to divide each checker chamber into three blocks, as shown in Fig. 17, effectively breaking a continuously distributed system into a

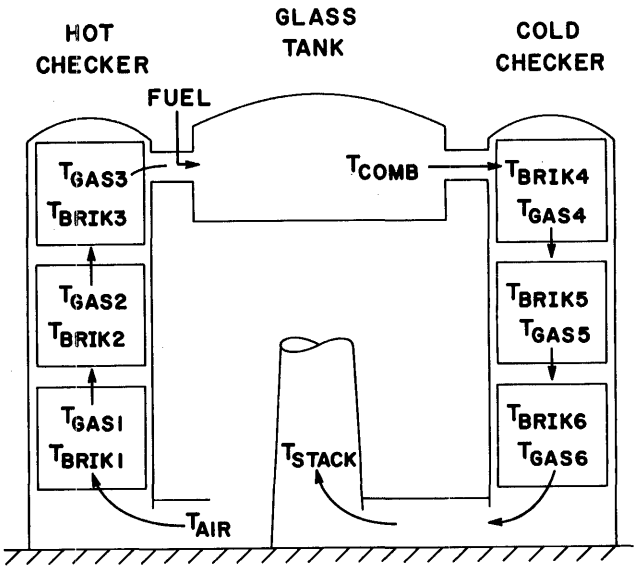


Figure 17. Reversing furnace—end view.

sequence of lumped-parameter segments. The nonlinear heat transfer relationships for each block are given by Eqs. 1 and 2.

$$\frac{d}{dt}\rho_2 V\sigma_A T_{GAS} = \sigma_A F_1 T_{IN} - \sigma_A F_1 T_{GAS} + hA_1(T_{BRICK} - T_{GAS}) + K[(T_{BRICK} + 460)^4 - (T_{GAS} + 460)^4] \quad (1)$$

$$\frac{d}{dt} M\sigma_B T_{BRICK} = hA_2(T_{BRICK} - T_{AMB}) - hA_1(T_{BRICK} - T_{GAS}) - K[(T_{BRICK} + 460)^4 - (T_{GAS} + 460)^4] \quad (2)$$

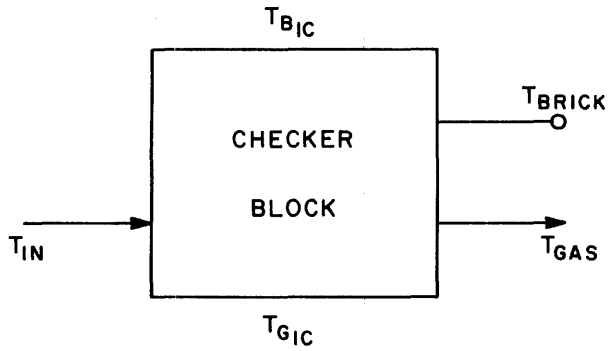


Figure 18. Checker block.

where σ_A = specific heat of the gas,
σ_B = specific heat of the brick,
V = volume of the checker,
M = mass of the checker,
ρ_2 = gas density,
F_1 = gas flow,
A_1 = heat-transfer surface area of brick,
h = conductive heat-transfer coefficient,
K = radiation heat-transfer coefficient,
T_{BRICK} = checker brick temperature,
T_{GAS} = checker gas temperature, and
T_{IN} = input gas temperature to checker.

These differential equations were programmed in FORTRAN and used to define the characteristics of a checker-block, shown in Fig. 18.

The following assumptions and approximations hold for Eqs. 1 and 2.

Assumptions

1. Heat transfer by radiation and convection.
2. Temperature of checker is a function of time and space (1-dimensional).
3. Checker temperature is uniform in any plane perpendicular to flow.
4. Gas temperature is uniform in any plane perpendicular to flow.
5. Brick thermal conductivity is infinite.

Approximation

1. Distributed temperature in each checker is represented by a lumped parameter system of three stages.

The generalized block of Fig. 18 has one input, the entering gas temperature, and two outputs, the exiting gas temperature and the internal brick temperature. Once the block has been programmed and checked out, the user can connect any number of these together to represent the system by simply using the DSL/90 statement:

```
TGAS, TBRIK = CHEKR(TGIC, TBIC, TIN),
```

where TGAS = output gas temperature of checker,
TBRIK = internal brick temperature of checker,
TGIC = initial gas temperature,
TBIC = initial brick temperature, and
TIN = input gas temperature.

Figure 19 shows the block model of one complete checker. Three checker blocks have been used

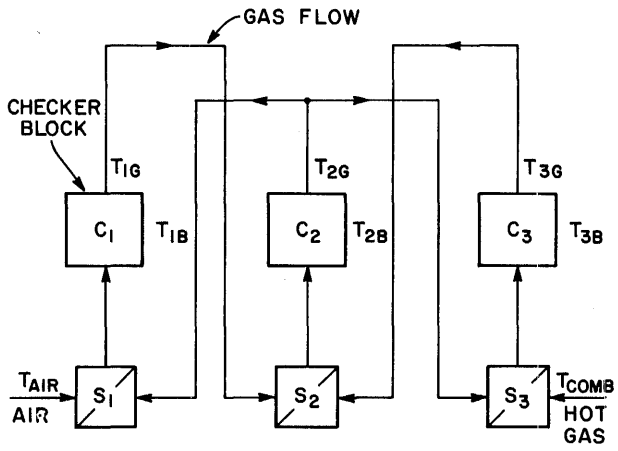


Figure 19. Block model of checker gas flow.

together with three switching blocks that reverse the flow direction through the blocks.

Now if this block model is used as a model of each checker, the DSL/90 statements which represent this system can easily be written by the user

in terms of the basic checker blocks as follows:

```
* ... STRUCTURE STATEMENTS

* ... CHECKER SWITCHES
      C1IN=INSW(TRIGR,TAIR,TGAS2)
      C2IN=INSW(TRIGR,TGAS1,TGAS3)
      C3IN=INSW(TRIGR,TGAS2,TCOMB)
      C4IN=INSW(TRIGR,TCOMB,TGAS5)
      C5IN=INSW(TRIGR,TGAS4,TGAS6)
      C6IN=INSW(TRIGR,TGAS5,TAIR)
      TRIGR=-0.5+STEP(TREVRS)

* ... HOT CHECKER BLOCKS
      TGAS1,TBRIK1=CHEKR(TG1IC,TB1IC,C1IN)
      TGAS2,TBRIK2=CHEKR(TG2IC,TB2IC,C2IN)
      TGAS3,TBRIK3=CHEKR(TG3IC,TB3IC,C3IN)

* ... COLD CHECKER BLOCKS
      TGAS4,TBRIK4=CHEKR(TG4IC,TB4IC,C4IN)
      TGAS5,TBRIK5=CHEKR(TG5IC,TB5IC,C5IN)
      TGAS6,TBRIK6=CHEKR(TG6IC,TB6IC,C6IN)

* ... DATA

PARAM  F1=120000.,  SIGMAA=0.24,  SIGMAB=0.24,  ...
       TAIR=360.,   TCOMB=2800.,  TAMB=120.,    ...
       M=100000.,   A1=15000.,    A2=300.,      ...
       K=4.5E-06,   H=10.,        V=5000.,      ...
       TREVRS=15.

INCON    TG1IC=850.,  TG2IC=1300.,  TG3IC=1800.,  ...
         TB1IC=1600.,  TB2IC=2000.,  TB3IC=2500.,  ...
         TG4IC=2300.,  TG5IC=1900.,  TG6IC=1500.,  ...
         TB4IC=1300.,  TB5IC=1000.,  TB6IC=700.

PRINT    0.1,  TGAS1,  TGAS2,  TGAS3,  TGAS4,  TGAS5,  TGAS6,  ...
         TBRIK1,  TBRIK2,  TBRIK3,  TBRIK4,  TBRIK5,  TBRIK6,  TRIGR,  C1IN

CONTRL  FINTIM=30.,  DELT=0.01

PREPAR 0.05,  TGAS3,  TGAS6,  TBRIK3,  TBRIK6,TGAS1,  TGAS4,  TBRIK1,  TBRIK4
GRAPH  6.0,  4.0,  TIME,  TGAS3,  TBRIK3
LABEL    3RD CHECKER BLOCK TEMPS   RUN 4
GRAPH  6.0,  4.0,  TGAS6,  TBRIK6
LABEL    6TH CHECKER BLOCK TEMPS   RUN 4
END
STOP
```

Note that the parameter and variable names are almost direct symbolic equivalents of the physical notation used for describing the furnace.

Figures 20 and 21 show the actual plotted results of temperature variations at the outlets of the hot and cold checkers for a 15-minute flow reversal cycle. Advantages of this approach in addition to those already mentioned in example no. 1 include the ability to expand the simulation easily to include control system blocks and other system dynamics without disturbing the existing furnace simulation. This feature has proven particularly powerful in analyzing complex industrial processes.

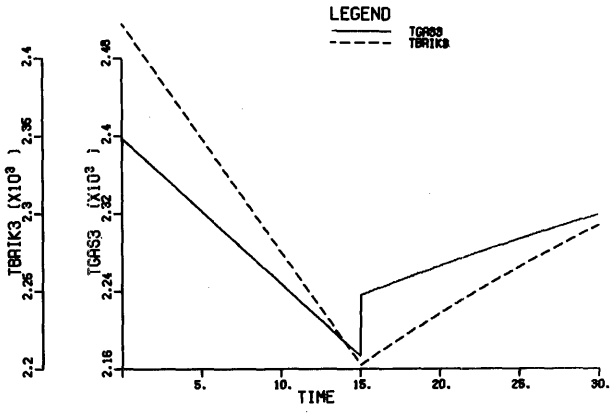


Figure 20. Third checker block temperatures, run 5.

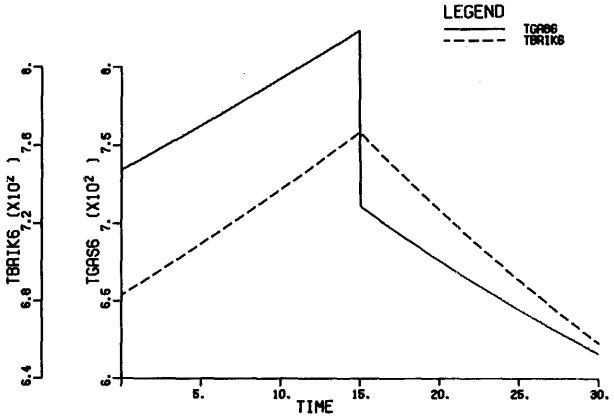


Figure 21. Sixth checker block temperatures, run 5.

Application No. 3—Saturn V Booster Rocket

Vehicle Description. This study applies digital simulation to the flight dynamics analysis of a large space vehicle booster. The problem illustrates the use of DSL/90 algebraic notation statements. In this study, the system example was drawn from the aerospace industry, but the use of DSL/90 algebraic notation can be applied to a broad range of problems including parts of the previous two examples.

The vehicle used in this study was the SATURN V launch vehicle for the APOLLO lunar mission. As shown in Fig. 22, the vehicle configuration consists of three booster stages and the APOLLO spacecraft. The overall length is 360 feet and, fully fueled, the vehicle weighs approximately 6 million pounds. The first, or S-IC, stage is powered by five F-1 engines, each of which provides a thrust of 1.5 million pounds. The four outboard engines are swiveled and provide for thrust vector control during powered flight. The SATURN V vehicle has an independent inertial navigation and guidance system from that in the APOLLO spacecraft in addition to a control computer and required sensors.

Trajectory. This simulation is concerned with the analysis of flight dynamics from launch through first-stage burnout. The booster-stage flight profile is shown in Fig. 22 and consists of a gravity turn for 150 seconds with separation occurring at approximately 60,000 meters altitude and a 2350-m/sec velocity. The rigid body equations of motion that were simulated form a perturbation set with respect to a reference frame moving along the nominal trajectory as shown in Fig. 23.

Axes X_1, X_2 X_3 form an orthogonal set, with X_2 aligned along the nominal velocity vector and axes X_1, X_2 lying in the nominal boost plane. The fuel sloshing dynamics of the first stage propellants were

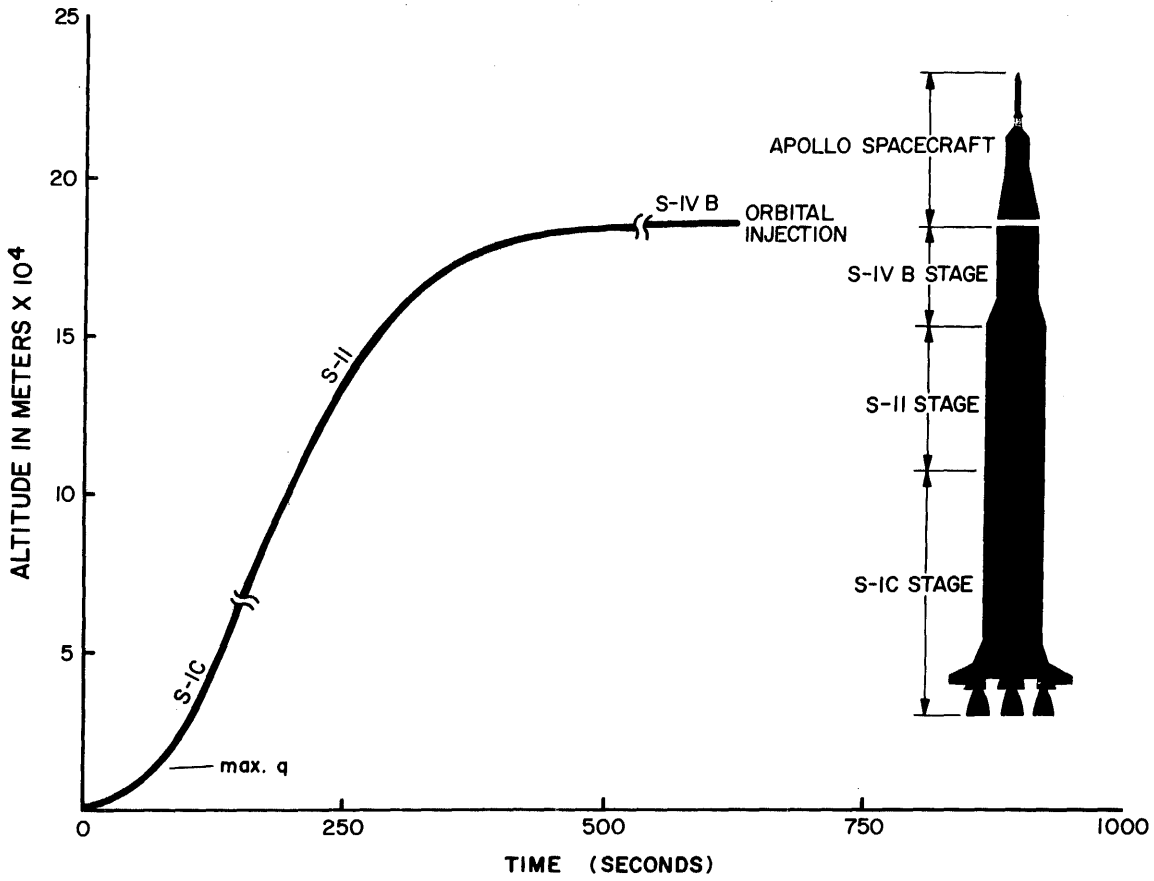


Figure 22. SATURN V configuration and flight profile (from Ref. 5).

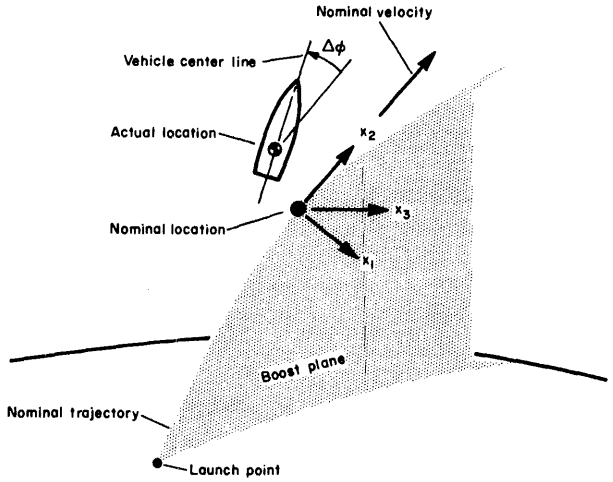


Figure 23. Reference frame axes (from Ref. 6).

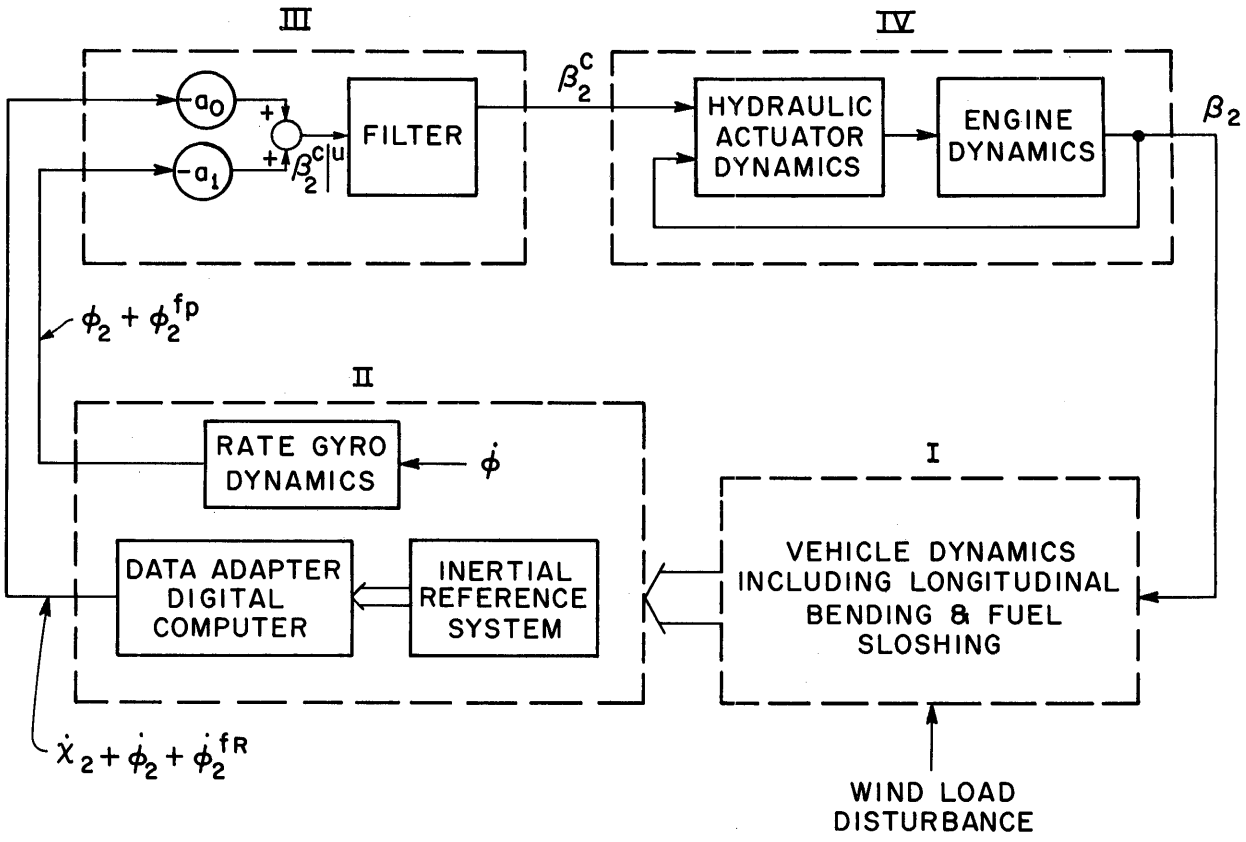
included as well as the dynamic effects of elastic bending along the booster longitudinal axis. The attitude control system was also included in the simulation, together with the dynamics of the gimballed thrust VECTOR control system and hydraulic actuators for the engines, as shown in Fig. 24.

Since the defining equations of vehicle motion are far too complex for the purposes of this paper, the reader is referred to the basic documentation for the complete problem description. To illustrate the features of DSL/90, only a small portion of the larger problem will be treated—the pitch axis control system. Figure 25 is an expanded description of the control system filters, together with actuator and engine dynamics. The command signal filter block processes the pitch command signal from the control computer prior to applying it to the engine gimbal hydraulic actuators.

In order to investigate booster flight dynamics, a primary wind disturbance was applied to the vehicle during the first stage of powered flight as shown in Fig. 26. Horizontal wind loading was assumed, with varying azimuth angles for wind heading.

Referring to Fig. 25, the transfer functions for the command signal filter and engine dynamics can be expended in Laplace notation to yield the equivalent linear operational equations:

$$S^2\beta_2^c = K_1\beta_2^c|^u + K_2 S\beta_2^c + K_3\beta_2^c \qquad (3)$$



$\dot{\chi}_2$ NOMINAL PITCH RATE - DEG/SEC

$\dot{\phi}_2$ PERTUBATION IN RIGID BODY PITCH RATE-DEG/SEC

$\dot{\phi}_2^{fR}$ PITCH RATE DUE TO VEHICLE FLEXING MEASURED AT THE RATE GYRO STATION - DEG/SEC

ϕ_2 PERTUBATION IN PITCH ATTITUDE - DEGREES

ϕ_2^{fp} ATTITUDE DUE TO VEHICLE FLEXING MEASURED AT THE STABLE PLATFORM STATION -DEG/SEC

$\beta_2^{c|u}$ β_2 PITCH ATTITUDE COMMAND, UNFILTERED

β_2^c β PITCH ATTITUDE COMMAND, FILTERED

β_2 ENGINE GIMBAL ANGLE (PITCH AXIS)-DEGREES

Figure 24. Simulation signal flow diagram (from Ref. 5).

and

$$S^2\beta_2 = (K_{32}\beta_2^c - K_{31}\beta_2)S + (K_{34}\beta_2^c - K_{33}\beta_2) + (K_{36}\beta_2 - K_{35}\beta_2)\frac{1}{S} \quad (4)$$

where S is the conventional Laplace operator.

From Fig. 24, the expression for the unfiltered pitch command signal $\beta_2^c|^u$ becomes:

$$\beta_2^c|^u = -[\alpha_0(\phi_2 + \phi_2^{fp}) + \alpha_1(\dot{\chi} + \dot{\phi}_2 + \dot{\phi}_2^{fr})] \quad (5)$$

Equations (3) through (5) can be directly programmed as DSL/90 statements as follows:

```
*  PITCH ATTITUDE CONTROL SECTION
   BET2CU = -(AO*(PH12+PH12FP)
            +A1*(CH12D+PH12D
            +PH2DFR))
   BET2CD = INTGRL(B2CDO, K1*BET2CU
            +K2*BET2CD+K3*BET2C)
```

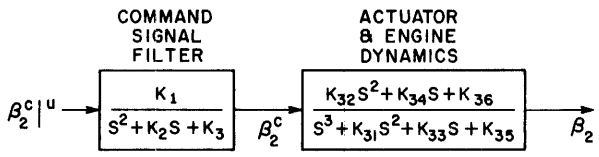


Figure 25. Pitch axes control system.

```
BET2C = INTGRL(BET2CO, BET2CD)
BET2DD = K32*BET2CD – K31*BET2D
         + K34*BET2C...
         – K33*BET2 + INTGRL(IC53,
         K36*BET2C – K35*BET2)
BET2D = INTGRL(BET2DO, BET2DD)
BET2 = INTGRL(BET2O, BET2D)
```

For the complete simulation, over 400 DSL/90 statements were required, not including the function generators and data statements. Both block and algebraic notation were used for describing the simulation configuration. The above small portion of problem coding is an excellent example of the ease of using both algebraic and block statements in DSL/90. Note the use of symbolic names for variable and data names which closely resemble the actual names. This feature has proven particularly helpful for large simulations.

The SATURN V flight dynamics were simulated for the first 120 seconds of powered flight. Figures 27, 28 and 29 show resultant DSL/90 plots for three of the system variables being studied.

The SATURN V simulation demonstrated several important features of digital simulation. First, a complex nonlinear aerospace problem could be successfully solved in DSL/90 by engineers relatively unskilled in programming. Second, many problems require both algebraic and block notation. The ability of DSL/90 to handle both of these requirements was amply proved. Third, problem solutions could be obtained quickly with a minimum of setup time. The original programming required approximately 16 hours of an engineer's time for problem setup. Each run of 120 seconds flight time required approximately 25 minutes of IBM 7094 computer time. In addition to the above features, DSL/90 allowed the user to model his problem in segments, checking out portions of the simulated vehicle independently, and then to hook these sections together. As an example, the trajectory equations form one section of the simulation, programmed in algebraic notation, of which the control system is another independent part programmed in block notation.

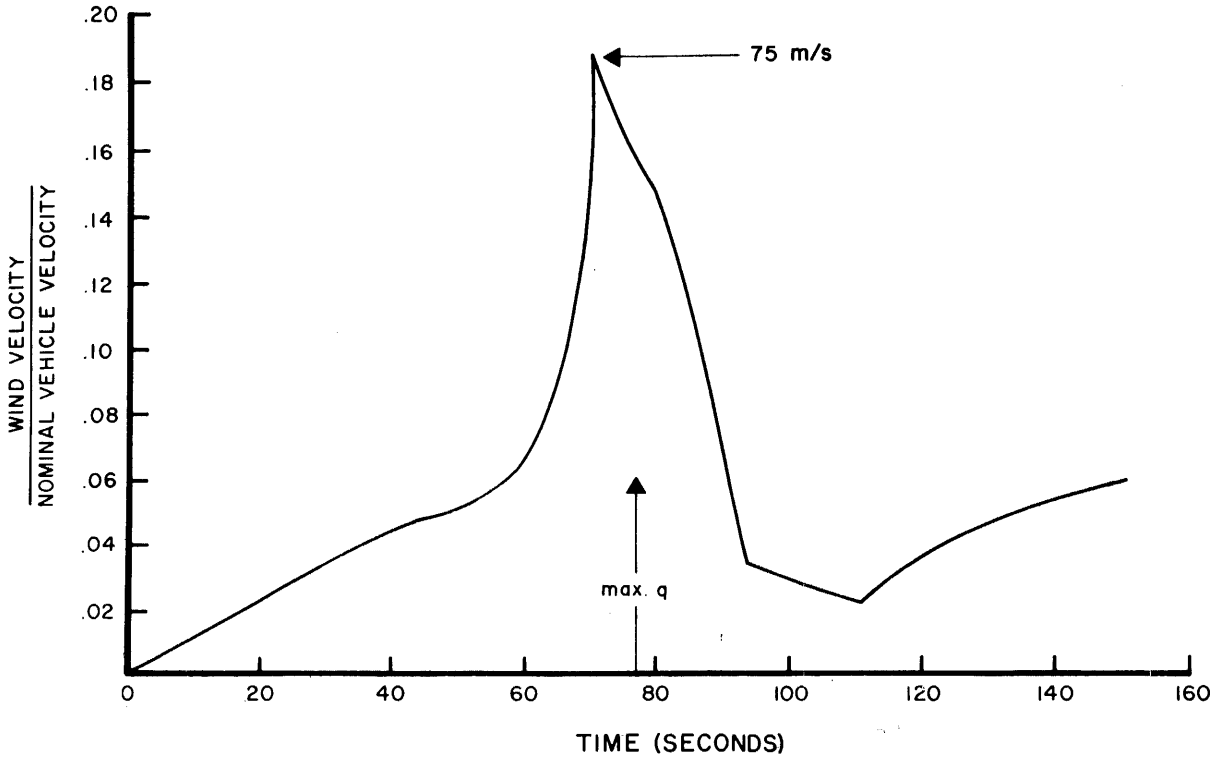


Figure 26. Primary wind disturbance (from Ref. 5).

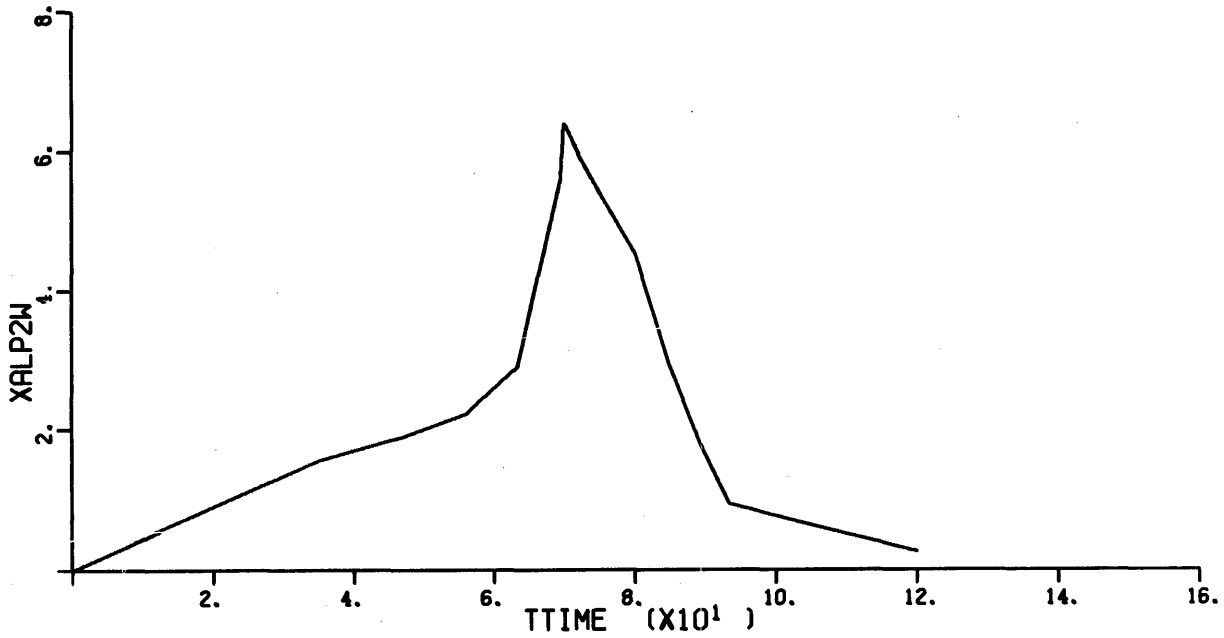


Figure 27. Pitch axis angular acceleration.

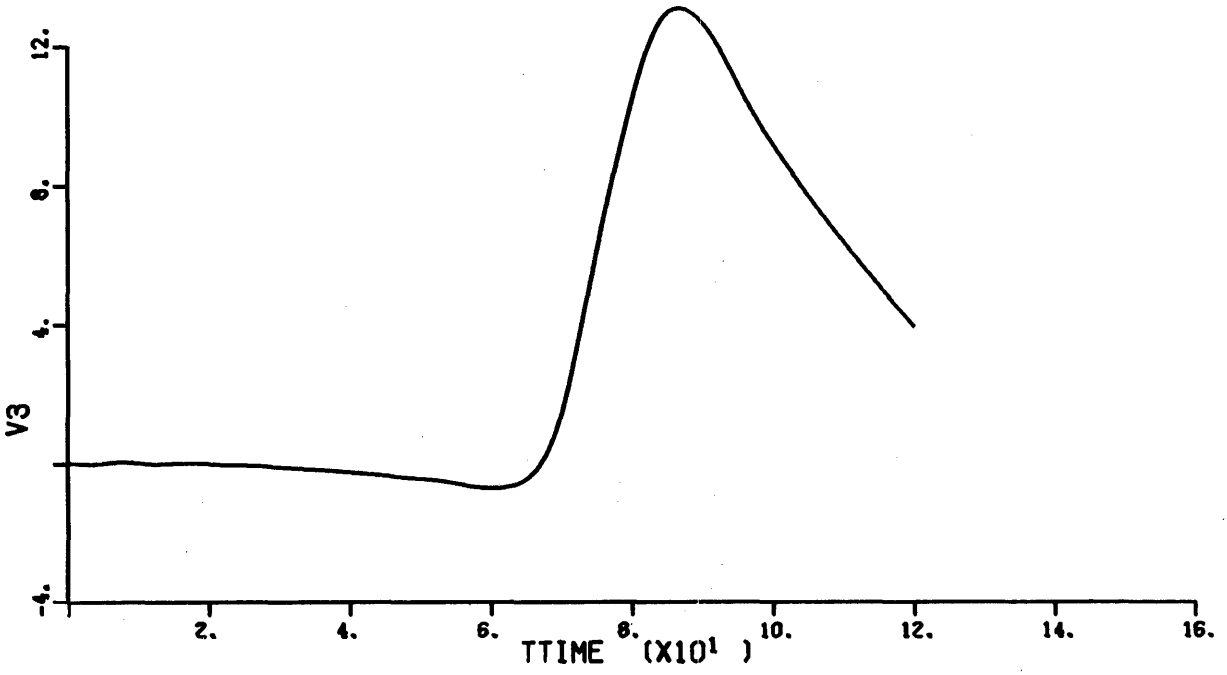


Figure 28. Velocity along X_3 axis.

CONCLUSIONS

Within IBM, DSL/90 has been used extensively in many different application areas including circuit design, mechanical dynamics, process analysis and control, servo design, aerospace flight simulation and biomedical modeling. Simplicity of the input language, clarity and completeness of both print and plot output, and the ease with which data is handled are some of the features which have made DSL/90 attractive to an increasing number of problem solvers from both camps—analog and digital. In DSL/90 workshops, it was observed that engineers with hardly any analog or digital computer ex-

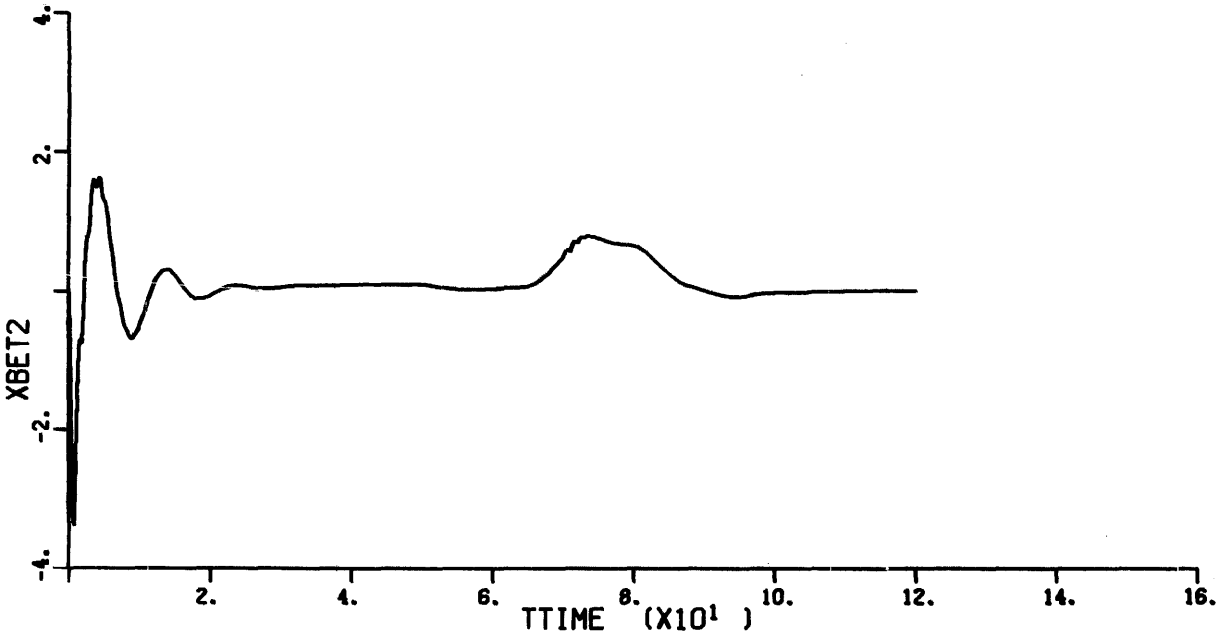


Figure 29. Engine gimbal angle for pitch axis.

perience successfully "programmed" in DSL/90 at the end of the first two-hour session. With this quick "shot" of confidence and further experience, many have proceeded to more difficult problems using the more advanced features of the language.

The examples shown indicate only a few of the broad range of problem areas to which DSL/90 can be applied. In addition to the above examples, DSL/90 has successfully simulated the process dynamics and control system responses for a paper machine dryer section control system. In this study, actual process noise gathered at the plant site was introduced into the simulation through the MAIN routine. Several nonlinear process and control elements were successfully modeled using the external block features of DSL/90, including nonlinear process controllers and scanning moisture gauges. DSL/90 was recently used for the simulation of an ammonia reaction process involving two-point boundary value matching. In this case, severe simulation problems were created by the fact that the system had two regions of time response, each governed by different differential equations and interfacing through initial values. Both the features of the "MAIN" program and the ability to introduce logical functions into the DSL/90 block structure were extensively employed.

Many of these simulation areas previously handled with analog techniques have long been troubled with problems of component reliability, accuracy, repeatability, and a lack of flexibility in modeling basic dynamic components and phenomena. In some respects, the trend toward digital simulation methods is a result of seeking answers to these problems. Some of the advantages of digital simulation as observed in the above application studies can be listed as follows:

1. Problem accuracy control.
2. Elimination of problem scaling.
3. Simulation run repeatability.
4. Reliable digital simulation elements.
5. Significantly reduced problem preparation time and simulation checkout time.
6. Simple problem coding. The majority of detailed circuit knowledge for analog programming is unnecessary.
7. Easy performance by the digital computer of some operations which at best are only approximated by analog computers.
8. Effortless provision of positive documentation of simulation configuration and parameter values.

To date, digital simulation techniques have shown themselves easy to learn, efficient to operate, accurate, and extremely flexible. They provide the engineer with an easy and quick method of digitally simulating complex systems, familiar block notation concepts, and the power of digital computation

methods. The result represents a significant new simulation tool for engineering analysis and design.

ACKNOWLEDGMENTS

To our co-worker in this project, Mr. D. G. Wyman, we gratefully acknowledge his excellent contributions in both programming and concepts. We have benefited greatly from the many valuable suggestions of members of the computation laboratory, Systems Development Division, IBM, San Jose. Special thanks are due to Mr. A. H. Hoffman whose contributions to the exploratory program PLIANT paved the way for DSL/90.

The contributions of J. G. DeFares, P. E. Cowley, and F. Crane to the three application examples are particularly acknowledged.

BIBLIOGRAPHY

1. Brennan, R. D., and R. N. Linebarger, "A Survey of Digital Simulation: Digital-Analog Simulator Programs," *Simulation*, vol. 3, no. 6, pp. 23–36 (Dec. 1964).

2. ____, "A Survey of Digital Simulation: Part II—More Digital Analog Simulator Programs," ibid (to be published).

3. Dahlin, E. B., and R. N. Linebarger, "Digital Simulation Applied to Paper Machine Dryer Studies," *Proceedings*, Instrument Society of America, 6th International Pulp and Paper Instrumentation Symposium, Green Bay, Wisconsin, May 4–8, 1965.

4. DeFares, J. G., H. Hara, and E. M. Billinghurst, "The Stability of the Respiratory Servomechanism: An Analog Computer Study," *Progress in Biocybernetics*, Elsevier Publishing Company, New York, 1964, vol. 1.

5. Gunderson, R. W., and G. H. Hardy, "Piloted Guidance and Control of the SATURN V Launch Vehicle," *Proceedings*, IFAC Symposium on the Peaceful Uses of Space, Stavenger, Norway, June 1965.

6. Hardy, G. H., J. V. West and R. W. Gunderson, "Evaluation of Pilot's Ability to Stabilize a Flexible Launch Vehicle During First Stage Boost," Technical Note D-2807, National Aeronautics and Space Administration, Washington, D. C. (May 1965).

7. Shah, M. J., C. James and J. M. Duffin, "Simulation of an Ammonia Synthesis Reactor," *1966 Conference Proceedings*, International Federation of Automatic Control, London.

8. Wegstein, J., "Accelerating Convergence of Iterative Processes," National Bureau of Standards, Washington, D. C.

TECHNIQUES FOR REPLACING CHARACTERS THAT ARE GARBLED ON INPUT

Gary Carlson
Computer Research Center, Brigham Young University
Provo, Utah

With the rapid increase in the availability of mass storage, we now find that there is an increasing need for massive data input. This input is requiring increasingly large volumes of data conversion to machine language. As this load expands, we find that we must pay more and more attention to the rigorous control of errors on data creation. This study reports the results of a computer technique to reduce errors of input data.

The standard techniques of key stroke and verify, or double punch and compare, are often prohibitively expensive for very large file conversion. Anyone faced with the problem of large file conversion must consider the possibility of using optical scanning equipment. We are concerned with a potentially very large file that is in nonmachine language and should be converted. We sought a better way to convert and then correct, or proofread, or verify the data. By a "better way" we mean that we sought a technique that would give acceptable accuracy at a cost less than other techniques of conversion.

Any human involvement in the proofreading or verifying phase has a relatively high cost and still a moderate error. We wondered if we could use the computer to reduce the errors on input data.

The basic nature of our records is genealogical, that is our records contain *names of people*, *dates*, and *places*. For the conversion to machine language of the millions of records that we are concerned with, we are considering the possibility of using optical scanning equipment on the existing, typewritten, multi-font documents. However, most scanning equipment still seems to have some 2 to 5% character error rate. Fortunately, this equipment can usually indicate confusion on a given character which is not decisively read. In other words, the scanning equipment can say that it recognizes the first, third, and fourth characters but got confused on the second character. The errors of ambiguity offer the possibility of direct computer correction.

The problem then is, given that a character is garbled, can we effectively replace the character. Later on, we hope to work on places or place names, and even dates. But to begin our study, we started on the materials available and of most interest—English names. Specifically, then, the object of the study is to determine if we can replace garbled characters in names.

The basic plan was to develop the empirical frequency of occurrence of sets of characters in names and use these statistics to replace a missing character. I am happy to report that we have developed such techniques—and in most cases a garbled character can be replaced with better than 90% accuracy.

The basic technique was to develop programs to compile the required statistics, and then other programs to replace the character in question. All programs are written in COBOL, and have been run on the IBM 7040 tape system.

The frequency of occurrence of trigrams (that is, a

string of three sequential characters) was first developed for a sample of 73,000 christening records from English parishes.

Trigrams were used as the most parsimonious solution. Single-character probabilities were tried and gave very poor replacement. Two-character strings also gave poor results. In some cases it may be necessary to consider four-character strings. Trigrams give generally good results with reasonable processing times.

We worked first with all names, that is, given and surnames combined, over 300,000 names. Approximately 78% correct character replacements were achieved with this set of data. We then developed separate trigrams for male given names, female given names, and then for surnames. This refinement greatly improved the accuracy of replacement.

Additional information was obtained and used. This includes the location within the name of the first character of the trigram, e.g., the trigram MAR from the name MARY has a position of 1. The trigram ARY from MARY has a position of 2. A final refinement that proved very useful was an indication of the trigram being at the END or NOT END of a name. The trigram ARY from MARY is an END trigram.

We separated the data into two halves, using the first half to develop the probabilities and basic data, and then tested the procedures against the first half and then against the second half. Our techniques systematically delete the characters one at a time in each name and then replace the character using the trigram statistics. The basis for the choice to replace the missing character was the character from the trigrams that had the highest probability of occurrence. The percentage of correct replacements is the measure of success. For example see Table 1.

Here we see that the distribution of trigrams for the first three characters MA? clearly show MA*R* as the most probable. In this particular case, if the trigram is shifted to cover the next character position, A?Y, it is again seen that A*R*Y is by far the most probable. By combining these two, the choice for the missing character is the letter "R." The highly probable A*R*Y trigram is also an END trigram, indicating that the name is now complete.

The second example of MA?GARET uses the first MA? as well as the two shown. In this case, the probabilities do not appear to be quite as striking, but nevertheless are clearly decisive to select "R" as the missing character. In fact, in this particular case, "R" is the only possible character when all three trigrams are considered. For example, starting with the first trigram set, there are several possibilities:

Table 1.

Example—MA?Y

MA?			A?Y		
3rd Position Trigram		Frequency of Occurrence	2nd Position Trigram		Frequency of Occurrence
MA B		5	A B Y		1
MA G		5	—		—
MA L		5	A L Y		4
—		—	A M Y		46
MA R	NE*	8,316	A R Y	End	6,466
MA R	End	5	A R Y	NE*	20
MA S		22	—		—
MA T		12	—		—
MA U		2	—		—
MA W		1	—		—
MA Y		4	—		—

Example—MA?GARET

A?G		?GA	
2nd Position Trigram	Frequency of Occurrence	1st Position Trigram	Frequency of Occurrence
A G G	6	—	—
—	—	I GA	59
A N G	2	—	—
A R G	802	R GA	616
A U G	1	—	—

*NE means the trigram was not at the end of the name.

MA*G* — A*GG*
MA*R* — A*RG*
MA*U* — A*UG*

When the third trigram set is introduced, only MA*R* – A*RG* and *R*GA remain. This means that MA*R*GA must be the sequence of letters that was seen by the scanner.

The combined, or joint probability is similar to a precoordinated index, where only those items containing a complete set of *AND* logical relations are accepted. Such a technique will make some mistakes for rare sequences of letters.

What results have been achieved? Let us first examine in detail the first character position of the male given, female given, and surname (Table 2). Here we see that position 1 or the first character of names is, of course, the hardest one to replace correctly since there is only a blank preceding the character in question. At best, a trigram arrangement uses information based only on the two characters

Table 2.

	Position 1			
	Tape 1		*Tape 2*	
	Total Processed	%	Total Processed	%
Male given name	51,629	94.24	51,629	94.23
Female given name	32,730	83.05	32,730	83.05
Surname	73,684	42.24	(not yet available)	

following the first character. In Table 2, the numbers mean that there were 51,629 male given names used on tape 1 to develop the basic frequencies of occurrence. Then, using these as probabilities, we correctly replaced 94.24% of the first characters that were deleted in male given names. We then used the same probabilities and the same COBOL routine to process the second set of names, which were *not* used to develop the basic frequencies of occurrence, and came up with the almost identical percentage correct replacement. These numbers are not a typing error, these just happened in this particular position to come out identical for both tape 1 (used to develop the probabilities) and tape 2 (the new data) for each set of male and female given names. So you can see that we were developing the trigrams here on 51,629 names on the male, 32,730 names for the female, and 73,684 surnames. We are processing a large batch of names, developing the frequency of occurrence of trigrams, then using this as a basis to predict the character that is missing in subsequent data.

As we move away from the first character position into the body of the name, it is possible to use three trigrams as is shown in Table 1 to make the computer estimate of what the character should be. As the processing moves further into the name, we find better results, and then taper off to a less striking correction possibility at the end of the name. It should be noted that in addition to recording the trigram frequency, we recorded a separate condition within each category of male, female, and surname that indicates whether the trigram was an end or non-end trigram. The overall results are given in Table 3.

This table indicates that we often get better than 98% correct replacement of the garbled character. The specific effect on error reduction is impressive. If a scanner gives a 5% character error rate, the trigram replacement technique can correct approximately 95% of these read errors. The remaining error is thus 5% of the original 5%, or 0.25% overall. Such a low error rate is a fond hope of the very best verification procedures.

Table 3. Percent Correct Character Replacement

Position of Character	Male Given		Female Given		Surname
	Tape 1	*Tape 1*	*Tape 1*	*Tape 2*	*Tape 1*
1	94.24	94.23	83.05	83.05	42.24
2	99.56	99.45	99.29	99.18	73.23
3	99.33	99.14	99.74	99.56	74.15
4	99.45	99.25	99.14	97.99	72.66
5	99.58	99.32	98.51	98.19	79.85
6	99.11	98.94	95.63	95.32	81.05
7	98.06	97.88	98.84	98.60	84.06
8	99.03	98.28	98.97	98.82	87.99
9	99.39	98.16	98.96	98.89	89.74
10	99.54	98.16	86.56	83.07	92.12
11	100.00	99.46	100.00	88.88	92.21
12	—	—	100.00	87.50	96.77
13	—	—	—	—	99.00
14	—	—	—	—	100.00
15	—	—	—	—	100.00

Preliminary results on given names from a completely different set of data of English Parish registers gives 96.6% correct character replacement. This result indicates that the techniques are consistent for comparable data. Such results imply that modern names can also be "corrected" if we develop a new set of probabilities for modern names.

What are the implications of this study? We find that there are at least three implications. One, a technique like this may, indeed, reduce the cost of verifying the mass of data input coming from scanners. Two, techniques like this may also reduce the cost of verifying massive data conversion coming from conventional data input devices like keyboards, remote terminals, etc. Three, techniques like this may be effective in other areas of linguistic manipulation, such as newspaper proofreading, or may even be developed to locate the error and then make the correction.

What does this mean for the future, and what future research do we see that should follow from the work done to date? Some future research is indicated:

1. We should try to replace the characters in another data base using our existing trigram frequencies. Preliminary work on modern American given names, *using the English name probabilities,* indicates a 30 to 50% correct character replacement, about 20 to 30% wrong replacement, and *no* replacement indicated in the remaining 30 to 40%. This last result came as a surprise, but seems to be holding. The implication is that if a different data base is used on a set of data, the present procedures give no basis for a decision in a large number of

cases. This no-decision is not as good as a correct decision, but is also decidedly better than a wrong replacement. Work is continuing to explore the ramifications of homogeneity of trigram development data and garbled data to be corrected.

2. We must experiment with other than *trig*rams, especially for the first and last part of names or words. For example, we may have to go to a four-gram or more at the end of words or names. This work is currently in process also.

3. We need to try another body of words, other than English names. This might be newspaper or magazine material to see if the technique could be applicable there. This is yet to be done.

4. Techniques must be undertaken to speed up the processing on the computer. These routines are fairly fast now, but with a tape system there is an excessive amount of sorting.

5. There are other techniques that seem to be indicated from the data analysis. It is not clear that you should always take the most probable character as your choice. In fact, it may be that if the distribution of choices is a rectangular distribution, some heuristic type choice mechanism may give better overall results than a straight maximum probability choice. We have begun work on this, but the results are not yet conclusive.

6. Definite procedures should be developed to locate probable errors of character sequence. These routines could be of value where scanning equipment was not used, and thus the location of the error is not immediately obvious. This is yet to be done.

It appears that this kind of analysis can be of interest in reducing the cost of massive data conversion.

ADAM—A GENERALIZED DATA MANAGEMENT SYSTEM*

Thomas L. Connors
The MITRE Corporation, Bedford, Massachusetts

INTRODUCTION

ADAM is a computer program system built by The MITRE Corporation for use in the MITRE/ESD Systems Design Laboratory as a tool to aid the design and evaluation of data management systems. ADAM operates on the IBM 7030 (STRETCH) computer and has been operational since early 1965.

Primarily, ADAM is a tool with which a system designer can simulate alternative proposed designs for his data management system. When ADAM is run, the computer and its associated displays, typewriters, and other peripheral equipment become an operating mock-up of the proposed system, with simulated or real users working on-line with the model. System designers may test, evaluate, change, and retest the model without the usual reprogramming costs associated with changing a design already embodied in a large computer program. The revised system models actually operate in real time.

A secondary purpose is to provide a test-bed in which techniques of large-system design and programming can be implemented, compared, and evaluated. To this end, partial models and new programs may be added to the ADAM framework and tests run on them.

This paper describes some of the things ADAM can do, mentions some of the problems to which it has been applied, and conclusions reached, and gives a few of the internal details of its operation. It is intended as a description of one of the operating resources of a laboratory devoted to experimentation in the design of information systems and as the germ of a concept for a way to build large systems.

FUNCTIONS

To accomplish its purpose, ADAM incorporates generalized facilities for performing those functions which characterize data management systems:

1. File generation and maintenance.
2. Translation and processing of queries and file processing messages.
3. On-line and off-line input and output.
4. Report generation and formatting.
5. Compilation and execution of subroutines.
6. Dynamic allocation of computer resources.

These facilities are generalized in that the programs of ADAM operate independently of the form, format, or size of the data, of the input message language, and of the report formats required. Specifications of the particular characteristics of a model reside in the data-base files or dictionaries along with the problem-data itself. Thus, the definitions of the data-base structure, message-languages, output-formats and problem-specific processes become data, subject to modification and update with system procedures.

*The material contained in this article has been approved for public dissemination under Contract AF19(628)-5165.

The user of ADAM specifies his data-base structure—how the data is to be organized, named and interrelated—in file generation statements. He opts either to use available ADAM message languages and report formats or to specify his own. He programs whatever subroutines are required to accomplish complex or time-consuming calculations not suitable for specification in message languages. He adds this material to the generalized foundation which is the ADAM system, and is then ready to exercise and evaluate his model. To change his model, he need change only those components he has specified; these are only a small part of all the material of which the model is composed. This generalization of common functions is the essence of ADAM and leads to:

- Its versatility—its ability to accept specifications for many different data-management systems.
- Its changeability—the ability to change data structure, report formats, etc., without reprogramming, in a short time and in many cases on-line.
- The wide range of its capabilities—input/output which includes on-line remote inputs is one example; file processing which includes complex data indexing is another.

DESCRIPTION

The following material is a general description of ADAM—its operation, data structures, file processing, and provisions for user subroutines and format descriptions. A system as large as ADAM defies description in full detail—many capabilities (such as remote operation) can be only mentioned, others not treated at all; however a summary of ADAM capabilities appears below.

Operation

The operating ADAM system accepts messages from and sends output to on-line input/output devices, such as typewriters and display consoles, or operates with either off-line input, off-line output, or both. Messages may query the data contained in ADAM: add, change, or delete data; or cause the operation of ADAM or user programs.

All data in ADAM are contained in *files*; creation, maintenance, processing, and querying the files are major functions of the system. In addition to files of problem data, the system keeps files of subroutines (both problem-specific subroutines and the subroutines which constitute ADAM itself), language specifications, format specifications, and various other special-purpose files. Data for reports are initially made up as an output file, to be formatted before actual output.

The file structure and data formats are elaborately described by a set of dictionaries *cum* directories called by the ADAM term *rolls*. As a dictionary, a roll stores alphanumeric names of files, entries in files, data items, routines, formats, and so forth. As a directory, a roll stores the dynamically changing information which describes: the current physical location in the computer, the format, and the size of elements of the system—files, entries, data items, etc.

ADAM message processing is shown in Figs. 1 and 2. As each message is received, the system is interrupted to determine its priority, to determine the language in which it is written, and to stack the message.

Major message processing programs are the Translator, Processor and Output Generator. When a message is unstacked, the Translator (with reference to the appropriate language specification from the Language File and to the rolls) translates the message into a list of things to be done, called a *process table*. The Processor executes the steps specified in the process table interpretively, accessing or modifying the data base and operating any routines specified. If output is produced, it is always in the form of a file. The Output Generator performs any formatting required on the data from this file, as described by a format from the format file, then delivers the output to be stacked, pending availability of an output device. When processing of one message is finished, the next is unstacked.

(With "time-sharing" so much in the vogue, it

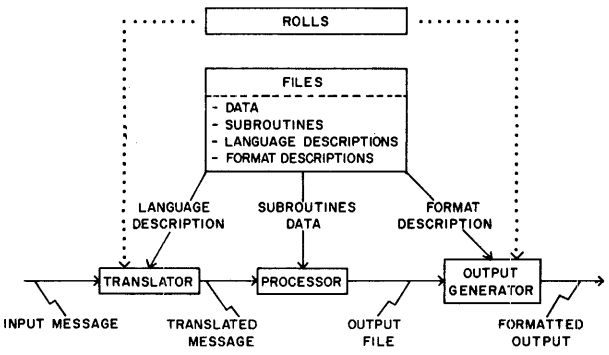


Figure 1. Basic execution cycle.

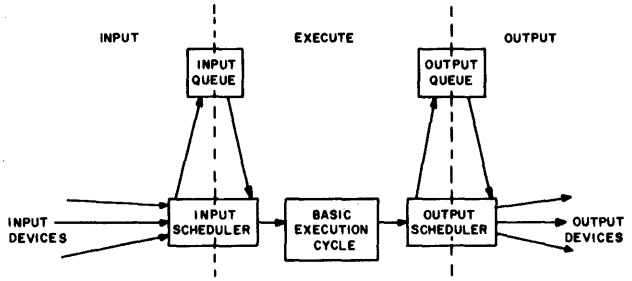


Figure 2. Input—Execute—Output.

cannot be ignored in the description of an on-line system. ADAM is an asynchronous system, but it does not swap tasks in and out of memory. Inputs come in at any time and are stacked, and outputs go out as fast as the equipment will take them. Both input and output are asynchronous with the basic execution cycle which, when it begins a task, runs the task to completion except for interruptions to receive inputs and send out outputs.)

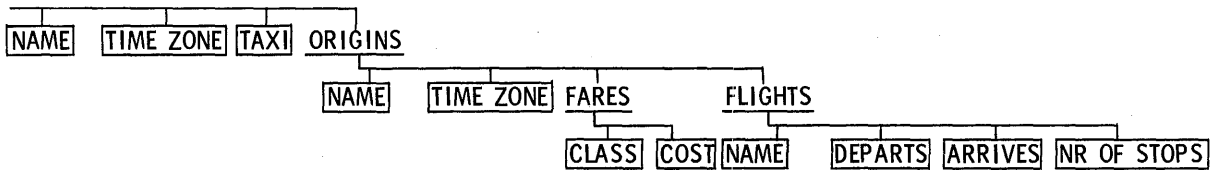
File Structure

A file is organized as a series of entries, each of which has an identical structure, i.e., all entries in the file are characterized by the same *properties*. The actual data is stored as *property values*; the collection of property values for a single object constitutes an entry in the file. The entries of a file may contain substructures called *repeating groups* which are collections of associated sets of properties, with repeated associated sets of property values. A repeating group may contain repeating groups, and any repeating group may have an arbitrary number of repetitions of sets of its property values.

Figure 3 shows the structure of a file of commercial airline flights organized as a file, with destination as entries. It illustrates an entry for Boston, a set of properties which characterize all entries, and the repeating groups "origins," "fares," and "flights."

Property values, the actual data, are stored in various ways, depending on the declared *property type*. Property types presently available are:

1. *Numeric, signed integer,* stored as binary number in a field long enough to contain its declared size.
2. *Numeric, floating point,* stored as floating-point number.
3. *Roll-valued,* stored as the internal code for a name or other alphanumeric, with the actual characters stored in a roll. Alphanumerics in rolls may be multiword and arbitrarily long; their codes are automatically assigned by ADAM and are fixed-length.
4. *Raw,* stored as a string of bits of arbitrary length. The length of a value for a raw property may vary from one entry to another, and may change dynamically. Raw-type properties are used to store variable length, nonnumeric data, such as alphabetic text.



Boston	EDT	1.25	Worcester	EDT	F	6.55	MO-180	0920	0944	0
					M	5.00	MO-182	1252	1316	0
							NE-792	2015	2126	1
			Yarmouth	ADT	F	31.00	TC-461	1235	1255	0
					Y	23.00				
Chicago	CST	3.00								

-- and so forth --

Figure 3. Destination file structure.

5. *Repeating group,* stored as the repeated values of the properties in the group. Storage is used only for those repetitions of groups which exist (e.g., if there is no defined tourist-class fare from Worcester to Boston, no space is left for it). The number of repetitions of a group may vary from entry to entry and from time to time during the operation of the system.

As a consequence of the variability of the size of raw properties and the number of repetitions of repeating groups, entries in the same file may differ in length. The variability in entry size is dynamic—entries and entry size may be changed during ADAM operation without rewriting or recopying the whole file.

Each file has one roll-valued property whose values name entries. Data may be retrieved randomly by direct access to a specified entry through the roll which contains entry names. This roll also contains the current location in the entry relative to the beginning of the file. (Another roll contains the current location in the computer of each file.) Alternatively, all entries in the file may be accessed serially. Within an entry, access to a specific repetition of a repeating group is always serial; the entry is selected and each repetition of the group examined until the required one is located.

Rolls

A roll is organized as a series of *elements.* Each element consists of one or more external names associated with a single numeric code used as the internal name. The several names associated with the same code are considered synonyms of one another. Thus, the roll in which values for a property "city" were described might have both BOSTON and BOS equivalent to the same internal code, say the integer 5. All external references could then use either term; internally, 5 would be the unique name for Boston.

In addition to the names and code, a roll element may contain subsidiary values which describe the thing named by the element. For example, if BOSTON were the name of one of the entries in the Destination file, its roll element (and those of the names of all other entries) would contain subsidiary values which gave the relative location in the file of the entry data.

Each file has associated with it, but physically separate, an object roll which describes entries and their names; and a property roll which describes properties and their names and includes as subsidiary values the type, field-size, location within an entry, and other characteristics of the property.

Rolls are usually accessed implicitly and need not be handled by a user. A user may, however, specify in which rolls his roll-valued properties are to be defined, thereby providing a context for evaluating names—names and internal codes are unique only within the same roll. A user who wanted AST (the airline code for the city Astoria, Oregon) distinguished from AST (Atlantic Standard Time) would define the names in different rolls. This is one way the system can perform validity checks on data entered.

File Processing

ADAM is fundamentally a file processing system. Because of its adaptability and the option available to include user-programmed machine-language subroutines, one could say that ADAM can do anything any other program on the IBM 7030 can do. However, the interesting capabilities are those for which specific provision is made, without any programming required. The examples below are presented in a file processing language developed to help check-out ADAM itself. Although the language has been found useful enough to be applied directly to models so far constructed in ADAM, language design is not a goal of the ADAM project and the capabilities exemplified are more important here than the form in which they are stated.

Messages may be queries about individual objects and data items as in the first example of Fig. 4, in which the object Boston is accessed directly and the groups Origins and Flights scanned serially to find Flight AA123. Messages may change file data, as in the second example, in which the Destination file is accessed serially and changed under the condition "time zone equals EST." The third example shows the message for the generation of a report, in which the output file which is produced is formatted according to the specifications in the (previously prepared) format F23, and the resulting output titled.

In Fig. 5, a calculation to determine which are "short flights" is complicated by the possibility that a flight may begin just before midnight and end at an earlier time, producing a negative flight time. The next example shows a cross-file reference, with a decision made about what to print based on values of population from those entries in the City file which correspond to origin names in the Destina-

```
FOR DESTINATION BOSTON.IF FLIGHT NAME EQ AA123,TYPE FLIGHT NAME,DEPARTURE TIME,ARRIVAL TIME.

FOR DESTINATION.  IF TIME ZONE EQ EST, CHANGE TIME ZONE TO EDT.

FOR DESTINATION BOSTON. PRINT FORMAT F23 TITLE 'FLIGHTS TO BOSTON' ORIGIN NAME,FLIGHT NAME.
```

Figure 4. File Processing Messages.

```
FOR DESTINATION.  FOR FLIGHT.  IF ARRIVAL TIME - DEPARTURE TIME LS 6Ø AND GR-6Ø CHANGE FLIGHT NOTE TO SHORT FLIGHT
                               OR ELSE CHANGE FLIGHT NOTE TO LONG FLIGHT, PRINT FLIGHT NAME, FLIGHT NOTE.

FOR DESTINATION BOSTON.  FOR ORIGIN.  IF CITY (ORIGIN NAME) POPULATION LS 1ØØØØ, DISPLAY ORIGIN NAME,
                                      'POPULATION' OF ORIGIN=CITY(ORIGIN NAME) POPULATION, FLIGHTS.

FOR DESTINATION.  IF FARE CLASS EQ FIRST,
                  SAVE ORIGIN, 'OVERNIGHT COST' OF ORIGIN = 2 * FARE COST + CITY (ORIGIN NAME) AVERAGE HOTEL COST.
                  NAME TRIPFILE.
```

Figure 5. Mole File Processing Messages.

tion file. The destination entry "Boston" is accessed directly, and the group origins serially. For each origin, a direct access is made to the City file entry with the corresponding name.

Finally, the last example shows the creation of a new property, "overnight cost," which did not exist in either of the files from which the data for it was taken, and shows an example of an output file being saved and named instead of being deleted immediately after output. The new file—Tripfile—is a standard ADAM file, subject to any future processing desired.

A more detailed examination of the access techniques and operations used in the creation of the Tripfile is given in Table 1.

These examples show only some of the file processing capabilities of ADAM—a list of these and other capabilities is given below. They show that an on-line interpretive language with a powerful system behind it can handle a wide range of processing tasks. But the range of capability required to perform the many other file processing tasks conceivable or already implemented in various systems is even wider than this. Extensions of ADAM intended but not yet implemented include file access and read protection, more extensive text handling, and more on-line interaction with a user, among many others.

Table 1

Structure	Name	Access	Operation
file	Destination	Serial	-fetch from
file	Tripfile (output file)	Serial	-create -store into
group	Fare	Serial	-fetch from -compare property "Class" -input property value "Cost" to arithmetic
group	Origin	Serial	-copy from input file to output file
property	Overnight cost	—	-create -store into with data from Destination and City files
file	City	Direct, indexed by Origin Name in Destination file	-input property value "Average Hotel Cost" to arithmetic

File Generation

File generation tasks in ADAM are treated in the same way as those specified by any other message inputs—a message which describes the file and the data which go into it is translated and processed

through the same procedure by which queries and other tasks are treated. The generation of files from data contained within files already in the system was described above, and involves saving and naming the output file produced by a query or report request. Generation of files from bulk data on cards or tape is specified in a different operator language than queries or other messages. As with the language of the examples above, the language presently available in ADAM for checkout has been used in applications thus far, in preference to specifying another file generation language.

Specifications for file generation include directions for reading-in data from cards or tape as well as a structural description of the resulting file. An example is given in Appendix B. The data may include fields out of order and variable-length fields, and separate fields may be subjected to user-supplied or system standard conversion subroutines on the way in.

Messages and Languages

A designer modeling his system in ADAM has the choice of using a language already prepared and inserted in the ADAM language file (such as the language used in examples thus far) or preparing a language specification of his own. (The preparation of new languages is not treated here for lack of space. Suffice it to say that the ADAM translator is a syntax-directed translator of a type found in many compilers and that a new language is prepared by constructing a syntax-diagram and inserting it into the ADAM "language" file.) Regardless of the language(s) used, messages may be inserted on-line or off-line on cards.

During ADAM on-line operation, a user has available a language change capability through the ADAM string-substitution facility, by which he may define the meanings of certain input words. Words defined by string substitutions are replaced by their defined equivalents before an input message is translated. String-substitution definitions may specify that parameters (comprising the words following a use of the defined word) be inserted. Thus, for example:

LET NONSTOP MEAN (IF NR OF STOPS EQ ∅). and
LET SKED MEAN (FOR DESTINATION/2/./3/ TYPE ORIGIN/1/FLIGHTS) USING REINSERT.

define substitutions, and the message

SKED BOSTON CHICAGO NONSTOP.

would be transformed to

FOR DESTINATION CHICAGO.IF NR OF STOPS EQ ∅, TYPE ORIGIN BOSTON FLIGHTS.

before translation.

Problem-Specific Calculations

In order to handle problems which are not easily or efficiently stated in an on-line message language, ADAM includes a capability for incorporating subroutines specifically coded for a model. An ADAM compiler, called DAMSEL, compiles routines for inclusion in the routine file. Although the DAMSEL compiler constitutes the primary means for writing user-specific subroutines, other compilers may be used. In particular, a set of routines called COMFORT (COMpatible FORTran routines) adjusts the output of specially prepared FORTRAN compilations to be compatible with ADAM.

DAMSEL itself allows the usual complement of arithmetic-assignment, conditional, and subroutine-call statements, and includes a macro facility for extending the DAMSEL language. In addition, it provides statements specifically designed to create, augment, modify and retrieve from ADAM data structures. File-manipulation statements refer directly to files by name, or use names which the subroutine receives as input parameters from other routines or from a message input. In either case, statements in the subroutine are independent of the format of data referred to; data descriptions are retrieved from the rolls when a subroutine is compiled or executed.

A routine in the system is called by a message such as

DO OPTIMAL (DESTINATION,BOSTON, ORIGIN,CHICAGO)

or within a message as a function

FOR DESTINATION.IF TIMECALC(CHICAGO) GR ∅,...

(in which OPTIMAL and TIMECALC are hypothetical user-coded routines).

Output Formatting

Output formatting in ADAM is the process of rearranging the data from an ADAM file into an order appropriate for output, translating names from their internal coded form to alphanumeric, and sending the resulting messages to an output scheduling program for actual output. The entire process is directed by a format specification, or *format*, from the ADAM format file.

Formats are typically prepared off-line with a formatting macro written for the purpose. A format specification describes margins; spacing; pagination; file data; titles; and, for display devices, vectors and points. The formatting program adjusts the output as required to conform to the physical characteristics of the device to which the output is directed, without any necessary specification on the part of the user (who may, however, control the format on the basis of device, if he desires).

SUMMARY OF CAPABILITIES

ADAM, as an on-line system, provides for multiple consoles used simultaneously, remote operation, cathode-ray tube output both character and pictorial, and light-pen, push-button, typewriter, and teletype input. As a file-handling system, ADAM incorporates most of the features found in the more advanced file handling systems available today.

The file structure allows variable-size data fields, nested groups of sublevel fields, dynamically variable file and entry size, and both dictionary and cross-file indexing. File data may be integer, floating-point, alphanumeric, or coded.

File generation may be from card or tape input, from data in existing files, or from data entered on-line. Bulk file generation provides for fixed- or variable-field input data, optional validity checks on incoming data, and the provision to define and use special conversion subroutines on data before it is stored in the newly generated file. On-line file creation may restructure existing files or extract data from subsets or arithmetic combinations of data in existing files. Files generated on-line may include newly defined fields.

File processing includes querying on logical or arithmetic criteria applied to any data items or combinations of data items, data updates or changes, operation of general or special-purpose subroutines, file sorting, and output. Subroutines may be programmed in FORTRAN or in DAMSEL, an ADAM compatible compiler.

Output formatting provides for formats prepared off-line and selected and performed on-line, titles, routing to other terminals, and pictorial, as well as symbolic, outputs.

Language variability includes, in addition to the provision for defining a completely new file processing language, provision for on-line definition of synonyms (for file names, field names, etc.) and for on-line definition of special-purpose words to be used at selected input terminals.

EXPERIENCE

Experience with ADAM since it became operational early in 1965 included five diverse applications:

1. A system for scheduling resource use and activities aboard a manned satellite. The model uses elaborate display formats of file data to present potential schedules to analysts.
2. A subsystem for the command and control of tactical forces through the use of an airborne sensor to locate friendly units. The application is a hybrid of real-time sensor inputs (which become frequent and voluminous file updates) and file retrieval requests inserted on-line by operators.
3. A study of a personnel and organization subsystem devised by the System Development Corporation. This application involved the manipulation of file structures accomplished with query language inputs instead of programs.
4. A system of inventory management which involves a large data base, processing of items from several files at the same time with cross-file indexing, and reporting in formats appropriate to condensed output.
5. A man-job-match model in which personnel qualifications for defined jobs were assessed and tentative personnel assignments made. This model used push-button and light-pen inputs to select parameters for user-prepared subroutines which performed the specialized qualification and assignment algorithms.

CONCLUSION

From experience with building ADAM and applying it to a wide range of problems, a qualitative assessment of some of its principles can be made. No quantitative studies have been made yet.

Generality is possible. The fact that ADAM was built and applied to the diverse applications described previously shows that a general-purpose file-handling system can be made to work.

Generality is expensive in computer time and space. The ADAM system serves its intended purpose, as a design verification tool, but is hardly the way an operational system would be built. ADAM comprises 90,000 instructions, representing a range of capability unlikely to be required in any single application. During ADAM operation, the efficiency of the programs is reduced somewhat by their generality, but the cost is considered accept-

able for ADAM purposes (in a laboratory environment) in the light of the capabilities offered.

Generality saves implementation time. If a generalized data management system is already available, the time to implement a defined problem is considerably shortened, since functions required for problem solution already exist. In one case, a partial model of a subsystem to display pictorial information on satellite positions as viewed from a ground station was implemented concurrently in ADAM and a FORTRAN program. With the query handling and display capabilities already available in ADAM, but required as new programs in FORTRAN, the job took nine times as long to implement in FORTRAN.

ADAM, then, is a system which provides a modeling capability of general utility as a design tool. But, more importantly, ADAM represents a concept of generality, of potential application to large operational systems, and of demonstrated usefulness.

Appendix A

IMPLEMENTATION

ADAM accomplishes its functions through a variety of well-known programming techniques, most of which have been used individually but which are not usually combined in a single system. The five major principles observed in the design of ADAM are:

1. Keep as much of the model specification as possible in data, not in program—exemplified by the use of files to contain formats and language specifications, routines, and other components of the system being modeled.

2. Make programs insensitive to change in the size and format of data—generally, through the dictionaries and other mechanisms which make the ADAM system and models within it self-descriptive, and therefore able to vary dynamically.

3. Make the system contain itself as data—for example, all ADAM routines are in a routine file and ADAM files and dictionaries are just like user files and dictionaries.

4. Allow for handling exceptional cases and for expansion by including in all specifications the option "do a subroutine"—for example, in addition to the usual description of an input data field, an option for the person specifying file generation is "do a subroutine here," (presumably a special conversion he wrote) and similarly a format-specification writer can specify that an arbitrary subroutine can be executed in the middle of the formatting process.

5. Allocate compiler resources (memory) only as needed—through dynamic allocation programs.

Environment

Some of the specifications of the computer on which ADAM runs are given in Table 2. Primary storage is approximately 65,000 64-bit registers of core; secondary bulk storage is 4 million 64-bit words of disk; input/output channels are provided for on-line terminals including typewriters, display consoles, teletypes, and printers.

Table 2. Configuration of the IBM 7030 in The MITRE/ESD Systems Design Laboratory

WORD:	64 Bits = 8 characters
CORE:	65K Words–520K characters
	Addressable by bit
	Access time 2 microseconds
DISK:	4 million words = 32 million characters
	Rotation time 33 milliseconds
	Transfer time 4 microseconds
I/O:	16 Channels
	3 Channels for 12 tapes
	1 Channel for 6 display-typewriter consoles
	1 Channel for 6 printers
	7 Channels for teletype or phone lines
	4 Channels for printer, punch, reader, operator console

Jobs run on the computer operate under control of an operating and monitor system called MCP. ADAM is no exception; as far as MCP is concerned the entire ADAM system is a single-user program. The MITRE version of MCP time-shares background and foreground problem programs, which are operated completely independently of one another. Thus, in addition to being a multiple-user system itself, ADAM operates in an environment in which it shares the computer with other jobs (which may be independent copies of ADAM systems).

System Control

The system control philosophy of ADAM is to accept input messages as they arrive, recognize them, and place them according to a priority scheme

in a job queue. ADAM will recognize any input language whose recognition rules are given to the system. When processing of a single message is completed, the top of the job queue is examined and the appropriate routine is called (problem program, translator, etc.). When processing of a message is started, it runs to completion, being interrupted only for recognition and stacking of input messages, or to initiate output.

Requests for output are handled immediately if the channel and peripheral gear are available; otherwise, they are queued up and sent out when possible. All system routines (input/output handling, job scheduling, memory allocation, etc.) are subroutines and may call each other as needed.

Allocation

In ADAM, computer resources are allocated to the task at hand as the need arises during the processing of a message. Separate allocation programs handle the allocation of secondary storage, core storage assigned to data storage, and core storage assigned to routines and tables. Second-level allocation programs use these programs to allocate space for files, rolls, and an ADAM artifice for working storage called a stream.

Disk Allocation

The disk allocation algorithm operates by linking together pages of disk, into *regions* with the link table kept in core. The disk allocator program also handles all disk-to-core and core-to-disk transfers, relieving other programs of the responsibility for following the links and making all disk allocations appear to be contiguous. The linking procedure allows the size and location of allocations to change at almost any time. From time to time, the system performs a wholesale reallocation of disk to make regions contiguous—they can subsequently be read with fewer disk accesses per region.

Core Allocation

The two separate core allocators (programs and data) use opposite ends of memory; data is allocated from one end and programs from the other, so that the dividing line can move and the ratio of memory-for-program to memory-for-data adjusts dynamically as the situation requires (Fig. 6).

For routine storage, the unit of allocation is the routine or table, with its size fixed at compile-time. Routines and tables are relocated when loaded into core from the routine file and may contain relocatable addresses within themselves. Once loaded, a routine does not move in core. Each routine and table is accessed through a program allocation table, created and maintained dynamically by the program allocator.

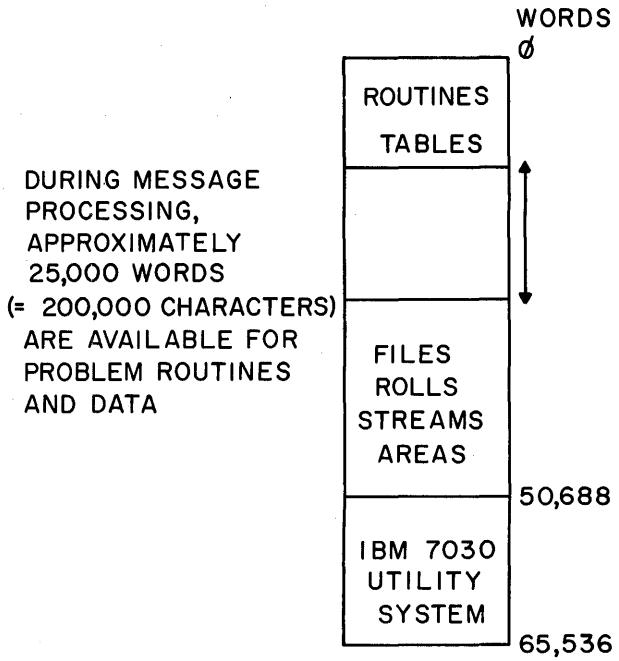


Figure 6. Dynamic core memory allocation.

Data allocations on the other hand may move in core at almost any time. The unit of allocation is 512 words, the disk-arc size, to facilitate transfers from disk to core and back. A single allocation always remains contiguous so that it may be continuously addressed and indexed. An instruction in the 7030 repertoire: "transmit (any number of sequential) words from one core location to another" allows the data allocator to dynamically change the size of an allocation and to move as much as necessary of the remaining contents of memory to keep data allocations contiguous. Data allocations are addressed through index registers which are updated by the allocator whenever data moves.

Stream Allocation

A stream is a combined allocation of several levels (presently core and disk) of storage, used as a single-level continuously addressable store. Streams are used as temporary storage for, for example, input and output queues. As it is made up of data-core areas and disk regions, it is allocated in units of 512 computer words, but the Stream Allocator makes it appear to a using routine as a con-

tinuous series of up to 2^{18} single registers. The portions of a stream in core at any time are called *blocks;* a stream may have an arbitrary number of core blocks attached at any time. The fundamental operation on a stream is "locate an address" which causes the allocator to insure that the desired address is in a core block. The core blocks thus operate as separate movable windows through which a routine may look at registers in the stream. Any number of independent core blocks may be attached to a stream at the same time, or none may be attached, in which case the stream is entirely on disk until core blocks are later attached.

The allocator assigns disk space only for those pages of a stream which contain registers into which data have been stored.

File Allocation

File allocation involves assigning data core to individual entries from files, to lists of entries in arbitrary order, or to an entire file. The allocation program assigns core space by an algorithm based on the size of the material requested (obtain from the rolls) and the amount of available core. A file allocation is always at least as big as the file entry being accessed, so that machine indexing can be used without interruption for input/output. The allocator automatically transfers file data from disk to core and core to disk as required.

File Structure in Storage

In main storage, all data for a file are contiguous. Figure 7 shows the appearance of file data in memory. A file may be discontinuous in secondary (disk) storage, but the allocator handles all discontinuities without requiring notice by any other program. Disk discontinuities may occur when the size of an entry or a file is changed during ADAM operation.

Every entry of every ADAM file contains a set of standard properties which describe the entry and thereby provide for dynamic variability. One of the standard properties is "current size of this entry." Each entry is stored in four parts: the standard properties, fixed-length properties, variable-length properties, and an area between fixed and variable parts available for expansion by addition of new material, either fixed or variable. When this area is used up, more is added here. Standard properties describe the sizes of the various parts.

All repeating group data appears in the variable

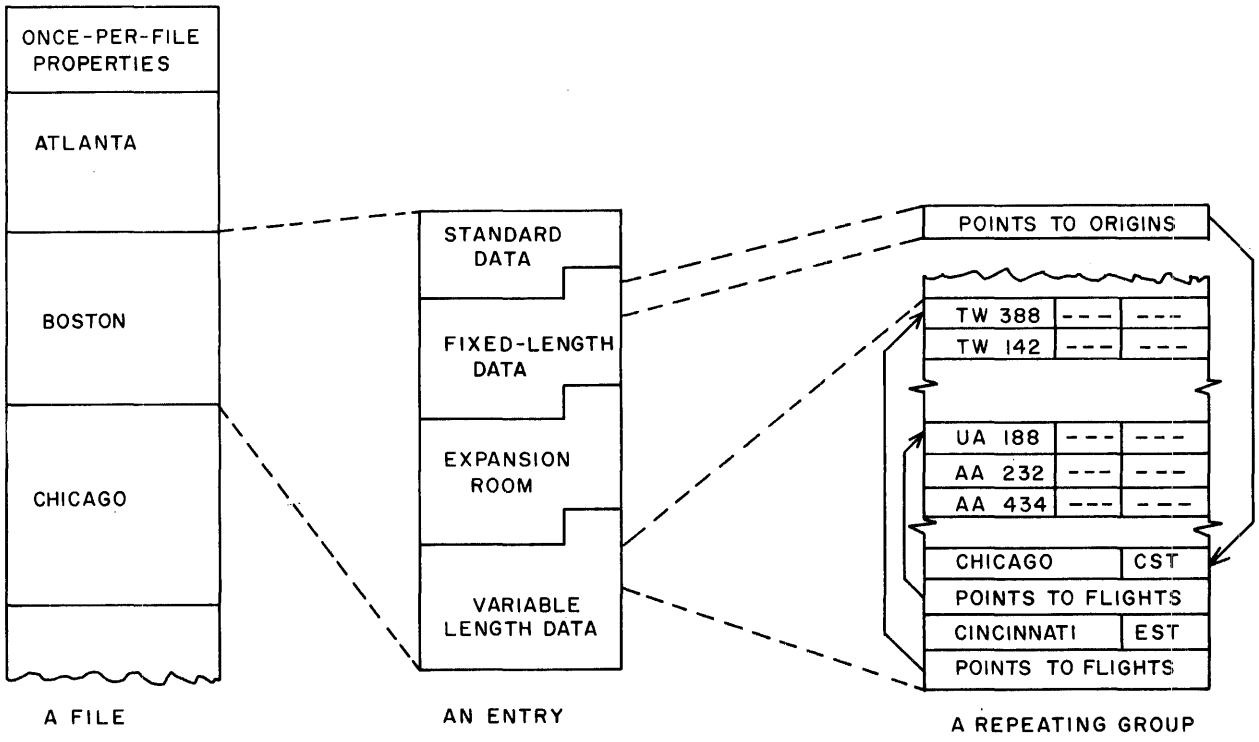


Figure 7. File data in storage.

```
GENERATE FILE,DESTINATION,CARDS.BEGIN OBJECT.
ROLLVALUED,     OBJECT NAME
                (CITY),          LENGTH VARIABLE.
                                 SCAN TO NON ' '.      ALPHA.USE OBJECT ROLL.
ROLLVALUED,     TIME ZONE,       LENGTH 3 COL.         ALPHA.USE ROLL TIMES.
DECIMAL,        TAXI
                (TAXI FARE),     SPACE TO NEXT CARD.
                                 SPACE 25 COL.
                                 LENGTH 5 COL.         NUMERIC.4 DIGITS.
BEGIN
GROUP,          ORIGIN
                (ORIGINS),       TERMINATED BY '-'.
                                 SPACE TO NEXT CARD.   BEGIN REPETITION
ROLLVALUED,     NAME,            LENGTH VARIABLE
                                 SCAN TO NON ' '.      ALPHA.USE OBJECT ROLL
                                                       OF CITY FILE.
ROLLVALUED,     TIME ZONE,       LENGTH 3 COL.         ALPHA.USE ROLL TIMES.
BEGIN
GROUP,          FARES,           TERMINATED BY '--'.
                                 SPACE TO NEXT CARD.   BEGIN REPETITION
ROLLVALUED,     NAME
                (CLASS),         SPACE 12 COL.
                                 LENGTH 2 COL.         ALPHA.USE NEW ROLL
                                                       CLASSES.
DECIMAL,        COST
                (FARE),          SPACE 12 COL.
                                 LENGTH 6 COL.         NUMERIC.5 DIGITS.
                                                       END REPETITION.
END
GROUP,          FARES,
BEGIN
GROUP,          FLIGHTS
                (FLIGHT),        TERMINATED BY '---'
                                 SPACE TO NEXT CARD.
ROLLVALUED,     NAME,            SPACE 25 COL.
                                 LENGTH 5 COL.         ALPHA.USE OBJECT ROLL
                                                       OF FLIGHT FILE.
INTEGER,        DEPARTS
                (DEPARTURE TIME), SPACE BACKWARD 25 COL.
                                 LENGTH 5 COL.         NUMERIC.4 DIGITS.
INTEGER,        ARRIVES
                (ARRIVAL TIME),  SPACE 4 COL.
                                 LENGTH 5 COL.         NUMERIC.4 DIGITS.
INTEGER,        NR OF STOPS,     SPACE 21 COL.
                                 LENGTH 1 COL.         NUMERIC.1 DIGIT.
                                                       END REPETITION.
END
GROUP,          FLIGHTS.                               END REPETITION.
END
GROUP,          ORIGINS.
END OBJECT.                      SPACE TO NEXT CARD.
```

Figure 8. Message to generate destination file.

part of an entry, to allow for variable numbers of repetitions in different entries. Groups are conceptually structured as entries, with a fixed and variable part, but physically they differ from entries in that all their data are not necessarily contiguous.

Variable data, including repeating groups, are accessed by pointers within the fixed-data section of the entry or group to which they belong. Property descriptions reside within the property roll for a file, rather than in the file itself.

Appendix B

AN EXAMPLE OF FILE GENERATION

A message to generate the Destination file is shown in Fig. 8. In this file generation language, spacing is not significant, so the message was spaced out for readability. The first column gives, for each property, the property type; the second column, the property name with synonyms enclosed in parentheses. The next column contains directions for reading input, in this case from cards. Variable length fields and fields out of order are handled here. The last column gives the type of conversion (e.g., ALPHA, NUMERIC, or the name of a user-supplied conversion subroutine) to be applied to the input data as it is read in.

THE ENGINEER–SCIENTIST AND AN INFORMATION RETRIEVAL SYSTEM

C. Allen Merritt and Paul J. Nelson
IBM Technical Information Retrieval Center
International Business Machines Corporation
Yorktown Heights, New York

INTRODUCTION

During the last few years a veritable explosion of study, effort and accomplishment by business, government, and university organizations has taken place in the realm of information retrieval and dissemination. Theoretical problems have been explored, new equipment and techniques have been developed, and a number of successful operating systems have been implemented.

This paper will look at the impact of such information retrieval systems or centers upon their most important clientele—the engineer–scientist or the technical professional. Who are these people? Let us define them as professionals, working in a scientific or engineering discipline, and very likely in a research and development environment. Most of them share some basic information problems. They recognize that the rapid expansion of knowledge and data in their technical fields is taxing their time and memory capacity to the limit. There are increasing pressures on them for more interdisciplinary knowledge. Technical obsolescence is a real problem, for they live in a fast-moving environment where today's research idea can be tomorrow's hardware.

As a result of these problems and pressures, the engineer–scientist, sceptical at first, has been drawn toward a new source of help—the machine-oriented information retrieval and dissemination system, backed by the technical library.

To illustrate this new relationship between the technical professional and the information retrieval system, we will examine the philosophy and mechanics of the IBM Technical Information Retrieval Center. This particular system represents an integrated approach to the storage, announcement, dissemination, and retrieval of technical information. It combines the best features of both machine processing and human information skills, and is applicable to a wide range of data bases and system activities.

PHILOSOPHY AND APPROACH

The IBM Technical Information Retrieval Center (ITIRC), located in Yorktown Heights, New York, was established to serve the IBM scientists and engineers in all their laboratory locations. A wide diversity of occupations are involved, ranging from physicist to circuit designer to programmer. ITIRC exists to supply the right information to the right person in the shortest possible time and at the least possible cost. This is no small undertaking when you consider the size of the company, the number of locations, and the tremendous range of interests in research, development, manufacturing, and sales. Add this complexity to the general information problems described earlier and the need for an information retrieval center becomes urgent.

Regardless of need, however, an information retrieval system cannot be established until man-

agement recognizes its value and is willing to set aside funds and manpower for its operation. Further, this funding must cover a period long enough to permit a valid judgment about the value of the services rendered. How this valid judgment can be arrived at will be discussed later.

Along with management and the staff of the information center, the individual engineer-scientist has certain responsibilities to the system. First, he must supply the system with a complete and accurate description of his occupation and related information needs. Second, as his assignments or interests change, he must be alert in notifying the system about the changes. Finally, the information center looks to him for feedback—is he receiving pertinent material, is he missing significant information, is he getting prompt service? The system has built-in techniques for making this feedback easier, but it still takes initiative on the part of the customer to supply complete and pertinent data.

In parallel with the user's responsibilities, the staff of ITIRC accepts similar ones. We are committed to translate his interests into the retrieval and dissemination programs accurately and fully, to act promptly on his suggestions and complaints, and to follow up with him personally whenever significant questions arise.

The ITIRC also has commitments to management. Like every operation in a large business, we must frequently justify our existence, measure our services in tangible terms, and demonstrate growth and improvement in many ways. This, too, is not an easy task because information retrieval deals so often with intangible results—but it must be done.

THE SCOPE OF THE SYSTEM

Dissemination

What we call the Current Information Selection program is one of the keystones of ITIRC activities. The engineer-scientist customer submits a textual description of his information needs and job responsibilities. This goes to an information retrieval specialist who is thoroughly familiar with the sophisticated machine-searching techniques that are used, as well as with the types of documents that are entering the system. He analyzes the user's specifications and constructs an accurate profile for entry into the system. If necessary, he goes back to the customer and obtains more detail or clarification.

Once the profile is checked out and operating, it is compared several times a month with all of the current documents being processed into the information center. Whenever the text of a document abstract matches the profile, a printed abstract is generated automatically and sent to the customer. He screens each notification and takes appropriate action—by telling the Center how relevant the document is, by reviewing it on microfilm, or by requesting a personal copy if he needs it.

Retrieval

Retrospective searching is another important function of ITIRC. All current document input is added to a set of master searching tapes, a file that now includes abstracts of over 125,000 documents. Search questions come in from customers throughout IBM, often through the local IBM libraries. They are directed to the appropriate IR specialist, depending on which of the several classes of documents need to be searched. The specialist analyzes the request and formulates one or more machine search questions, using the various logical tools at his disposal, as well as his broad knowledge of the data base to be searched. The output of abstracts that answer the question is reviewed by the specialist and sent to the customer. He may be satisfied with the answers, or request a deeper or narrower search, or ask for another search on a related topic that has been brought to light. In the latter case, the process is repeated.

Retrospective searching is normally scheduled to yield answers within 48 hours. However, an emergency question can be processed almost immediately. Also, to handle urgent requests from distant locations, the Center maintains two satellites at laboratories in San Jose, California, and LaGaude, France. These have the search programs and duplicate sets of the master search tapes that are updated monthly.

Announcement

Supplementing the Current Information Selection program already described, ITIRC publishes monthly three series of announcement bulletins. These cover *all* current documents processed in the three major data bases—IBM reports, IBM Invention Disclosure material, and selected external (non-IBM) documents and journal articles. These bulletins are made available to all IBM libraries, report centers, and publishing groups. They are also distributed to selected individuals who have expressed a need for reviewing all current literature in the system.

The bulletins contain abstracts of documents in order of accession, including titles, authors, sources, detailed abstracts, and descriptive index terms assigned by the IR specialists. Each issue also contains a category index that offers the reader a quick way to scan the contents selectively. This section lists each document title under one or more of 23 broad subject headings, with easy cross-reference by page number back to the abstracts. A second manual searching aid is an alphabetical subject index based on the descriptive terms assigned to each document. Each entry includes title, accession number, and page reference. Both indexes and abstracts can also direct the reader to a location on microfilm where the complete text of most documents is available for viewing.

Supplementing the general-purpose bulletins are indexes designed for library and reference use. For the same documents we publish monthly alphabetical author indexes, indexes of the original source numbers, and sequential listings of the accession numbers—all with complete titles. Any of these machine-produced indexes can be cumulated quarterly or as required. And, with the same machine programs, various special classes of documents can be pulled out and indexed.

A specialized type of announcement medium is a monthly compilation of all the current research and development projects in the company; we call this an automated project file. Updated regularly, it contains descriptions of the projects and budget, manpower, and planning information. It is distributed to a controlled listing of management people as an information and control vehicle.

Microfilm and Hard Copy

To make the complete text of input documents as widely available as possible, ITIRC is operating a comprehensive microfilming program. Currently we are putting on film all the invention disclosure material, almost all the IBM reports, and as much of the external material as copyright and distribution restrictions will permit.

The medium used is 100-foot reels of 16mm microfilm, with a capacity of about 2300 frames per reel. Depending on the equipment available, these are distributed in reel form or packaged in special cartridges. Most IBM library locations have complete files of film going back several years and now covering over 24,000 documents.

Thus, for complete copies of documents processed by ITIRC, the system customer has two alternatives. He can request them from his nearest IBM library, which has many current documents on hand or can order them if needed. Or he can go to a nearby microfilm reader to scan them. Some locations have reader-printers, in which case the user can make selective copies of pages or short documents.

Admittedly, the problem of supplying hard copy to many customers is a constant challenge. However, the Current Information Selection program helps the situation by preselecting a relatively small percentage of the total documents for the user. All of the major system output—machine listings and publications—offers complete bibliographical data and detailed abstracts. This gives the customer a chance to do his own screening without having to order documents blindly. By the time he decides he wants a personal copy of an item, he is reasonably sure of its value. The screening process built into the system, combined with microfilm accessibility, tends to control the amount of hard copy requested and to assure its worth when requested.

THE DATA BASE OF THE SYSTEM

One of the major goals of ITIRC is to cover the pertinent scientific and technical literature, both inside and outside IBM, as completely as possible. Obviously we had to start with what we considered the most important types of data and expand from there. The following are the major classes of input documents now being processed:

1. IBM Research and Engineering Project Files, mentioned earlier, are the official reporting medium for all R&D activities within the Corporation.
2. IBM reports include formal technical reports, laboratory and testing reports, informal published memos, IBM papers cleared for external use, patents issued to IBM personnel, reference and operating manuals, and a variety of miscellaneous documents.
3. IBM Invention Disclosures are novel ideas submitted as Inventions to solve specific problems. The most promising are selected to be filed for patent.
4. Non-IBM reports are selected documents of interest to IBM engineers and scientists. Typical examples are Defense Documentation Center reports, university reports, and technical journal articles.

5. IBM Suggestions. Like many other large companies, IBM has a suggestion system that over the years has amassed a large number of suggestion reports covering proposed changes in products or procedures. These have been placed on a separate set of search tapes in a structured format. When the numerous new suggestions are received they are matched against the data file to see whether similar ideas have been submitted in the past. If no match occurs, the suggestion is investigated further by the Suggestion Department to see whether it should be accepted and an award made for it.

As new technologies develop and new kinds of publications and reports appear, the ITIRC staff is constantly evaluating them and expanding the data base of the system.

THE SEARCH LOGIC OF THE SYSTEM

One of the most important factors in the ITIRC system is the computer logic used for searching (both retrospective searching and current dissemination). It is a flexible technique for searching a normal text data base (in this case, the text of the abstract). It is not a simple technique to use, but an experienced IR specialist can achieve a high degree of precision with it.

A fundamental point is inherent in the words "normal text." Since the data base contains the language of the original document (and author), we can phrase search questions or user profiles in the same kind of normal English or accepted technical language. We use not just single words but phrases and adjacent or associated words. We scan not only the abstract but the title, author, source information, and index terms associated with a document. With this in mind, let us look at the specific types of logic available.

Single Word Logic

Single words can be searched for individually.

a) RETRIEVAL

Families of single words can be searched with an OR technique, to cover variations in spelling, synonyms, and the like.

b) RETRIEVAL *or* RETRIEVING *or* SEARCHING *or* SEARCH *or* SEARCHES

We can use AND logic to search for combinations of associated single words, where two or more of the words must be present in the abstract to satisfy the request.

c) INFORMATION *and* RETRIEVAL

We can use OR logic within the AND logic groups.

d) (INFORMATION *or* DOCUMENTS *or* DATA *or* LITERATURE)
and
(RETRIEVAL *or* RETRIEVING *or* SEARCHING *or* SEARCH *or* SEARCHES)

And we can expand a broad, simple AND group to make it much more narrow and specific.

e) (INFORMATION *or* DOCUMENTS *or* DATA *or* LITERATURE)
and
(RETRIEVAL *or* RETRIEVING *or* SEARCHING *or* SEARCH *or* SEARCHES)
and
(MEDICAL *or* MEDICINE *or* BIOMEDICAL)

In the examples, a) and b) are very broad questions which would probably not be used to search a large file. Example c) is more specific, but would probably miss some of the documents that would be picked up by d). And e), which requires a match from each of three groups, is even more precise.

Adjacent Word Logic

This is a powerful tool for searching the normal text of abstracts. Two normally related words may by their contextual positioning have entirely changed meanings. To avoid such "false drops," we can search for them as adjacent words in a specified sequence.

a) INFORMATION RETRIEVAL

The above question would not match on RETRIEVAL OF INFORMATION or any other contextual arrangement. Within these adjacent word groups, we can make allowances for spelling and synonyms by means of OR logic.

b) (INFORMATION *or* LITERATURE) (RETRIEVAL *or* SEARCHING *or* SEARCH)

This would match on INFORMATION RETRIEVAL or LITERATURE SEARCHING, etc. Also, we can look for related groups of adjacent words by lumping them together as a single OR family.

c) INFORMATION RETRIEVAL
or
SELECTIVE DISSEMINATION

We can again use the AND technique to make a question more precise. One adjacent word group can be ANDed with another, or a single word.

d) INFORMATION (RETRIEVAL *or* SEARCHING *or* SEARCH)
and
MEDICAL (LITERATURE *or* DOCUMENTS *or* DATA)

e) INFORMATION (RETRIEVAL *or* SEARCHING *or* SEARCH)
and
MEDICINE

The adjacent word technique is particularly useful when we are searching for specific phrases that we know are likely to occur in pertinent documents. It makes it easy to look for "operations research," "numerical control," "time sharing," and the like.

The Match Criterion

This is simply a numerical designation of the number of matches required for the computer to register that a document answers a given question or satisfies a user's profile. Either a single word or a complete logical group (OR, AND, or adjacent word group) is considered as one logical unit. Raising this criterion beyond a match of one is a helpful device when a question involves two distinctly different subject areas that we are trying to find in combination. For example, if we wanted documents about information retrieval and dissemination *only* when they related to medicine and medical literature, we could raise the match criterion to two and phrase the question as follows:

INFORMATION (RETRIEVAL or SEARCHING or SEARCH) *or* SELECTIVE DISSEMINATION	one logical unit
and	
MEDICAL (LITERATURE or DOCUMENTS) *or* MEDICINE or BIOMEDICAL	one logical unit

Both logical units would have to be found in the document to satisfy the criterion of two.

NOT Logic (Negative)

If a user is interested in certain aspects of a given subject area but wishes to eliminate or bypass portions of it, we can instruct the computer to ignore the documents matched if they contain specified words or phrases. For example:

INFORMATION (RETRIEVAL *or* RETRIEVING *or* SEARCH *or* SEARCHING)
not SDI
not SELECTIVE DISSEMINATION

Absolute YES Logic (Imperative)

If a user wants to see *all* abstracts that contain a specified word or phrase or name, regardless of the rest of the document's content, this can be achieved by appropriate coding. The specified imperative will override all other logic, including the match criterion and NOT logic. If the NOT example just given also contained MEDICINE as an imperative, an answer would be printed out even if the abstract contained both MEDICINE and SDI. This technique is very useful for extremely specific words that we know will identify documents pertinent to the user's interests.

The examples used to describe the search logic were necessarily brief and simple. Figures 1 to 3 give a more complete illustration. They show how the original information supplied by the customer is converted into a working profile by the IR specialist, and how the profile actually matches against current input documents.

HOW DO WE EVALUATE SYSTEM PERFORMANCE?

Measuring the results obtained from a system like ITIRC is a challenge that increases as the system grows, the data base broadens, and the number of users expands. It cannot be done on a hit-or-miss policy of voluntary feedback, although spontaneous reports from users are very helpful. We have worked out more formal techniques.

Each set of answers to a retrospective search request is sent out with a simple return card. The customer is asked to check off the following:

—The following items were not specific answers to my question.
—The following were not listed as answers but I believe should have been.

"I am responsible for development and marketing of Information Retrieval applications across all industry lines. This includes feasibility studies of techniques of automatic indexing, abstracting; optical character recognition, type composition and editing; language translation, syntactic analysis; query languages; file organization, image storage and retrieval; transmission of images; copyright problems; dissemination of information, as well as mechanization of library operations."

(User also primarily interested in external sources)

Figure 1. The original information supplied to the IR specialist by a user on his data sheet.

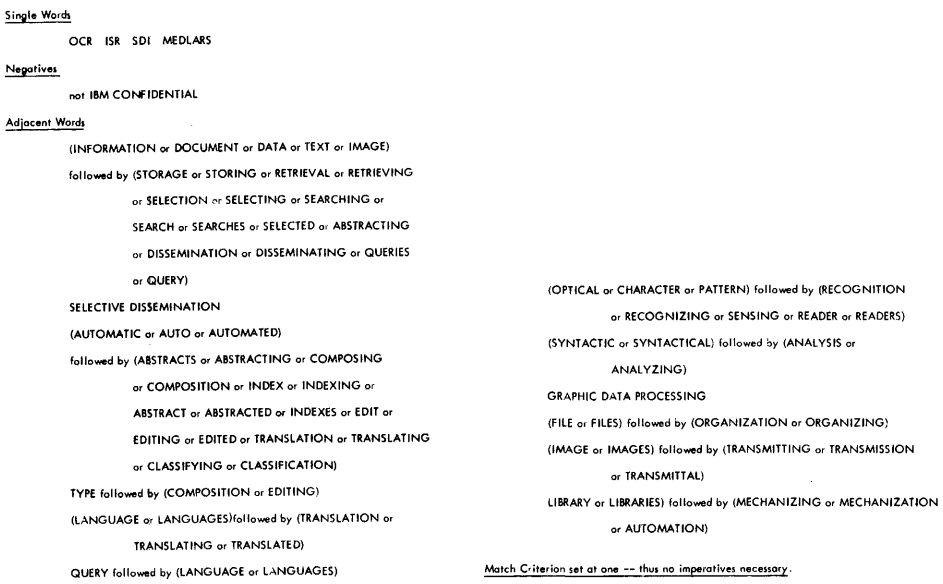


Figure 2. The completed profile created by the IR specialist from the information on the original data sheet.

—The answer report proved satisfactory to my question.
—Other comments.

If the response is negative or if the search seems to be incomplete, the IR specialist concerned promptly goes back to the requester, by phone or letter, and offers further assistance. For example, he may run a revised search, based on added data supplied by the user.

Records are kept of the number of searches, the processing time, the user's reaction to the answers, and whether any further action was required.

Since the Current Information Selection program deals with over a thousand users on a recurring basis, a mechanized feedback system was in order. Accompanying each printed abstract sent to a customer is a matching Port-A-Punch response card. When he reviews the abstract, he simply punches out the appropriate box and returns the card. He has a choice of the following reactions to a document:

1. Abstract of interest, document not needed.
2. Send copy of document.
3. Abstract of interest, have seen document before.
4. Abstract not relevant to my profile.
5. Comments—written below. (Change of address, change to profile, etc.)

The Port-A-Punch cards, into which the program has already punched user, document, and date identification, come back to ITIRC for processing. Document requests and comments are sorted out for immediate action. Then periodically the accumulated cards are run against a statistical program.

The statistical program supplies a complete analysis of the returns, for each individual user and for all users, with separate reports for each of the major data bases:

1. Total notifications sent out.
2. Number and percentage of response cards returned.
3. Number and percentage of interest (with a breakdown into each of the three responses listed above).
4. Number and percentage not relevant.

In addition, the program gives us several special listings:

5. Users who received no notifications in the current period.
6. Any users who failed to return their response cards within a specified period.
7. A list of users whose "not relevant" response exceeded a predetermined percentage.

With the help of these statistics, the IR specialist can quickly identify any customers who do not seem to be getting satisfactory results from the system. He can then review the profiles and if necessary make personal contact with the users to revise or

AD-608747. LINGUISTIC TRANSFORMATIONAL ANALYSIS. OCTOBER 1964.
DDC

THORNE, JP LYONS, J
INDIANA UNIVERSITY

AD-608747 RADC-TDR-64-200
CONTR AF-30(602)-2951

THE CONTRACT WAS CONCERNED WITH THE FEASIBILITY AND UTILITY OF A KERNELIZATION PROCEDURE FOR PURPOSES OF INFORMATION RETRIEVAL. THE LEADING SECTION DISCUSSES, IN GENERAL, THE PROBLEMS INVOLVED IN THE KERNELIZATION OF COMPLEX ENGLISH SENTENCES. THE REMAINDER IS IN THE FORM OF APPENDICES. APPENDIX I CONTAINS A DETAILED REPORT OF THE KERNELIZATION PROCEDURE. APPENDIX II REPORTS ON A SERIES OF EXPERIMENTS, TO DETERMINE TO WHAT EXTENT INFORMATION WAS PRESERVED IN KERNELIZED VERSIONS OF SENTENCES. APPENDIX III REPORTS ON A FREQUENCY COUNT OF THE TRANSFORMATIONS EXHIBITED BY A STRETCH OF RUNNING TEXT. FINALLY, APPENDIX IV CONTAINS A LIST OF TRANSFORMATIONAL RULES WHICH HAVE ACTUALLY BEEN WRITTEN, WITH REFERENCES TO SIGNIFICANT PUBLISHED (AND SOME UNPUBLISHED) MATERIAL. 119P.

23-MISCELLANEOUS LANGUAGE DOCUMENTATION
INFORMATION RETRIEVAL SYNTAX
GRAMMARS

65B 00403-MF001

Figure 3a. A typical document match against the profile. This document matched on INFORMATION RETRIEVAL. Note that there is also a relationship between SYNTACTIC ANALYSIS in the profile and LINGUISTIC ANALYSIS in the document.

AD-608404. DESCRIPTORS AND COMPUTER CODES USED IN NAVAL ORDNANCE LABORATORY LIBRARY RETRIEVAL PROGRAM. DECEMBER 1964.
DDC

LIBERMAN, E U.S. NAVAL ORDNANCE LABORATORY

AD-608404 NOLTR-64-20

THE DESCRIPTOR AND COMPUTER CODES ARE LISTED SEPARATELY BY SUBJECT, EQUIPMENT DESIGNATIONS (INCLUDING ACRONYMS, TRADE NAMES, CODE NAMES, ETC.) AND CORPORATE AUTHORS, PERSONAL NAMES, AND GEOGRAPHIC PLACE NAMES. THESE DESCRIPTORS HAVE BEEN DEVELOPED OVER A FOUR YEAR PERIOD. THEY PROVIDE A SUBJECT APPROACH TO TECHNICAL REPORTS LITERATURE FOR USE WITH IBM 7090 COMPUTER. THESE DESCRIPTORS CONFORM TO THE AREAS OF LABORATORY INTEREST IN RESEARCH, DEVELOPMENT, TEST, AND EVALUATION IN ORDNANCE AND RELATED FIELDS. 228P.

23-MISCELLANEOUS DESCRIPTORS CODES
INFORMATION RETRIEVAL LIBRARIES

65B 00412-MF001

Figure 3b. A typical document match against the profile. The match here also occurred on INFORMATION RETRIEVAL. In addition there is a close connection between LIBRARY AUTOMATION in the profile and LIBRARY RETRIEVAL in the document.

AD-608574. IS AUTOMATIC CLASSIFICATION A REASONABLE APPLICATION OF STATISTICAL ANALYSIS OF TEXT. AUGUST 1964.
DDC

DOYLE, LB SYSTEM DEVELOPMENT CORP.

AD-608574

THE CRUCIAL QUESTION OF THE QUALITY OF AUTOMATIC CLASSIFICATION IS TREATED AT CONSIDERABLE LENGTH, AND EMPIRICAL DATA ARE INTRODUCED TO SUPPORT THE HYPOTHESIS THAT CLASSIFICATION QUALITY INPROVES AS MORE INFORMATION ABOUT EACH DOCUMENT IS USED FOR INPUT TO THE CLASSIFICATION PROGRAM. SIX NON JUDGMENTAL CRITERIA ARE USED IN TESTING THE HYPOTHESIS FOR 100 KEYWORD LISTS (EACH LIST REPRESENTING A DOCUMENT) FOR A SERIES OF COMPUTER RUNS IN WHICH THE NUMBER OF WORDS PER DOCUMENT IS INCREASED PROGRESSIVELY FROM 12 TO 36. FOUR OF THE SIX CRITERIA INDICATE THE HYPOTHESIS HOLDS, AND TWO POINT TO NO EFFECT. PREVIOUS WORK OF THIS KIND HAS BEEN CONFINED TO THE RANGE OF ONE THROUGH EIGHT WORDS PER DOCUMENT. FINALLY, THE FUTURE OF AUTOMATIC CLASSIFICATION AND SOME OF THE PRACTICAL PROBLEMS TO BE FACED ARE OUTLINED. 34P.

23-MISCELLANEOUS INFORMATION RETRIEVAL
CLASSIFYING INDEXING FILE DOCUMENTATION
COMPUTERS

65B 00420-MF001

Figure 3c. A typical document match against the profile. The match is on both INFORMATION RETRIEVAL and AUTOMATIC CLASSIFICATION.

update their profiles. Thereafter he can monitor program results to make sure that the revisions are producing the desired effect.

The overall statistics offer a good yardstick for measuring system performance. Such figures as number of notifications sent out, percentage returned, current relevance percentage, and quantity of documents requested are all pertinent to evaluating how well the system is operating.

HOW HAS THE SYSTEM PERFORMED TO DATE?

Growth

1. The data base described now includes abstracts of 125,000 documents, with current additions at the rate of approximately 10,000 per year.

2. The number of IBM users of the system is increasing steadily. From the pilot group of 500 professionals participating in Current Information Selection early in 1965, we have expanded to 1700 users. The rate of retrospective searching activity is now up to about 300 searches per month at Yorktown and 100 searches at La Gaude, France, and is increasing.

3. The satellite operation in La Gaude, France, is now offering full service to World Trade Corporation personnel in Europe, supplying both retrospective searching and Current Information Selection to more than 400 customers all over Europe.

User Response Statistics

After a statistical analysis of over 100,000 CIS response cards, we found that the overall proportion of relevance was 79.1% of the returns. This figure was 77% in the first 6 months, and increased to over 80% in the third quarter. Much of the improvement was due to the rewriting or revising of the initial profiles.

Operating Results

1. As expected, the first several months of actual operating experience showed us numerous avenues of improvement, since this was the first effort to correlate user profiles with a natural text searching program. The group had prior experience in writing retrospective search questions. However, we found a noticeable difference in practical strategy between searching a large data base with a few questions and searching a relatively small data base with over a thousand questions (profiles). When we set up the European satellite in September with this experience in back of us, we found that most of the earlier

start-up problems virtually disappeared. The initial statistical results from Europe were very similar to our current domestic figures.

2. Overall, between 12 and 13% of the CIS returns were requests for copies of the complete documents. A much higher percentage of requests (15 to 20%) came from the Data Processing Division and World Trade personnel who were scattered in small locations without direct IBM library facilities. For our largest single group of users in a major IBM complex with library facilities, this figure dropped to about 10%.

3. Only about 5% of the total responses were "Relevant, but have already seen the document." This can be viewed as an encouraging comment on the timeliness of the announcements, as well as the nonavailability of some documents from any other source.

4. The highest percentage of relevance was returned for the data base containing IBM internal documents (technical reports and the like), which was to be expected.

The Importance of Personal Contacts

Even though we are talking about a highly mechanized dissemination system with 1500 customers, we have found that the "personal touch" is extremely important. With a small staff serving so many, it is obviously not possible to talk directly to every user every month. However, the personal conversations and letters that we do have time for have paid dividends in terms of customer satisfaction and participation.

The quarterly statistical run selects and prints out a "trouble-shooting" list. On it are all the users who during the quarter (a) had a high percentage of irrelevant notifications, (b) received none at all, or (c) returned none of their response cards. With this list, the information retrieval specialists look at the operating profiles and often are able to adjust them. If not, they go directly to the users, by telephone if possible. The ensuing discussions usually pinpoint the trouble—a profile needs changing, a user did not realize the function of the response cards, etc.

On a daily basis, all response cards are screened and those with "Comments" punched are sorted out for immediate handling. Some of these request profile changes, or ask questions about the abstracts they have received. Again, the information retrieval specialist checks out the current profile and goes back to the user to make necessary revisions.

The net result of this personal contact is to improve the caliber of the profiles, particularly the problem cases. Each call also has the effect of assuring the user that he is not merely a number in an impersonal computer system. He knows that he can get help to change his profile as needed, that his responses to the system are being monitored by a group of specialists, and that if he has complaints or suggestions they will be acted upon.

CONCLUSIONS

What is ITIRC accomplishing now, and what are the future possibilities? First, it is in full operation, serving many hundreds of users throughout the Corporation. Second, the data base is a broad one, now covering the most critical areas and capable of unlimited future expansion. Third, a single manipulation of input data produces output tailored for a variety of needs—dissemination, announcement, searching, and microfilm. Fourth, the normal text-searching logic that we have developed is an effective technique today—and will be readily adaptable to future developments that can put the entire text of a document into a computer automatically.

Up to now, our efforts have been concentrated on operating a practical system to meet the immediate needs of the IBM engineer-scientist. However, we have not lost sight of the future. Both within and outside IBM, the field of information retrieval will continue to require constant study, research, and development. For these activities, one of the best environments may well be within the framework of a live operating system.

EFFECTS OF QUANTIZATION NOISE IN DIGITAL FILTERS

Bernard Gold and Charles M. Rader
Lincoln Laboratory, Massachusetts Institute of Technology*
Lexington, Massachusetts

GENERAL EXPRESSIONS FOR QUANTIZATION NOISE

If a discrete time linear system, hereafter called a digital filter, is programmed on a digital computer or realized with digital elements, computational errors due to finite word length are unavoidable. These errors may be subdivided into three classes, namely, the error caused by discretization of the system parameters, the error caused by analog to digital conversion of the input analog signal, and the error caused by roundoff of the results which are needed in further computations. The first type of error results in a fixed deviation in system parameters and is akin to a slightly wrong value of (say) an inductance in an analog filter. We shall not treat this problem here; it has been treated in some detail by Kaiser.[1] The other two sources of error are more complicated but if reasonable simplifying assumptions are made they can be treated by the techniques of linear system noise theory.[2] It is our aim to set up a model of a digital filter which includes these two latter sources of error and, through analysis of the model, to relate the desired system performance to the required length of computer registers.

Both analog to digital conversion and roundoff may be considered as noise introducing processes, very similar in nature. In each case a quantity known to great precision is expressed with considerably less precision. If the digitized or rounded quantity is allowed to occupy the nearest of a large number of levels whose smallest separation is E_0, then, provided that the original quantity is large compared to E_0 and is reasonably well behaved, the effect of the quantization or rounding may be treated as additive random noise. Bennett[3] has shown that such additive noise is nearly white, with mean squared value of $E_0^2/12$. Furthermore the noise is reasonably assumed to be independent from sample to sample, and roundoff noises occurring due to different multiplications should be independent. It is possible to show pathological examples which disprove each of these assumptions, but they are reasonable for the great majority of cases. Ultimately our results must rest on experimental verification, of course.

Since the noise of A-D conversion is assumed independent of the noise created by roundoff, we can compute the output of any filter due to either excitation alone, or due to the signal alone, and combine them to get the true filter output (of course the noise terms are known only statistically); therefore, we will begin by finding an expression for the mean squared output of an arbitrary filter excited by a single noise source. Let the filter function be $H(z)$; it is understood that $H(z)$ is the transfer function between the output of the filter and the node where noise is injected; $H(z)$ may thus be different from the transfer function between the filter's normal input and output. Let us thus consider the

*Operated with support from the U.S. Air Force.

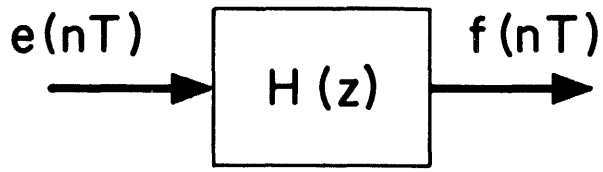


Figure 1. Random noise applied to a filter.

situation of Fig. 1, where a given noise sequence $e(nT)$ is applied to $H(z)$, resulting in an output noise sequence $f(nT)$.

We can conveniently examine this model using the convolution sum. Thus,

$$f(nT) = \sum_{m=0}^{n} h(mT)e(nT - mT) \qquad (1)$$

where $h(mT)$ is the inverse z transform of $H(z)$. The input noise $e(nT)$ is presumed to be zero for $m < 0$ and the system is initially at rest. Squaring Eq. (1) yields

$$f^2(nT) = \sum_{m=0}^{n} \sum_{l=0}^{n} h(mT)h(lT) \times e(nT - mT)e(nT - lT) \qquad (2)$$

Now, if $e(nT)$ is a random variable with zero mean and variance σ^2, and recalling our assumption that $e(nT)$ is independent from sample to sample, the statistical mean of Eq. (2) reduces to

$$E[f^2(nT)] = \sigma^2 \sum_{m=0}^{n} h^2(mT) \qquad (3)$$

For a system for which the right side of (3) converges, the steady state mean squared value of $f(nT)$ can be obtained by letting n approach infinity. For this case, a formula which is usually more convenient can be obtained in terms of the system function $H(z)$. Noting the definition.

$$H(z) = \sum_{m=0}^{\infty} h(mT)z^{-m} \qquad (4)$$

of the z transform, we can form the product $H(z)$ $H\left(\frac{1}{z}\right)z^{-1}$ and, by performing a closed contour integration in the z plane within the region of convergence of both $H(z)$ and $H\left(\frac{1}{z}\right)$, arrive at the identity

$$\sum_{m=0}^{\infty} h^2(mT) = \frac{1}{2\pi j}\oint H(z)H\left(\frac{1}{z}\right)z^{-1}dz \qquad (5)$$

Either the right- or left-hand side of (5) may be used to evaluate the steady state mean squared value of $f(nT)$.

EXAMPLE—FIRST ORDER SYSTEM

As an example, consider the first order system of Fig. 2. Let the analog-digital conversion noise $e_1(nT)$ have variance σ_1^2 and the roundoff noise $e_2(nT)$ have variance σ_2^2. The system function $H(z)$ of Fig. 2 is given by $1/(1 - Kz^{-1})$ and $h(mT) = K^m$. The output $y(nT)$ can be expressed as the sum of a signal term $y_0(nT)$, caused by $x(nT)$, and a noise term $f(nT)$, whose mean squared value can be written, from (3), as

$$E[f^2(nT)] = (\sigma_1^2 + \sigma_2^2) \sum_{m=0}^{n} (K^m)^2 \qquad (6)$$

from which the steady state value can be instantly written as

$$\sigma_n^2 = \lim_{n\to\infty} E(f^2(nT)) = \frac{(\sigma_1^2 + \sigma_2^2)}{1 - K^2} \qquad (7)$$

The implications of Eq. (7) are tricky. The mean squared value of the noise clearly increases as K approaches unity. The maximum gain of the filter also increases (the gain of the system of Fig. 2 at dc is $(1/(1 - K))$. For this filter with low frequency input the signal power to noise power ratio (S^2/N^2) is proportional to $(1 + K)/(1 - K)$ which approaches infinity as the pole of the filter approaches the unit circle. This is a general result. However, with a finite word length, the input signal must be kept small enough that it does not cause overflow

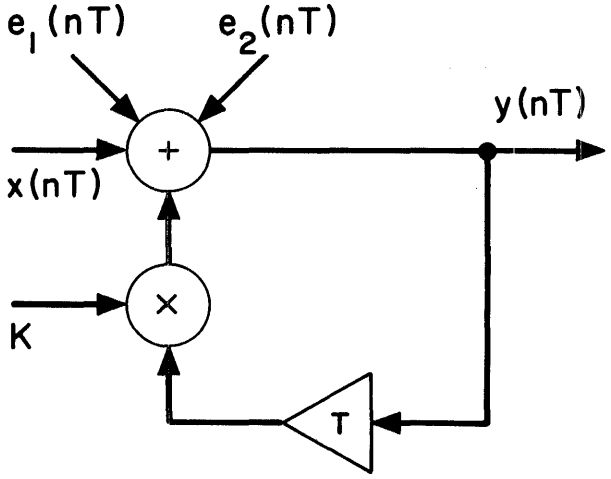


Figure 2. Noise mode for first order system.

in the computation. Thus, the obtainable signal-to-noise ratio decreases as K approaches unity. Clearly, each case deserves its own considerations, as the signal-to-noise ratio in the filter depends very much on the actual conditions of the use of the filter.

Finally, we comment that the cases $K = 0$, $K = 1$, in Eq. (7) are unique because σ_2 becomes zero since no multiplications are performed.

EFFECT OF DIFFERENT REALIZATIONS OF THE SAME FILTER

There are a variety of ways of programming a second order digital filter (or in general a filter with more than two singularities). Suppose a particular system function $H(z)$ is desired. If quantization is ignored, then only the relative speed and memory requirements of the different methods are of interest in deciding which way to use. However, Kaiser's work shows that the truncation of system constants affects different realizations differently, and may in fact lead to instability in some realizations. The noise effects described here also yield different results for different programming configurations. The point is illustrated through the examination of the two systems of Fig. 3. Fig. 3a represents a noisy programmed realization of the difference equation:

$$y(nT) = 2r\cos bT\,y(nT - T) - r^2 y(nT - 2T) + x(nT) - r\cos bT\,x(nT - T) \qquad (8)$$

and Fig. 3b represents the pair of simultaneous difference equations:

$$\left.\begin{aligned} w(nT) &= x(nT) + 2r\cos bT\,w(nT - T) - r^2 w(nT - 2T) \\ y(nT) &= w(nT) - r\cos bT\,w(nT - T) \end{aligned}\right\} \qquad (9)$$

Both systems have the transfer function

$$H(z) = \frac{1 - r\cos bT\,z^{-1}}{1 - 2r\cos bT\,z^{-1} + r^2 z^{-2}}$$

By examination of the poles and zeros of $H(z)$ in Fig. 4, we see that our network behaves as a resonator tuned to the radian center frequency b for the sampling interval T.

In Figs. 3a and 3b, $X(nT)$ represents the noiseless input to the filter, $e_1(nT)$ represents the noise due to A-D conversion of the input, and $e_2(nT)$ represents the noise added by rounding. The roundoff noise can be caused either by a single roundoff after all products are summed, or by the sum of the roundoff error due to each of the multiplications. It is simpler to program the latter, but more noise is created. Note that, while in the realization of Fig. 3b the noise terms $e_1(nT)$ and $e_2(nT)$ are injected into the filter at the same place as the input $X(nT)$ and thus see the same transfer function $H(z)$, in Fig. 3a the noise term $e_2(nT)$ is injected in a different part of the filter and sees a different transfer function:

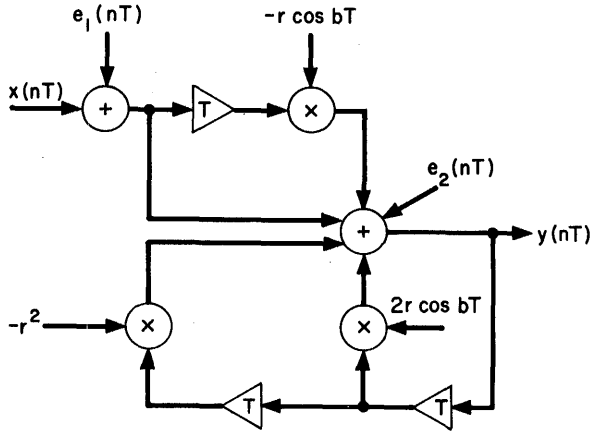


Figure 3a. Noise model for second order system—direct realization.

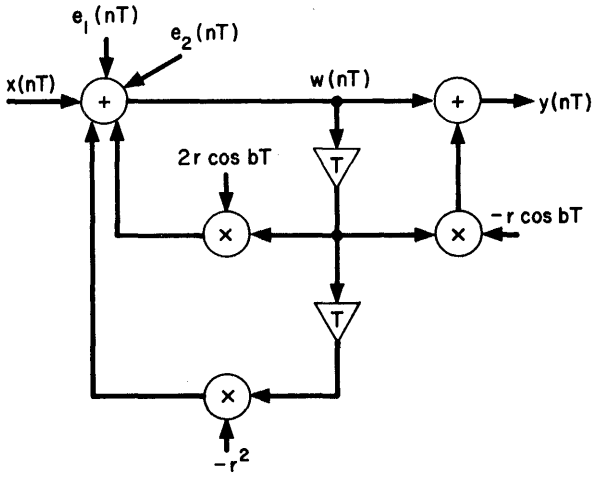


Figure 3b. Noise model for second order system—canonical realization.

$$H_1(z) = \frac{1}{1 - 2r\cos bT\,z^{-1} + r^2 z^{-2}} \qquad (10)$$

Thus we can expect that the noise due to $e_2(nT)$ will be different for the filters of Figs. 3a and 3b.

Considering first the realization of Fig. 3a, we can, after some manipulation, obtain the result,

$$\sigma_n^2 = \sigma_1^2 u_1(r,bT) + \sigma_2^2 u_2(r,bT) \qquad (11)$$

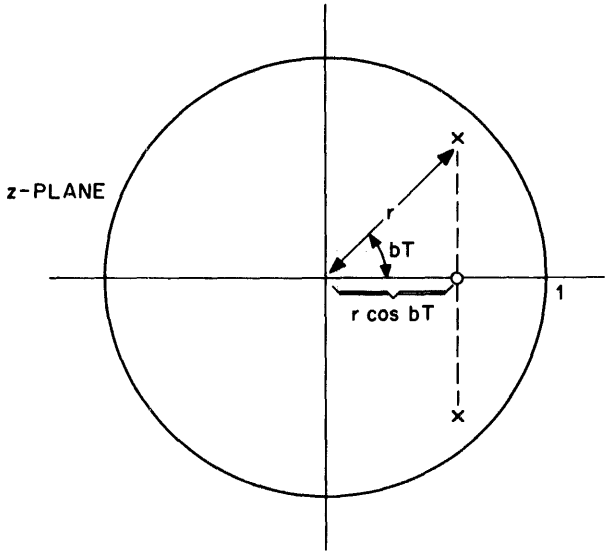


Figure 4. Pole zero representation of Eqs. (8) or (9).

where σ_1^2 and σ_2^2 are the variances of $e_1(nT)$ and $e_2(nT)$, and with

$$u_2 = \frac{1 + r^2}{1 - r^2} \times \frac{1}{r^4 + 1 - 2r^2 \cos 2bT}$$

and

$$u_1 = \frac{1}{1 - r^2}\left[1 - \frac{r^2 \sin^2 bT(1 + r^2)}{r^4 + 1 - 2r^2 \cos 2bT}\right]$$

More insight can be obtained into these results by letting $r = 1 - \epsilon$ and allowing ϵ to be quite small, of the order of 0.05 or less. Then (11) reduces to the simple form

$$\sigma_n^2 = \frac{1}{4\epsilon}\left[\sigma_1^2 + \frac{\sigma_2^2}{\sin^2 bT}\right] \tag{12}$$

Carrying through a similar computation for the realization of Fig. 3b yields

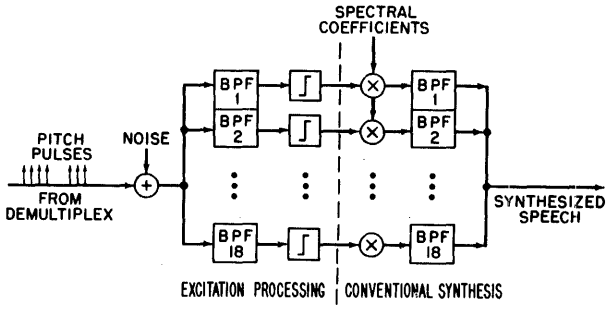
$$\sigma_n^2 = (\sigma_1^2 + \sigma_2^2)u_1(r,bT) \tag{13}$$

which can also be reduced, for small values of ϵ, to

$$\sigma_n^2 = \frac{1}{4\epsilon}[\sigma_1^2 + \sigma_2^2] \tag{14}$$

Several important facts can be deduced from Eqs. (12) and (14). First, the so-called "straightforward" realization of Fig. 3a leads to increased noise for low resonance frequencies whereas the "canonic" realization of Fig. 3b does not. Physically, this result can be explained by noting that, in the straightforward realization, the noise "passes through" only the poles of the filter, so that at low frequencies, the complex conjugate poles interact to form a low pass filter. In the "canonic" realization the noise is also filtered by a zero which is close to dc and thus the output noise is of a band-pass nature and less total energy is able to pass through the filter. Second, we note that Eqs. (13) and (14) have the same functional dependence on pole positions, namely, that the mean squared output noise is inversely proportional to the distance from the pole to the unit circle and therefore directly proportional to the gain of the filter.

From these results one can, for example, estimate the word length needed for a simulation requiring many filters. One such system is a vocoder synthesizer shown in Fig. 5. Typically, a vocoder synthesizer will contain about 100 resonators. Assuming that the noise from each resonator is additive to the noise from all other resonators and picking an effective average ϵ of 0.01, we arrive at a total noise output of about 7 or 8 bits. It is clear that word lengths of at least 20 bits are needed to avoid audible noise outputs superimposed on the vocoder generated synthetic speech.



VOCODER SYNTHESIZER

Figure 5. Vocoder synthesizer.

EXPERIMENTAL VERIFICATION FOR FIRST AND SECOND ORDER FILTERS

The results of the preceding computations were experimentally verified by programming various realizations of first and second order difference equations on the TX-2 digital computer. To perform a measurement of output noise for a given digital filter, the computations were performed with rounded arithmetic using a 36-bit word, and simultaneously, using rounded arithmetic with a shorter word and exactly the same input. The outputs of

the two filters were subtracted, squared, integrated and divided by the number of iterations of the equation. The inputs to the filters were random noise or sampled sinusoids. The filters were programmed using the PATSI[4] compiler, and the various waveforms of interest, including the mean squared output noise, were displayed during the computation. The measurement was taken when the mean squared output noise seemed to reach a steady value, or in the case of the very high gain filters, when the patience of the observer was exhausted. As we shall see below, the necessary observation time for confidence in such a measurement is highly dependent on the gain of the filter.

Figure 6 shows the predicted and measured output noises for some one-pole filters, as Eq. (7), with $\sigma_1^2 = 0$. The horizontal axis is the pole position and the vertical axis is the mean squared output noise normalized to σ_2^2. Table 1 gives the predicted versus measured output noises for several two-pole filters (no real zeros) with various pole positions, along with the measurement error. All of the results seem to confirm the theory.

It is advisable to determine, on a statistical basis, the measurement time required before the variance of such statistical observations is sufficiently small. Thus, consider a random variable q defined as

$$q = \frac{1}{n} \sum_{m=0}^{n} f^2(nT) \tag{15}$$

where $f(nT)$ is an output noise signal as indicated in Fig. 1 due to a set of mutually independent input noise samples $e(nT)$.

Assuming $f(nT)$ to have zero mean, we can immediately perceive that the mean value of the measurement q is given by

$$E[q] = \sigma_f^2 \tag{16}$$

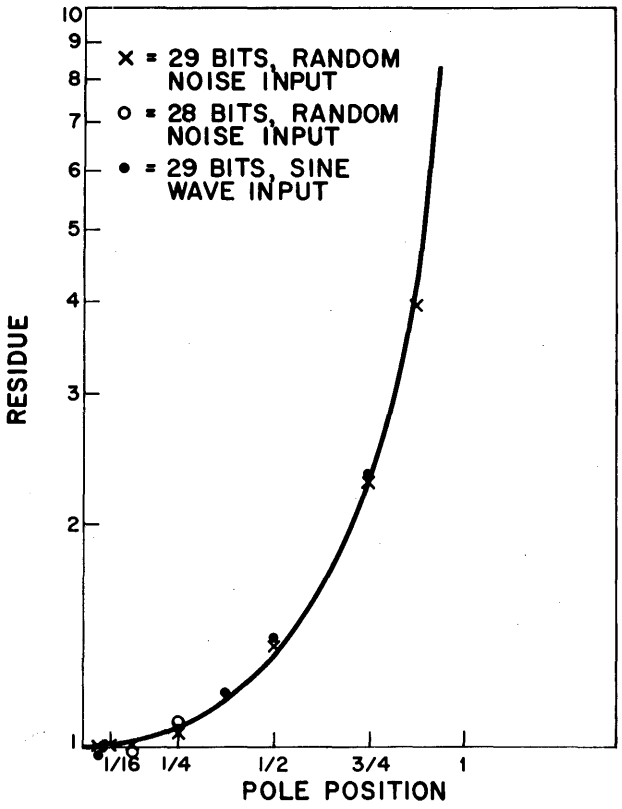


Figure 6. Predicted vs measured quantization noise for first order system.

Table 1. Two-Pole Filter Noise Measurement

Mean Squared Output Noise Predicted	Measured	Error %	Pole Positions
204	203	0.49%	.5 ± .5j
289	297	2.77	.5 ± .707j
508	520	2.36	.5 ± .778j
1011	1058	4.65	.75 ± .56j
2824	2880	1.98	.875 ± .332j
5553	5933	6.40	.90625 ± .235j
5553	5503	0.90	.90625 ± .235j
11014	11450	3.96	.921875 ± .169j
11014	11079	0.59	.921875 ± .169j
3306	3740	13.12	.75 ± .654j
3306	3359	1.60	.75 ± .654j

where σ_f^2 is the variance of the (stationary) random variable $f(nT)$. Now assuming that $f(nT)$ is a set of stationary Gaussian variables with correlation coefficient $\rho(rT)$, then it can be shown that[5]

$$E[f^2(mT)f^2(lT)] = \sigma_f^4 + 2R^2(mT - lT) \quad (17)$$

where R^2 is defined as the covariance between $f(mT)$ and $f(lT)$. From Eqs. (16) and (17), we arrive at the expression for the variance of q,

$$\sigma_q^2 = E[q^2] - E^2[q]$$

$$= \frac{2}{n^2} \sum_{m=0}^{n} \sum_{l=0}^{n} R^2(mT - lT) \quad (18)$$

This can be evaluated for first order system of Fig. 2. For that case $R(mT - lT) = K^{2|m-l|}$ and, for large n, Eq. (18) reduces to

$$\sigma_q^2 = \frac{4\sigma_e^4}{n(1 - K^2)^3} \quad (19)$$

where σ_e^2 is the variance of the input $e(nT)$ as in Fig. 1. Of major interest in determining the time needed to perform the measurement is the ratio of the standard deviation to the mean of q. Using an argument similar to the one that leads to Eq. (7) we can for the first order system relate σ_e^2 to σ_f^2 by the formula $\sigma_f^2 = \sigma_e^2/(1 - K^2)$, which combined with Eqs. (19) and (16) yields

$$\frac{E[q]}{\sigma_q} = \frac{(1 - K^2)\sqrt{n}}{2} \quad (20)$$

Thus, for example, if $K^2 = 0.99$, we need 10^8 terms in the measurement of Eq. (15) in order to reduce the standard deviation of the measurement to 2% of the mean of the measurement. Assuming that an iteration could be done in 100 μsec, 10^4 seconds would be required for such accuracy.

NOISE CONSIDERATIONS IN PROGRAMMING ITERATIVE SINE WAVE GENERATORS

One must be especially attentive to noise considerations in the programming of iterative sine wave generators. Various efficient routines exist to compute the sine or cosine of a random argument rapidly, but for instances where the argument is nT for successive integers n, the most efficient way to generate sinusoidal functions is by the use of iterative difference equations. These are, of course, digital filters with poles directly on the unit circle, inputs equal to zero, and initial conditions which specify the magnitude and phase of the output. Since the poles of the filter are directly on the unit circle, the noise, according to Eq. (12) or (14) becomes infinite. This is indeed the situation.* The saving feature is the *gradual* increase of the noise term, so that if one runs the program for a limited time, or periodically resets the initial conditions, catastrophe can be avoided. To study this problem theoretically, consider the simultaneous difference equations

$$\left.\begin{aligned} y(nT + T) &= \cos bT\, y(nT) \\ &\quad + \sin bT\, x(nT) \\ x(nT + T) &= -\sin bT\, y(nT) \\ &\quad + \cos bT\, x(nT) \end{aligned}\right\} \quad (21)$$

with initial conditions $x(0) = 1$, $y(0) = 0$. The "circuit" is shown in Fig. 7.

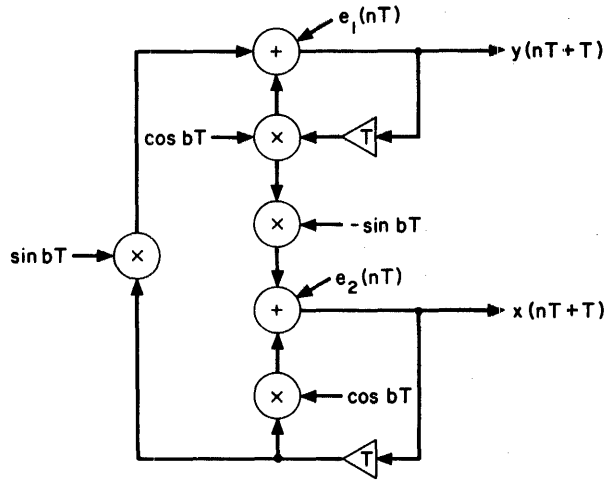


Figure 7. Iterative sine and cosine generator.

The z transform $X(z)$ of one output $x(nT)$ can be written

$$X(z) = \frac{z^2 - z\cos bT + zE_2(z) - \cos bTE_2(z) - \sin bTE_1(z)}{z^2 - 2z\cos bT + 1} \quad (22)$$

We see that the first two terms of the numerator correspond to the signal and the remaining terms to the noise, $E_1(z)$ and $E_2(z)$ being respectively the z

*Various nonlinearities can be introduced to keep the noise finite. This is adequate for many applications since the selectivity of the filter can be relied on to keep the output spectrally pure even if the phase of the output is unpredictable.

transforms of the added noises $e_1(nT)$ and $e_2(nT)$, both introduced by roundoff error.

Defining:

$$h_1(nT) = Z^{-1}\left\{\frac{z - \cos bT}{z^2 - 2z\cos bT + 1}\right\}$$
$$h_2(nT) = Z^{-1}\left\{\frac{-\sin bT}{z^2 - 2z\cos bT + 1}\right\} \quad (23)$$

where Z^{-1} is the inverse z transform, we can from Eq. (3) write the total noise as

$$E(f^2(nT)) = \sigma_1^2 \sum_{m=0}^{n} h_1^2(nT) + \sigma_2^2 \sum_{m=0}^{n} h_2^2(nT) \quad (24)$$

Solving Eq. (23) explicitly and letting $\sigma_1^2 = \sigma_2^2 = \frac{E_0^2}{12}$ we arrive at the result

$$E(f^2(nT)) = \frac{E_0^2}{12}\left\{\sum_{m=0}^{n} \cos^2(nbT - bT) + \sin^2(nbT - bT)\right\} = \frac{E_0^2}{12} n \quad (25)$$

Notice that is was impossible to use Eq. (5), since the result obtained would be infinite and thus no time-dependent result could be formulated. Equation (25) tells us that the noise increases linearly with the number of iterations of the difference equations. For example, after 10^6 iterations, the noise is about 10 bits. Assuming that one iteration is performed in 100 μsec, several minutes could certainly pass, even in an 18-bit machine, before the generated sine and cosine waves begin to look noisy.

Another program for generating a cosine wave is expressed by the iteration

$$y(nT + 2T) = 2\cos bT\, y(nT + T) - y(nT) \quad (26)$$

with initial conditions $y(0) = 1$, $y(T) = \cos bT$. Noise analysis of Eq. (26) leads to a functional dependence of the mean squared noise, of the form $\frac{n}{\sin^2 bT}$; thus appreciably greater quantities of noise are generated at low frequencies, and fewer iterations are available before the program becomes unusable.

The comparison of Eqs. (21) and (26) was performed qualitatively on TX-2 by programming identical sine wave generators using both methods. For all frequencies, the method of Eq. (21) produced sinusoids of more nearly constant amplitude than the method of Eq. (26), but this difference in behavior was negligible for frequencies greater than one fourth of the sampling frequency, and, using 36-bit arithmetic, the distortions were almost unobservable for these frequencies. For low frequencies (of the order of one thousandth of the sampling rate) the method of Eq. (26) was completely unusable, with the generated sine wave being terribly distorted in the first period.

REFERENCES

1. J. F. Kaiser, "Some Practical Considerations in the Realization of Linear Digital Filter," 3rd Allerton Conference (Oct. 20–22, 1965).

2. J. B. Knowles and R. Edwards, "Effect of a Finite-Word-Length Computer in a Sampled-Data Feedback System," *Proc. IEEE,* vol. 112, no. 6, (June 1965).

3. W. R. Bennett, "Spectra of Quantized Signals," *Bell System Technical Journal,* vol. 27, pp. 446–472 (July 1948).

4. C. M. Rader, "Speech Compression Simulation Compiler," *J. Acoust. Soc. Am.* (A), June 1965.

5. J. L. Lawson and G. E. Uhlenbeck, *Threshold Signals,* MIT Rad. Lab. Series 24, McGraw-Hill, New York, 1950.

A REAL-TIME COMPUTING SYSTEM FOR LASA

H. W. Briscoe and P. L. Fleck
Lincoln Laboratory, Massachusetts Institute of Technology*
Lexington, Massachusetts

PHYSICAL DESCRIPTION OF LASA

The Large Aperture Seismic Array (LASA) consists of 525 vertical motion seismometers installed in an area of approximately 10,000 square miles in eastern Montana. Each seismometer is located at the bottom of a 200-foot borehole to reduce noise generated by wind, traffic, and livestock. The seismometers are grouped in clusters (subarrays) of 25 seismometers each, and the 21 subarrays are arranged in a series of successively inscribed squares (see Fig. 1) to produce a logarithmic density taper.

The data from each subarray of 25 seismometers are collected in a buried vault at the center of the cluster where they are low-pass filtered to avoid aliasing and are digitized at 20 samples per second. The frequency passband of the seismometers, wellhead amplifiers, and low-pass filters is approximately 0.5 to 5 cycles per second. Figure 2 shows the inside of one of the buried vaults including the rack of equipment for filtering, multiplexing, and digitizing the signals and a second rack containing digital phone-line modulating equipment used to transmit the digital data over open wire and microwave links to the LASA Data Center (LDC) in Billings, Montana.

The Data Center, shown in Fig. 3, contains general purpose and special purpose digital processing equipment which is used for recording and processing data from the entire array. The data is recorded on standard ½-inch 7-channel digital tape in a format that is compatible with most commercially available processing equipment. The data rate is such that a full 2500-foot reel of tape is written every 6 minutes. In order to reduce the array output to a manageable level, it is imperative that some on-line processing be done. Since significant seismic events appear at the array as discreet bursts of energy lasting from one minute to several hours spread throughout the day, this on-line data reduction for LASA consists of selecting those sections of recordings which contain data from interesting seismic locations. Thus, the on-line processing, part of which takes place in the same general-purpose processor that is used for formatting and recording the digital tapes, consists of 1) predetection processing to improve the signal-to-noise ratio for detection, 2) detection of teleseismic events, 3) source location of detected events, and 4) testing various remote components in the system. Events occurring within preselected regions are then further processed off-line either on-site or at remote processing centers. In this paper, we will be primarily concerned with the first two steps in the on-line processing; predetection processing and detection.

PREDETECTION PROCESSING TECHNIQUES

Predetection processing uses the LASA as a wideband phased array antenna to improve the signal-

*Operated with the support of the U.S. Advanced Research Projects Agency.

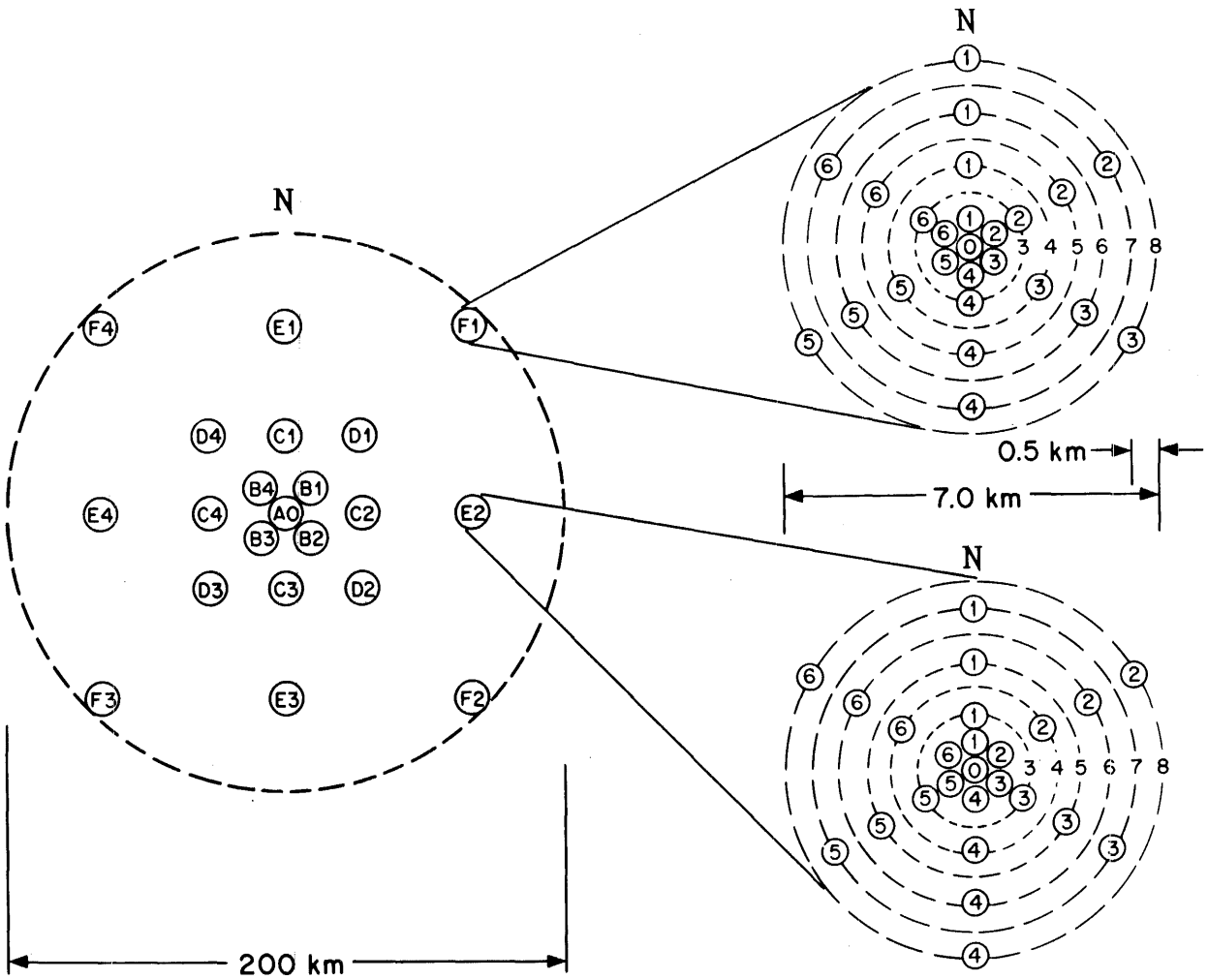


Figure 1.

to-noise ratio for detection of small teleseismic events. In a conventional phased array, this is accomplished by appropriately delaying (phasing) and adding the signals from each element in the array. The phasing delay is adjusted so that the signals add coherently and the noises add incoherently. Delaying the data from the elements of the array is equivalent to pointing a conventional antenna in a certain geometric direction in space. Since seismic noise tends to come from discrete sources with discrete velocities, the processing can be further improved; strong noise sources can be suppressed by selecting gains or amplitude weights (the antenna "taper"), which place nulls of the antenna sidelobe pattern in the proper direction. Thus the noise does not add with random phase but with a controlled anti-phase. If the noise is allowed to add with random phase, the antenna gain in signal-to-noise ratio should be approximately the square root of the number of elements, but this gain may be much greater if the strong sources of organized noise can be specifically rejected.

Since the LASA is designed to receive wideband signal energy (the passband includes more than an octave) and since many seismic noise sources emit energy over much narrower bands in frequency, additional signal-to-noise gain can be obtained by optimum frequency filtering. In fact, one of the most powerful techniques for combining data from the elements of a seismic array involves employing a different set of amplitude weights (a different taper) at each of several frequencies in order to optimize the use of sidelobe nulls at each frequency. Varying the amplitude gain on a single element as

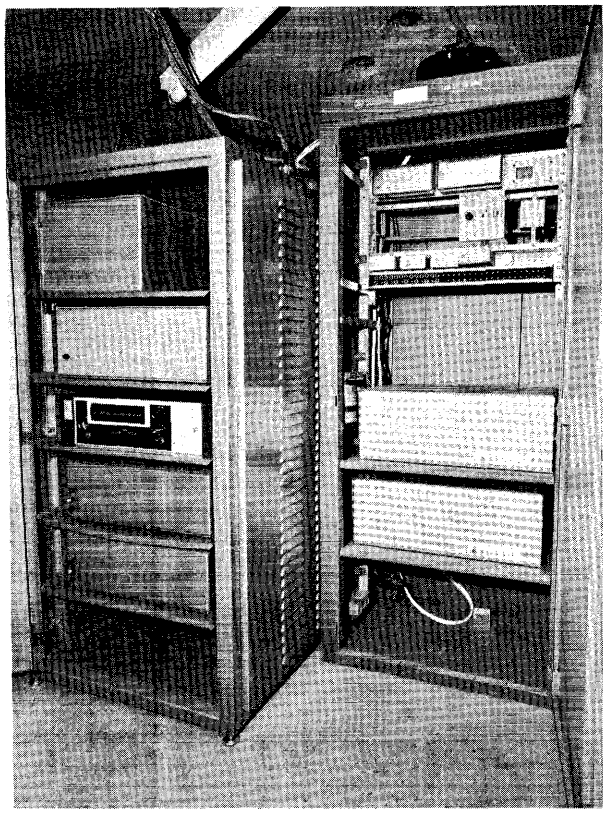


Figure 2.

a function of frequency is the same as filtering the data from that element. It has been shown that the optimum design for combining data from the elements of an array of seismometers consists of applying a different filter and time delay to data from each element and adding the resulting data from all the elements. Most other linear processing techniques are degenerate forms of this "filter, delay and sum" processing. For the detection of seismic signals it is often advantageous to further filter the data through a narrow bandpass filter. On-line processing for the LASA involves the use of both convolution and recursive digital filtering and will be described in more detail in a later section of this paper.

ON-LINE PROCESSING FOR THE LASA

We now turn our attention to the specific implementation of the processing concepts described above. The functions to be performed by the on-line processing facility are:

1. Read data into memory.
2. Write data onto magnetic tape.
3. Form 10 beams.
4. Make eight event detectors.
5. Locate source of events.
6. Test components in the system.
7. Output data for monitoring.

First, all the data is read into core memory of the general-purpose computer. That is, every sampling interval (50 msec), each seismometer waveform is digitized, multiplexed, and transmitted to the core memory.

Second, all this data is written onto magnetic tape. Each record on tape contains two samples from each sensor (100 msec of data).

Third, approximately 10 processed outputs are

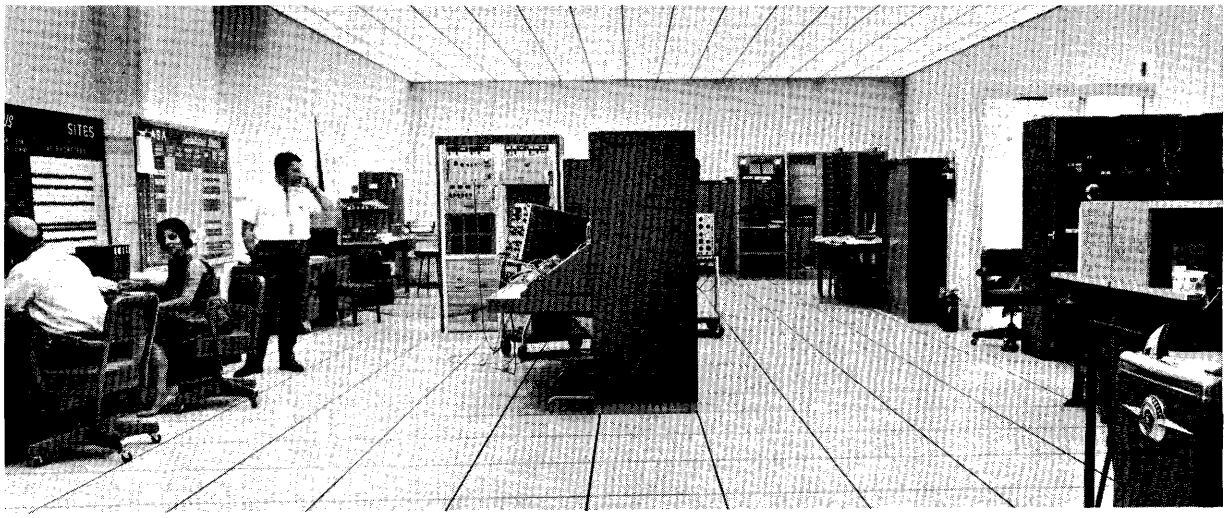


Figure 3.

formed. Five of these are formed using the delay-and-sum method to steer the entire array to monitor five predetermined locations. The other five are formed using the more powerful filter-and-sum method on up to 25 channels to produce five overlapping beams to monitor the entire region of the world from approximately 20° distance to 100° distance. All these beams have a predetection signal-to-noise ratio improvement which allows detection of weaker events in the areas being monitored.

Fourth, several independent event detectors are connected to selected channels from widely separated sensors or beam outputs. The event detectors output the GMT time of any event occurring in their input waveforms.

Fifth, the outputs from event detectors connected to widely separated sensors are put into a source location program which determines where the event originated. On the basis of the location of detected events, a decision to save or reuse the tapes is made.

Sixth, the general-purpose computer periodically tests each component in the system and types out its status so that maintenance teams can be dispatched to repair faults as they occur.

Seventh, the computer provides several on-line analog outputs for monitoring the system performance.

Figure 4 shows the equipment configuration used to accomplish these tasks on-line. The two general-purpose computers are identical machines. Each machine has a 16,384 eighteen-bit word memory with a 1.75 microsecond cycle time. The tape units shared by the two machines are standard seven-track IBM compatible drives operating at 75 inches per second with character density of 800 characters per inch. The special-purpose processor (usually referred to as the "Multi-Channel Filter" or MCF) is designed to perform filter-and-sum processing with 25 input channels and up to five processed output channels. The data displays indicated in the diagram provide the on-line monitoring of the system.

Now we shall discuss in more detail how these points are accomplished. The real-time program-

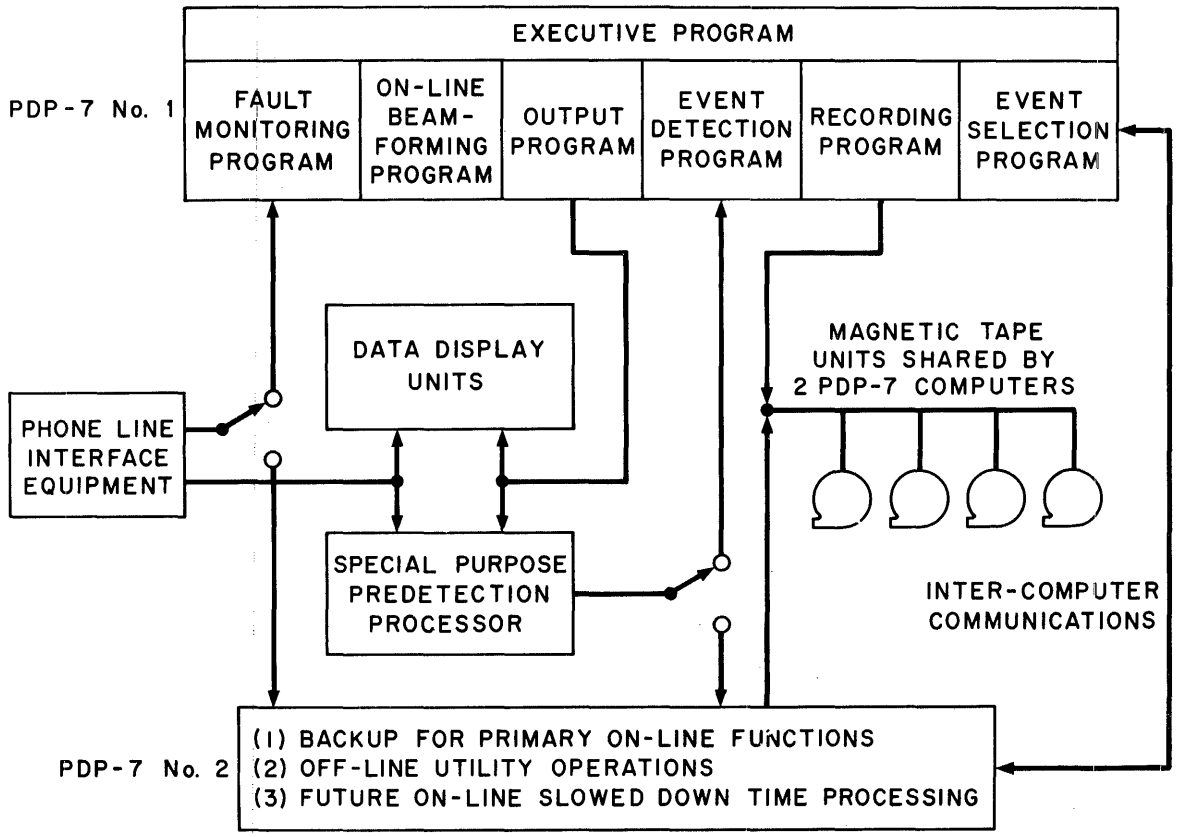


Figure 4.

ming system is timed by an interrupt pulse which occurs every 50 msec whenever a new set of data samples is ready to be read into the memory. At this time the program starts a block transfer which automatically reads the data into 651 consecutive locations. When the next data ready interrupt comes, the computer reads the data into the next 651 consecutive locations and starts another block transfer which writes out onto magnetic tape the data stored from the start of the first block to the end of the second block. The data rates of the input and output are just right so both these data transfers interleave without running into each other.

While the data are being transferred into core and onto tape, the main program is processing the data. All the processing is subject to the following constraints: It must operate on each data sample as it comes in; the total time for all the processing must be less than 50 msec; and everything must fit into the remaining memory not used for input/output buffering.

The first waveform processing we do is to form five delay-and-sum beams. This is shown in Fig. 5 where the h(t)'s represent simple delay lines. A teleseism from an interesting area of the world can take up to 15 seconds to propagate across the array so the simple minded method of putting 15-second delay lines (300 locations) for each data channel would take all the remaining core memory. Instead, the delay for each particular channel is used as an index to tell into which sum box each data sample is to be added. This way only 300 locations are used for each beam, and each data sample is only used once. This program takes 5 msec (about 3000 cycles) and occupies about 2400 locations in core.

The other five processed outputs are formed by filter-and-summing in the MCF special purpose

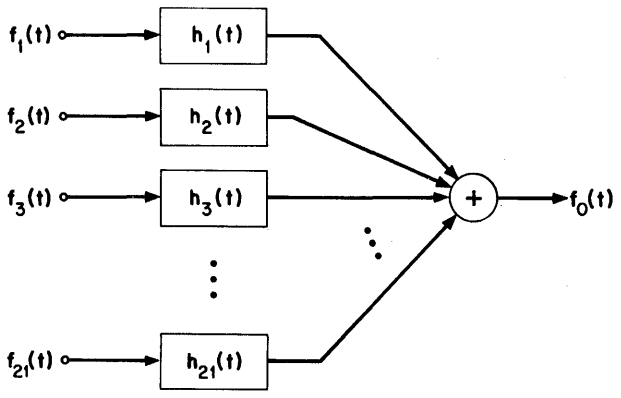


Figure 5.

computer. Figure 5 also shows this processing, but here the h(t)'s represent linear filters. Each output is formed by passing each of the 25 input channels through a different filter and summing the filtered waveforms. The 25 input channels for the MCF can come directly from a single subarray of seismometers via the phone-line interface, or from the general-purpose computer which can form clusters of widely separated seismometers. The MCF interrupts the program in the central computer each time an output sample is ready. The interrupt answering program accepts the sample, stores it in core memory, and clears the interrupt.

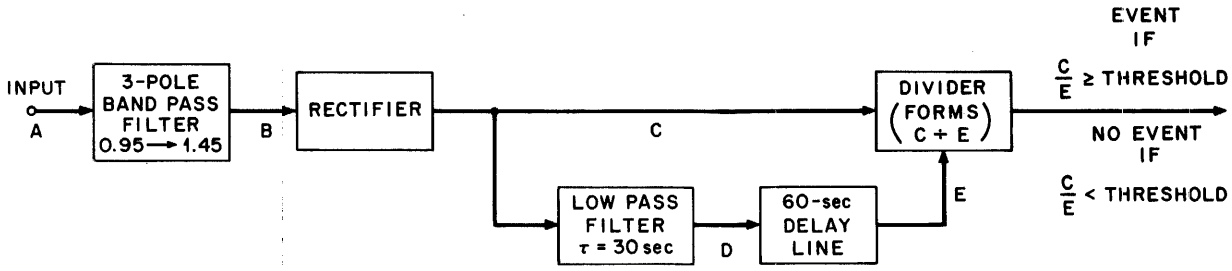
The actual operations of this special-purpose computer are:

$$f_n = \sum_{j=1}^{25} \sum_{i=0}^{78} h_{i,j} f_{n-i,j}$$

where f_n = the output at time n,
$f_{n-i,j}$ = the ith most recent data input from the jth channel, and
$h_{i,j}$ = the impulse response for the jth channel.

Each filter is a 78-point convolutional filter. That is, the last 78 data samples from each input are stored in memory and every sampling interval (50 msec) there are 78 multiplications between these data and 78 prestored "filter constants" with the accumulated sum of the product pairs being the output sample of one filter. Then the 78 inputs are pushed down with the new data sample being stored at the head of the list and the 78th one dropped. All this is repeated 25 × 5 or 125 times, making a total of 9750 multiplications and 9875 additions. A trick is used to get all these operations done in 50 msec. The memory cycles that get the filter points are standard read-restore cycles, but the cycles that get the data points are read-save-restore-previous-data sample, so that the "push down list" can be done in essentially no extra time. Phasing delays can be introduced by using filter response functions less than 78 points long and adding appropriate number of zeros at the beginning and end of each response function.

The next processing step consists of event detection on selected channels. Figure 6 shows a block diagram of one event detector. First, the input waveform is passed through a bandpass filter and the output energy is measured. If this energy should suddenly increase over what it had been, the detector decides there has just been an event and the current GMT time is typed out, along with the channel number that triggered the detector.



AUTOMATIC EVENT DETECTOR

Figure 6.

Each pole in the bandpass filter is of the recursive type as opposed to the convolutional type described earlier. This is shown in Fig. 7. To synthesize a transfer function with two complex conjugate poles and a zero at the origin of the S plane (a RLC filter), we use a recursion formula where the present value of the filter output is a linear combination of the two previous filter outputs and the first difference of the input waveform. The difference equation is shown in the figure. This has the advantage of requiring only 15 registers in memory for the constants per pole, independent of the ringing time of the filter.

If we look again at Fig. 6, we see that the event detector uses three recursive filters in cascade in order to provide very rapid attenuation of low frequency microseismic noise energy. The three poles are synchronously tuned to resonate at the same frequency so that the shape of the filter makes an approximate match to the spectrum of the energy of a teleseism. This is done in order to maximize the signal-to-noise ratio at point B.

The passband of the filter is 0.95 cps to 1.45 cps because our investigations of many weak teleseisms showed this to be the frequency where the energy lies for short-period vertical seismometers. This passband is very effective in eliminating local events which have energy at frequencies much higher than this.

This filter output is then rectified so that the signal at point C is an approximation to the power of the signal at point B. The waveform is rectified instead of being squared for two reasons. First, it takes less time to compute, and second the dynamic range that the following blocks require is considerably reduced.

Now, following the lower path, the signal is passed through a low-pass filter whose time constant is 30 seconds. This is long enough to smooth the fluctuations caused by the seismic signal, yet short enough so that it will accurately follow any long-term variation in the noise power and system calibration. Thus, at point D we have a slowly-varying signal which is proportional to the background noise level in the input seismic signal. This signal is then passed through a 60-second delay line such that its output at E is simply the signal at D delayed in time by one minute.

Now, the signal at C, which is proportional to the energy in the frequency band 0.95 to 1.45 cps is divided by the average background noise energy in the same frequency band one minute earlier. We say there is an event if this ratio exceeds a given threshold; otherwise there is no event. The reason for the 60-second delay line becomes clear if you consider what would happen if it weren't there and a slowly emerging event should come in. Both the signals at C and E would rise together and their ratio would stay constant. Clearly, putting a delay line in circumvents this problem.

If the threshold is set too low, the event detector will have a high false alarm rate—that is, it will be constantly triggering on every little noise pulse that comes along. On the other hand, if the threshold is set too high, only very strong events will trigger the event detector. We have varied the threshold and empirically decided to make it 5.82. With this value and pure gaussian noise as an input, seven false alarms are registered per day on the average. Actually, however, the seismic signals we use are not gaussian noise, but have many small local man-

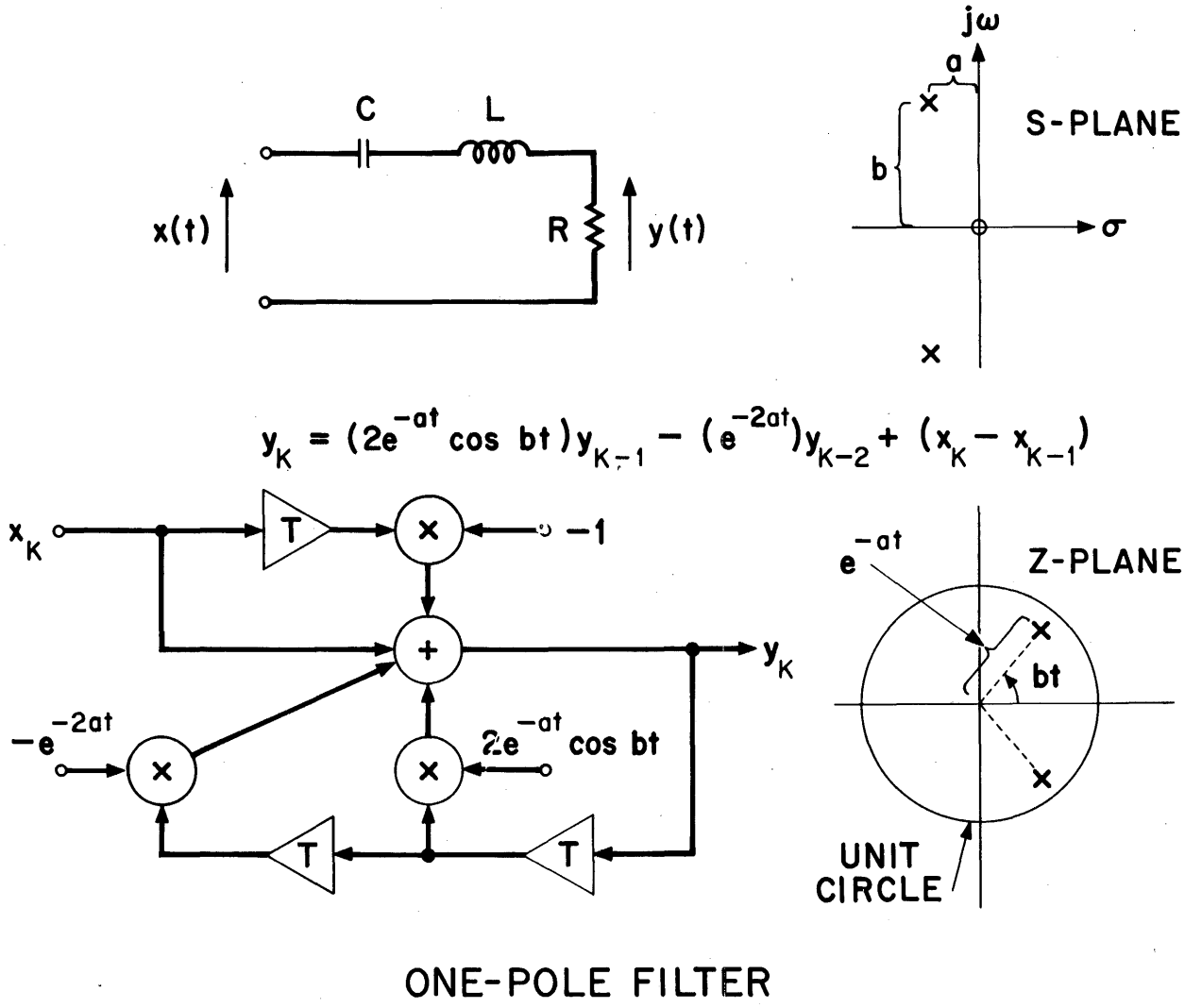


Figure 7.

made events in them; so in practice, the false alarm rate with a threshold of 5.82 is about four per hour. Using eight independent event detectors, the program takes 10 msec and occupies 1300 registers in core.

The main disadvantage of this event detector is that it cannot discriminate between a strong local event or a noise burst in the data line or a teleseism. Anything that has sufficient energy around 1¼ cps will trigger this event detector. We have found that for our data about one event in 10 is a genuine teleseism, the other nine being local events of one sort or another. We get around this by having more than one event detector connected to seismometers separated by several kilometers. Then, whenever several event detectors trigger within several seconds of each other, we say a teleseism has been received. The small local events are too weak to register on several event detectors, or else take a longer time due to their lower horizontal velocity. This screening effectively reduces the false alarm rate from 90% to about one per day.

When a teleseism has been detected, the program examines the relative arrival times at the various sensors and, since these determine the origin of the teleseism, the program looks to see if this location is in an area that it has been told is interesting. If it is, the program tells the operators to save the rele-

vant data which has been recorded on magnetic tape. Thus, the tape recorders have been acting as mass buffer storage for the data.

In order to save the data that has teleseisms that are too weak to be detected by the automatic event detector, one points a beam at the selected area and puts an event detector on its output, and if this single event detector triggers, the data is saved.

In conclusion, we have described a system with a real-time program directing the overall operation—from routine testing to selecting the data for further off-line processing. These operations are done on the basis of real-time waveform processing. Without automating these tasks, the Large Aperture Seismic Array would be quite impractical to operate because of its large size, both in data which accumulates and remote transducers which periodically fail.

HIGH-SPEED CONVOLUTION AND CORRELATION*

Thomas G. Stockham, Jr.
Massachusetts Institute of Technology, Project MAC
Cambridge, Massachusetts

INTRODUCTION

Cooley and Tukey[1] have disclosed a procedure for synthesizing and analyzing Fourier series for discrete periodic complex functions.† For functions of period N, where N is a power of 2, computation times are proportional to $N \log_2 N$ as expressed in Eq. (0).

$$T_{ct} = k_{ct} N \log_2 N \tag{0}$$

where k_{ct} is the constant of proportionality. For one realization for the IBM 7094, k_{ct} has been measured at 60 μsec. Normally the times required are proportional to N^2. For $N = 1000$ speed-up factors in the order of 50 have been realized! Eq. (1b) synthesizes the Fourier series in question. The complex Fourier coefficients are given by the analysis equation, Eq. (1a).

$$F(k) = \sum_{j=0}^{N-1} f(j) w^{-jk} \tag{1a}$$

$$f(j) = \frac{1}{N} \sum_{k=0}^{N-1} F(k) w^{jk} \tag{1b}$$

where $w = e^{2\pi i/N}$, the principal Nth root of unity. The functions f and F are said to form a discrete periodic complex transform pair. Both functions are of period N since

$$F(k) = F(k + cN) \tag{2a}$$

and

$$f(j) = f(j + cN) \tag{2b}$$

TRANSFORM PRODUCTS

Consider two functions g and h and their transforms G and H. Let G and H be multiplied to form the function C according to Eq. (3),

$$C(k) = G(k) \times H(k) \tag{3}$$

and consider the inverse transform $c(j)$. $c(j)$ is given by Eq. (4)

$$c(j) = \frac{1}{N} \sum_{J=0}^{N-1} g(J) h(j - J) = \frac{1}{N} \sum_{J=0}^{N-1} h(J) g(j - J) \tag{4}$$

as a sum of lagged products where the lags are performed circularly. Those values that are shifted from one end of the summation interval are circulated into the other.

The time required to compute $c(j)$ from either form of Eq. (4) is proportional to N^2. If one computes the transforms of g and h, performs the multiplication of Eq. (3), and then computes the inverse

*Work reported herein was supported (in part) by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01).

†To be able to use this procedure the period must be a highly composite number.

transform of C, one requires a time given by Eq. (5)

$$\begin{aligned} T_{\text{circ}} &= 3\,k_{ct}\,N\log_2 N + k_m N \\ &= k_{\text{circ}}\,N(\log_2 N + \mu) \end{aligned} \tag{5}$$

where $k_{\text{circ}} = 3k_{ct}$, $\mu = k_m/k_{\text{circ}}$, and $k_m N$ is the time required to compute Eq. (3). Of course this assumes N is a power of 2. Similar savings would be possible provided N is a highly composite number.

APERIODIC CONVOLUTION

The circular lagged product discussed above can be alternately regarded as a convolution of periodic functions of equal period. Through suitable modification a periodic convolution can be used to compute an aperiodic convolution when each aperiodic function has zero value everywhere outside some single finite aperture.

Let the functions be called $d(j)$ and $s(j)$. Let the larger finite aperture contain M discrete points and let the smaller contain N discrete points. The result of convolving these functions can be obtained from the result of circularly convolving suitable augmented functions. Let these augmented functions be periodic of period L, where L is the smallest power of 2 greater than or equal to $M + N$. Let them be called $da(j)$ and $sa(j)$ respectively, and be formed as indicated by Eq. (6).

$$\begin{aligned} fa(j) &= f(j + j_0) && 0 \le j \le M - 1 \\ &= 0 && M \le j \le L - 1 \\ &= fa(j + nL) && \text{otherwise} \end{aligned} \tag{6}$$

where j_0 symbolizes the first point in the aperture of the function in question. The intervals of zero values permit the two functions to be totally non-overlapped for at least one lagged product even though the lag is a circular one. Thus, while the result is itself a periodic function, each period is an exact replica of the desired aperiodic result.

The time required to compute this result is given in Eq. (7).

$$T_{\text{aper}} = k_{\text{circ}}\,L(\log_2 L + \mu) \tag{7}$$

where $M + N \le L < 2(M + N)$. For this case, while L must be adjusted to a power of 2 so that the high-speed Fourier transform can be applied, no restrictions are placed upon the values of either M or N.

SECTIONING

Let us assume that M is the aperture of $d(j)$ and N is that of $s(j)$. In situations where M is considerably larger than N, the procedure may be further streamlined by sectioning $d(j)$ into pieces each of which contains P discrete points where $P + N = L$, a power of 2. We require K sections where

$$K = \text{least integer} \ge M/P \tag{8}$$

Let the ith section of $d(j)$ be called $d_i(j)$. Each section is convolved aperiodically with $s(j)$ according to the discussion of the previous section, through the periodic convolution of the augmented sections, $da_i(j)$ and $sa(j)$.

Each result section, $r_i(j)$, has length $L = P + N$ and must be additively overlapped with its neighbors to form the composite result $r(j)$ which will be of length

$$KP + N \ge M + N \tag{9a}$$

If $r_i(j)$ is regarded as an aperiodic function with zero value for arguments outside the range $0 \le j \le L - 1$, these overlapped additions may be expressed as

$$r(j) = \sum_{i=0}^{K-1} r_i(j - iP) \quad j = 0,1,\ldots KP + N - 1 \tag{9b}$$

Each overlap margin has width N and there are $K - 1$ of them.

The time required for this aperiodic sectioned convolution is given in Eq. (10).

$$\begin{aligned} T_{\text{sect}} &= k_{ct}(P + N)\log_2(P + N) \\ &\quad + 2Kk_{ct}(P + N)\log_2(P + N) \\ &\quad + Kk_{\text{aux}}(P + N) \\ &= k_{ct}(2K + 1)(P + N)\log_2(P + N) \\ &\quad + Kk_{\text{aux}}(P + N) \\ &\approx k_{ct}(2K + 1)(P + N)[\log_2(P + N) + \mu'] \end{aligned} \tag{10}$$

where $\mu' = k_{\text{aux}}/2k_{ct}$. $Kk_{\text{aux}}(P + N)$ is the time required to complete auxiliary processes. These processes involve the multiplications of Eq. (3), the formation of the augmented sections $da_i(j)$, and the formation of $r(j)$ from the result sections $r_i(j)$. For the author's realization in which core memory was used for the secondary storage of input and output data, μ' was measured to be 1.5, which gives $k_{\text{aux}} = 3k_{ct} \approx 300\ \mu\text{sec}$. If slower forms of auxiliary storage were employed, this figure would be enlarged slightly.

For a specific pair of values M and N, P should be chosen to minimize T_{sect}. Since $P + N$ must be a

power of 2, it is a simple matter to evaluate Eq. (10) for a few values of P that are compatible with this constraint and select the optimum choice. The size of available memory will place an additional constraint on how large $P + N$ may be allowed to become. Memory allocation considerations degrade the benefits of these methods when N becomes too large. In extreme cases one is forced to split the kernel, $s(j)$, into packets, each of which is considered separately. The results corresponding to all packets are then added together after each has been shifted by a suitable number of packet widths. For the author's realization N must be limited to occupy about $\frac{1}{8}$ of the memory not used for the program or for the secondary storage of input/output data. For larger N, packets would be required.

COMBINATION OF SECTIONS IN PAIRS

If both functions to be convolved are real instead of complex, further time savings over Eq. (10) can be made by combining adjacent even and odd subscripted sections $da_i(j)$ into complex composites. Let even subscripted $da_i(j)$ be used as real parts and odd subscripted $da_{i+1}(j)$ be used as imaginary parts. Such a complex composite can then be transformed through the application of Eqs. (1a), (3), and (1b) to produce a complex composite result section. The desired even and odd subscripted result sections $r_i(j)$ and $r_{i+1}(j)$ are respectively the real and imaginary parts of that complex result section.

This device reduces the time required to perform the convolution by approximately a factor of 2. More precisely it modifies K by changing Eq. (8) to

$$K = \text{least integer} \geq M/2P \tag{11}$$

For very large numbers of sections, K, Eq. (10) can be simplified to a form involving M explicitly instead of implicitly through K. That form is given in Eq. (12)

$$T_{\text{fast}} \approx k_{ct} M((P + N)/P)[\log_2 (P + N) + \mu'] \tag{12}$$

Since it makes no sense to choose $P < N$, for simple estimates of an approximate computation time we can write

$$T_{\text{fast}} \approx 2k_{ct} M[\log_2 N + \mu' + 1] \tag{13}$$

EMPIRICAL TIMES

The process for combined-sectioned-aperiodic convolution of real functions described above was implemented in the MAD language on the IBM 7094 Computer. Comparisons were made with a MAD language realization of a standard sum of lagged products for N = 16, 24, 32, 48, 64, 96, 128, 192, and 256. In each case M was selected to cause Eq. (11) to be fulfilled with the equal sign. This step favors the fast method by avoiding edge effects. However, P was not selected according to the optimization method described above (under "Sectioning Convolution"), but rather by selecting L as large as possible under the constraint.

$$\ln L \geq P/N \tag{14}$$

This choice can favor the standard method.

Table 1 compares for various N the actual computation times required in seconds as well as times in milliseconds per unit lag. Values of M, K, and L are also given.

Relative speed factors are shown in Table 2.

ACCURACY

The accuracy of the computational procedure described above is expected to be as good or better

Table 1. Comparative Convolution Times for Various N

N	16	24	32	48	64	96	128	192	256
M	192	208	384	416	768	832	1536	1664	3584
K	2	1	2	1	2	1	2	1	1
L	64	128	128	256	256	512	512	1024	2048
	Time in seconds								
T_{standard}	0.2	0.31	0.8	1.25	3.0	5.0	12	20	48
T_{fast}	0.3	0.4	0.6	0.8	1.3	1.8	3.0	3.8	8.0
	Time in milliseconds per unit lag								
$T_{\text{standard}/M}$	1.0	1.4	2.0	3.0	3.9	6.0	7.8	12.0	13.3
$T_{\text{fast}/M}$	1.5	1.9	1.5	1.9	1.6	2.1	1.9	2.2	2.2

Table 2. Speed Factors for Various N

N	16	24	32	48	64	96	128	192	256	512	1024	2048	4096
Speed factor	$\frac{2}{3}$	$\frac{3}{4}$	$\frac{4}{3}$	1.5	2.3	2.8	4.0	5.2	6	13*	24*	44*	80*

*Estimated values.

than that obtainable by summing products. Specific investigations of the accuracy of the program used to accumulate the data of Tables 1 and 2 are in process at the time of this writing. The above expectations are fostered by accuracy measurements made for floating-point data on the Cooley-Tukey procedure and a standard Fourier procedure. Since the standard Fourier procedure computes summed products, its accuracy characteristics are similar to those of a standard convolution which also computes summed products. Cases involving functions of period 64 and 256 were measured and it was discovered that two Cooley-Tukey transforms in cascade produced respectively as much, and half as much, error as a single standard Fourier transform. This data implies that the procedures disclosed here may yield more accurate results than standard methods with increasing relative accuracy for larger N.

APPLICATIONS

Today the major applications for the computation of lagged products are digital signal processing and spectral analysis.

Digital signal processing, or digital filtering as it is sometimes called, is often accomplished through the use of suitable difference equation techniques. For difference equations characterized by only a few parameters, computations may be performed in times short compared to those required for a standard lagged product or the method described here. However, in some cases, the desired filter characteristics are too complex to permit realization by a sufficiently simple difference equation. The most notable cases are those requiring high frequency selectivity coupled with short-duration impulse response and those in which the impulse response is found through physical measurements. In these situations it is desirable to employ the techniques described here either alone or cascaded with difference equation filters.

The standard methods for performing spectral analysis[2] involves the computation of lagged products of the form

$$F(j) = \sum_{J=0}^{N-j-1} x(J)\,y(J+j) \qquad (15)$$

which, in turn, after weighting by so-called spectral windows are Fourier transformed into power spectrum estimates. Speed advantages can be gained when Eq. (15) is evaluated in a manner similar to that outlined above (under "Aperiodic Convolution") except that in this case L is only required to exceed $N + \Omega$ where Ω is the number of lags to be considered. This relaxed requirement on L is possible because it is not necessary to avoid the effect of performing the lags circularly for all L lags but rather for only Ω of them. An additional constraint is that Ω be larger than a multiple of $\log_2 L$. The usual practice is to evaluate Eq. (15) for a number of lags equal to a substantial fraction of N. Since the typical situation involves values of N in the hundreds and thousands, the associated savings may be appreciable for this application.

Digital spatial filtering is becoming an increasingly important subject.[3,4] The principles discussed here are easily extended to the computation of lagged products across two or more dimensions. Time savings depend on the total number of data points contained within the entire data space in question, and they depend on this number in a manner similar to that characterizing the one-dimension case.

ACKNOWLEDGMENTS

The author is indebted to Charles M. Rader of the MIT Lincoln Laboratory for his ideas concerning the Cooley-Tukey algorithm and to Alan V. Oppenheim of the Electrical Engineering Department, MIT, for suggesting that high-speed convolutions might be realized through the utilization of that algorithm. During the preparation of this work the author became aware of the related independent efforts of Howard D. Helms, Bell Telephone Laboratories, and Gordon Sande, Jr., Princeton University.

REFERENCES

1. J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, (Apr. 1965).

2. R. B. Blackman and J. W. Tukey, *The Measurement of Power Spectra*, Dover Publications, New York, 1959; also *Bell System Technical Journal*, Jan. and Mar. 1958.

3. T. S. Huang and O. J. Tretiak, "Research in Picture Processing," *Optical and Electro-Optical Information Processing*, J. Tippett et al, eds., MIT Press, Cambridge, Mass., 1965, Chap. 3.

4. T. S. Huang, "PCM Picture Transmission," *IEEE Spectrum*, vol. 2, no. 12, pp. 57–63 (Dec. 1965).

A COMPUTER PROGRAM TO TRANSLATE MACHINE LANGUAGE INTO FORTRAN

William A. Sassaman
TRW Systems, Inc.
Redondo Beach, California

This paper describes a computer program which translates machine language into FORTRAN. The program was developed at TRW, Inc., to aid in the conversion process from our existing equipment to a third generation computer. The translator was written to be a real help to people personally involved in conversion, and is intended to be an operational program rather than a pure research project.

As the title of this paper indicates, the output language is FORTRAN. Since the translator design is not very dependent upon output language, this appears to be arbitrary. FORTRAN was chosen since it is a standard. It is well defined and runs on both our second and third generation computers. It has deficiencies but they are known. It was desired to go to a problem-oriented language to increase future human productivity and therefore machine language output was not encouraged. In addition, compiler writers expand considerable effort to obtain efficient machine language codes and duplicating this effort appears a waste. Anyone who really wants machine language can get it as a by-product of the FORTRAN compilation and hand-massage it to any degree of perfection he desires. If the problem is so complex that no FORTRAN translation is possible, then a completely human effort appears in order.

The input is symbolic assembly language for the IBM 7000 series computer operating under a privately developed execution supervisor. The symbolic media was chosen over the binary machine language since the symbolic cards make it easier to distinguish between the various types of data, allow macro identification and contain otherwise useful information. Since continuity of usage is expected between the original and translated versions of the program, it appears highly desirable to maintain much of the symbolic notation.

In my original thinking of the translational process, I was impressed with the concept that actual translation was, in general, a clerical process rather than an inventive one. That is, the programmer following the assembly listing figures out from it (and any documentation) what the original programmer was doing and codes this in the appropriate language for the new machine. Although often it is necessary to have a knowledge of the problem being solved, much of the time the translating programmer operates as a clerical symbol manipulator. It is true that the rules for the symbol manipulation are complex, but the task is basically clerical and therefore subject to automation. In the translator I have tried to assign these simple clerical tasks to the machine and allow the human more time to perform in the areas where he can contribute the most.

From a technical viewpoint, it is probably impossible to write a program which will translate all of one computer program into a similarly efficient program for a second computer. However, as with

many mathematical processes, it is feasible to approach the solution as a limit, such that a maximum automated transfer of source programs may be effected with minimal cost and human intervention. The task of writing a translation program has as its major obstacles the definition of the rules for translation. It would seem unlikely that anyone could a priori define all the rules. Therefore a learning approach has been defined to allow the development of the model as experience is gained in translation.

I did not try to design for a 100% translation.

Input/output, functions, subroutines, standardized routines, etc., need not be translated. Further the conversion effort is not by nature one that may be completely automated. During the process of converting a program, decisions are made as to the plan of attack during conversion, i.e., the human programmer, who has cognizance of the physical problem being solved and the capabilities and shortcomings of the program, decides which areas are to be rewritten, which areas are to be deleted, which areas will be replaced by system subroutines or standardized routines, and finally the remainder is to be translated (or transliterated if you wish). It is rather apparent that these decisions are probably not the sort to be made by the computer. Further, the rewriting or regrouping of computations must also be performed by humans. The remaining area, translation, is a potential area for automation. Of course, I try to do a good job in that which is translated; however, the law of diminishing returns dictates that the translational rules limit one to about 90% of the code.

Further, I concluded that the source machine language program contains a great deal of information and the translator, retrieving and organizing this information could perform a very valuable service in documentation as well as aiding the conversion effort.

The translator during translation attempts to operate as the human does. The programmer in translation recognizes in coding not only the individual instructions but also, and more specifically, problem-oriented functions, which may be one or more machine language instructions. It is the purpose of the translator then to recognize these functions with their terminals as well as to gather and organize the program information pertaining to the translational rules. The functions are gathered into statements as appropriate before output.

Many of these functions are easily recognizable in the machine language code. A simple example of machine language instructions which are easily recognizable as functions are arithmetic codes. These may easily be built up into larger statements. One of these arithmetic statements consists of a string of functions appropriately connected. The translator inserts a right parenthesis prior to each multiplication or division, and a left parenthesis following each square root or other function. A similar left or right parenthesis must be entered at the start of the statement. The statement is normally terminated by a store instruction. The address of this store instruction is obviously the left side of the FORTRAN statement and followed by an equal sign. The method of translating addresses illustrates the buildup of rules for translation. Consider the coding:

```
1.)        CLA   A
2.)        AXT   10, 4
3.)  D     FAD   A + 1
4.)        FDP   B, 4
5.)        STQ   C + 10, 4
6.)        TIX   D, 4, 1
```

The first A is obviously translated as A. Since A = A (1), A + 1 must be translated as A (2). Instructions three through six are translated as a DO loop using the dummy variable ND X 4. Since it is subject to index register modification and has no additive address, the B is assumed to be a vector running backwards in storage and is translated as a forward running FORTRAN array B (ND X 4). The C is assumed to be forward running and is translated as C (ND X 4 + 10 − 10), where the tens cancel out leaving C (10). Obviously the D is translated as a FORTRAN numeric statement number. Functions may be more complex and require more complex rules for translation. A good example of these are the programmer tricks of using instructions for something the manufacturer never intended; for the IBM 7000 series, a PXD 0, 0 will clear the accumulator; a LRS of zero will impose the sign of the accumulator into the MQ, etc. Such translation is analogous to the handling of idioms and slang in human language outside of a word for word grammatical translation.

The last bit of philosophy in the design of the translator is the target. The programs to be converted are engineering applications involving algebraic algorithms. These algorithms are easily defined and form the basis of the translator rule set.

As a result of these thoughts, the translator was designed to intimately interface with and operate under the supervision of the human user. The human describes the rules for the particular pro-

gram involved via control cards, defines areas to be translated and criteria for recognition of areas of coding to be translated as FORTRAN subroutines. Operating with these rules and the basic set, the computer then performs any initial translation. This initial attempt normally tells the user what tasks cannot be handled in FORTRAN, indicates the need for additional rules such that the translator will give a better translation. The deck is then resubmitted to the computer. The human examines the computer output and either edits it to achieve the desired code or redefines the rules or control cards and translates over again. This learning process and human interface dictates the need for a system to afford maximum convenience and ease of communication to the user. Although this would appear to be an ideal on-line application, the schedule, hardware and manpower available dictated the utilization of a typical centralized large-scale computer.

OPERATION

The translator's functions are to retrieve information from the source deck, organize this information, merge it with other data, apply the rules for translation and provide interfaces with the human during the process. This is not done on-line, although the nature of the problem indicates an on-line solution might enhance the process. In order to accomplish these functions, the translator is designed in six separate (and recoverable) phases. The main task of each of these phases is:

Phase I	Separate the program into logical groups.
Phase II	Handle parameters — data types, dimensions, COMMON, initial values.
Phase III	Core map of symbol allocation and overlay.
Phase IV	Translation of macros.
Phase V	Translation into FORTRAN, routine by routine.
Phase VI	Editing and merging.

Although these are the main functions, the phases have additional tasks because of convenience of execution. In order to explain the process, I will go through it phase by phase, explaining what is done and where the information comes from.

Phase I is the initial phase whose primary task is to divide the program into logical groups of manageable size. The input to the translator is the source symbolic card deck, or a tape containing the card images. The input is read, basically a card at a time, and broken into the following categories.

- Areas to be treated as FORTRAN subroutines (tape)
- Data and parameters (tape)
- Symbolic equivalence (core/cards)
- Macro skeletons (tape)

In order to make these distinctions, the translator must know the algorithms for separation. The BEGIN pseudo operation is recognized as the start of a routine and the terminus of a previous routine. Origin and transfer cards are assumed to signal the end of a routine. Decimal, octal and Hollerith data are presumed to be in the data domain unless they appear to be in a routine and do not have a symbolic location assigned to them. The programmer is allowed to enter control cards in the data stream to allow the programmer to label name COMMON in the output stream. Each card included in a subroutine area is assigned a FORTRAN location number.

In order to build up an initial table of floating point (real) symbols, the address of each floating point instruction is saved as well as the address of a loading instruction immediately before it or a storing instruction immediately following it. Each symbolic name is saved, in sequence, with all origin, intermediate transfer cards and the final end card. The address of the first intermediate transfer card (if none, the address of the end card) is saved as the point at which computation will be initiated.

Phase II is designed to handle data and parameters. The list of floating point symbols generated in Phase I is organized and checked for redundancy. The translator then reads the data and parameter tape and compares the parameter symbolic name to the built up symbol table equivalences. The symbol is then checked to see whether it is an allowable FORTRAN symbol (alphanumeric, initiated, by a letter), what the type is, and what the dimension is. The translator tries to define a new symbolic reference for illegal symbols, expands the symbol table of equivalences and builds up a table of real and integer symbols not conforming to the FORTRAN rules. Every attempt is made to keep the original symbol as it is assumed to be mnemonic. From the contents of the data card image, a data statement is generated if initial values are assigned to the parameter. The name and dimension are included in the name COMMON block as assigned by the

translator or the programmer if he has such input information. Upon termination of a name COMMON block, the card images are saved on an intermediate tape with the images of the type statements.

Upon processing all data cards, the translator calls the computation, using the address of the end card (or first intermediate transfer card). A listing is then made of all parameters with their original symbolic name, the corrected name, if any, and the comments from the data card.

Phase III (storage allocation core map and overlay structure). The program reads the tape containing the above information, compares it to the equivalence table and breaks it ino *n* strings. These strings are then printed in *n* columns with origins matched on the vertical scale.

Phase IV (macro translation). Macros are currently translated as functions, if they can be, or are ignored. Details of translation are similar to the translation of subroutines.

Phase V (translation into FORTRAN subroutines). This is what most people consider the heart of translation. The card images of the area to be converted are read into core from the tape storage. Each card is assigned a sequential external formula number. Those which are not used will be suppressed at output time. An initial pass is made to find the address of all transfer instructions and to save the concomitant FORTRAN numeric location. Locations which are transfers, or transferred to, are appropriately flagged. New numeric locations are assigned for undefined transfer addresses. Special flags are set for the addresses of TIX instructions, TXI and TXH (under certain circumstances index loading instructions, address modification, etc.).

The actual translation is now begun. The translator is broken into two alternate paths here: the first being a search for instructions or functions that initialize a statement (a statement being merely a string of appropriately connected functions); the second being a search for functions that sustain or terminate a statement. In general, the translator scans the coding until it recognizes the start of a statement; then it switches to the terminal branch where it builds up the function into a statement until some terminating condition is reached. If on the initial scan, a sustaining type instruction were encountered, the translator initiates an appropriate function to start things off and transfers to the terminal branch. Similarly if in the terminal branch the translator finds an initiating statement, it supplies a terminal function to complete the statement being processed, tries to search out particular programmer tricks since something "different" is happening, and then transfers to the initial branch. Loading, storing and transfer instructions, and instructions which are transferred to, are samples of what are considered to start or end a statement. CALL type instruction (TSX) are considered to start or end a previous statement and start a new one unless the translator can determine that they are replaceable by a built-in arithmetic function or other functions, in which case the function is included in the statement being processed and the translator continues on the terminal branch. Arbitrarily terminated or initiated statements are stored in or picked up from "dummy" accumulator, MQ, registers, etc.

The translator on either branch attempts to search out TIX loops, where a register is counted down from *n* to 1 or TXI loops where a register is incremented from 0 to *n* and translates as DO loops nested to a level of 7; if all 7 index registers are used and the index registers are not saved internally. A DO statement is inserted just prior to initiation of the loop. The loop is terminated by a dummy CONTINUE. The pseudo symbol NDXA is used to represent index register A. The algorithm for conversion of parameter addresses while in a TIX loop deducts the initial value of the index register from any associated address. Note there is a difference in the assignment of vectors in machine language programs and FORTRAN, each considering the other backward. The algorithm attempts to cover the difference.

As previously mentioned, the easiest instructions to translate are the arithmetic instructions where each operation and address is added to the right-hand end of the statement being generated. For multiplication and division a pair of parentheses must be added, one at each end before the operator and operand are saved. For functions, a similar pair of parentheses must be added except of course, the functions appear on the left, before the initial parenthesis. A storing type instruction adds an equal sign on the left and the address of the operand to the left of that. Translated output statements are built up in a table, word by word, until a terminating condition is reached. The statement is compacted by reducing spurious blanks, continuation numbers are assigned if more than one card image is required; and the images are written on a blocked output tape with an alter number for each card image. If it has been referenced, the FORTRAN numeric location is also written out on the

first card image. At this point an almost side by side listing of the original coding and the translated code are printed for the user.

By actually scanning, instruction by instruction, while the whole subroutine is in core, the translator can look for particular sequences of coding which represent a special case. Much of the detail coding to effect these rules is lengthy and tends to be repetitive so that many subroutines are used in the areas of duplication. The coding of the translator is all very straightforward, and often tedious.

Phase VI allows the user to edit the translator output from the previous phases which consisted of an almost side by side listing of the original and translated coding. Also used is the tape which contains all of the previous translation including the alter numbers. It is the purpose of Phase VI to allow the editing of this tape by use of the alter numbers and to produce a new tape and/or a new listing and/or a punched card deck. The editing is performed by a series of control cards which allow the user to add or delete cards from the tape or to juggle large blocks from one place on the tape to another, without actually shuffling through the cards.

To maximize usefulness, the output tape may be fed into the FORTRAN compiler at this time without the submittal of a separate run or punching the cards.

USER INPUTS

The recipe for elephant stew traditionally starts with "clean one freshly killed elephant." Similarly the user's input starts with the program to be translated. This may be in the form of symbolic source cards or blocked card images on a tape. In general, the areas for which translation is not desired are deleted from the deck. This deck is preceded by a control card that tells the translator whether this is a SMASHT, SCAT or IBMAP deck and whether cards or tape is expected. A number of EQU cards may be input by the user to assign names to illegal symbols, rather than accept the translator's naming. REAL and FIXED define the type of data these operations refer to when the translator has insufficient information to arrive at the appropriate conclusion. Control cards define the subroutines of function calling sequences for translating the TSX address. The rules for separation of areas into subroutines may be defined either by special coding in the translator or insertion of dummy control cards. Special algorithms for translational rules are coded into the translator at this time. The run is now ready for submittal.

Upon return of the run, the data previously furnished may be modified and the appropriate processes repeated or the user may desire to continue into the editing phase and obtain an output deck.

SAMPLE TRANSLATION

Translation is such a complex function that no all-encompassing sample is feasible in such a short period of time.

CONCLUSION AND SUMMARY

The translator described here is not a perfect tool —it does not translate everything nor is everything it translates perfect. It does not handle dynamic programming, i.e., where coding is actually charged, nor does it handle indirect addressing. Patently it does not translate into FORTRAN those things FORTRAN cannot do. Complex and double precision arithmetic are not attempted. It is designed to relieve the programmer of much of the clerical task of translation and to allow the user input into the translational process and absolute control of the final output. For those applications we have used it for, it has performed rapidly and effectively.

TECHNIQUES AND ADVANTAGES OF USING THE FORMAL COMPILER WRITING SYSTEM FSL TO IMPLEMENT A FORMULA ALGOL COMPILER*

Renato Iturriaga, Thomas A. Standish, Rudolph A. Krutar
and Jackson C. Earley
Carnegie Institute of Technology
Pittsburgh, Pennsylvania

INTRODUCTION

Implementing a compiler, as everybody knows, is not an easy task. There have appeared in the past few years a number of compiler writing systems.[1-3] One of these is Feldman's "Formal Semantic Language" (FSL).[4,5] In Feldman's thesis the assertion is made that FSL is potentially a powerful compiler writing system. The Formula Algol compiler[6] is a large, nontrivial compiler incorporating several new language features, and the use of FSL to implement it constitutes the first significant test of the power of FSL. We find Feldman's assertion is justified, and the ideas he set forth in theory have been found to be successful in practice.

Some of the more important advantages of FSL that we have found are as follows.

First, the amount of time and programming effort required to implement a compiler such as the Formula Algol compiler is reasonably small (on the order of a man-year).

Second, because FSL is a high-level language incorporating certain power of expression, the task of describing compiling processes is sufficiently manageable and easy that experimental flexibility is achieved. By this we mean that we were able to experiment with a variety of organizations of parts of the compiler in order to select those with desired properties. In particular, we were able to experiment with the syntax of Formula Algol without appreciably changing its semantics. We could also change its semantics without appreciably changing the syntax. Thus we were able to use FSL as a tool to improve the source language and at the same time to improve its implementation by finding the best compilation techniques. In contrast to the case of a hand coded compiler, we were not forced to make any organizational commitments which were of prohibitive expense to change. This is the essential reason underlying the property of flexibility, and flexibility makes FSL a good tool for a programming language designer.

A third feature of using FSL to write a compiler is that it is a rare counterexample to the familiar tradeoff between efficiency and generality. The use of the Floyd-Evans production language[7,8] permits one to write an efficient syntax analyzer with sophisticated error recovery, and a compiler written in FSL not only is reasonably efficient but if written

*The research reported here was supported by the Advanced Research Projects Agency of the Department of Defense under Contract SD-146 to the Carnegie Institute of Technology. R. Iturriaga is partially supported by the National University of Mexico and the Instituto Nacional de la Investigacion Cientifica. T. A. Standish is a National Science Foundation graduate fellow.

properly can produce efficient object code. For example, the Formula Algol compiler produces for some classes of expressions more efficient code than the current handwritten Algol compiler in use at Carnegie Tech.

A fourth feature of FSL is that it is a language sufficiently general to allow several of the better-known useful compiling techniques to be expressed and utilized. For example, THUNKS[9] were used to implement parameter calls in procedures, dope vectors[10] were used to implement array storage and accessing, and the use of symbol table techniques was made easy by the fact that tables are primitive in the language. The reason for this generality is that FSL contains a powerful set of primitives that permit a user to express a large variety of compiling mechanisms directly by combination of these primitives. This feature also permitted us to invent several new variations on known compiling techniques which were well adapted to the problem at hand.

A fifth property for which, at present, we can produce no real evidence attesting to its usefulness, is that a compiler written in FSL is given a formal description. This means that in contrast to handwritten compilers we are provided with a framework in which we can begin to approach the problems of proving that compilers recognize given source languages correctly or that they compile correct code. In the case of handwritten compilers these questions are unthinkable. As an example of the kind of approach that can be made once a formal description is given we cite a doctoral thesis by Evans,[11] in which certain properties of the production language are proven.

Finally, the activity of implementing Formula Algol had a feedback effect on the design and implementation of FSL itself.[12] Modifications were made easy by the fact that FSL is in effect compiled in itself and thus possesses the same organization as the compilers it produces. For example, an accumulator symbol was introduced as a variable to allow the user to deal formally with the use of the accumulator. This represents a small change in the original philosophy of FSL, which was designed with machine independence in mind. It is, however, a small change with far-reaching consequences.

With the exception of the property of the usefulness of formal descriptions of compilers, we will present later in this paper concrete evidence supporting each of the claims we have made in this introduction. Our first task, however, is to explain briefly the operation of the compiler writing system.

A BRIEF EXPLANATION OF THE COMPILER WRITING SYSTEM

The compiler writing system uses two formal languages to describe a compiler. First, a syntax analyzer for the source language is written as a program in the *production language.* This program is processed by a translator called the *production loader* producing as output a set of driving tables which are stored for later use. Second, a collection of semantic routines is defined by writing a program in the *formal semantic language.* Another translator called the *semantic loader* then translates this collection of routines into a set of tables and a block of code, which code is compiled for use as a part of the compiler itself. This output is also intermediate and is stored for later use.

The compiler itself (Fig. 1) is another program which reads in both the syntax tables and the semantic tables and code, and by using these translates a source language program into an object program. For the sake of efficiency a preliminary lexical transformation is performed on source language text as it is read in by a routine called the *subscan.* This routine recognizes the primary units of the language which are operators, reserved words, identifiers, and constants. These primary units of the source language are not fixed by the system but are declared in the production language. The subscan is a closed routine called by statements in the production language. Each time it is called it returns with the next primary unit in the source language string. As each identifier is recognized by the subscan its print name is stored in a table unless it has been entered previously, and an integer representing its relative address in the table of print names is transmitted. This integer functions from that time on as the internal name of the identifier. Abbreviations of reserved words and of operators are transmitted directly, and constants which are too long to transmit directly are saved in a table and their locations are transmitted instead.

The fundamental mechanism in the compiler is a push-down stack of ordered pairs (α, β) where α is a primitive syntax unit and where β holds semantic information and is called the "description of α." Syntactic analysis of the source language proceeds by a sequence of manipulations of this stack. The production language is used to define these manipulations and it consists of a sequence of productions of the following form:

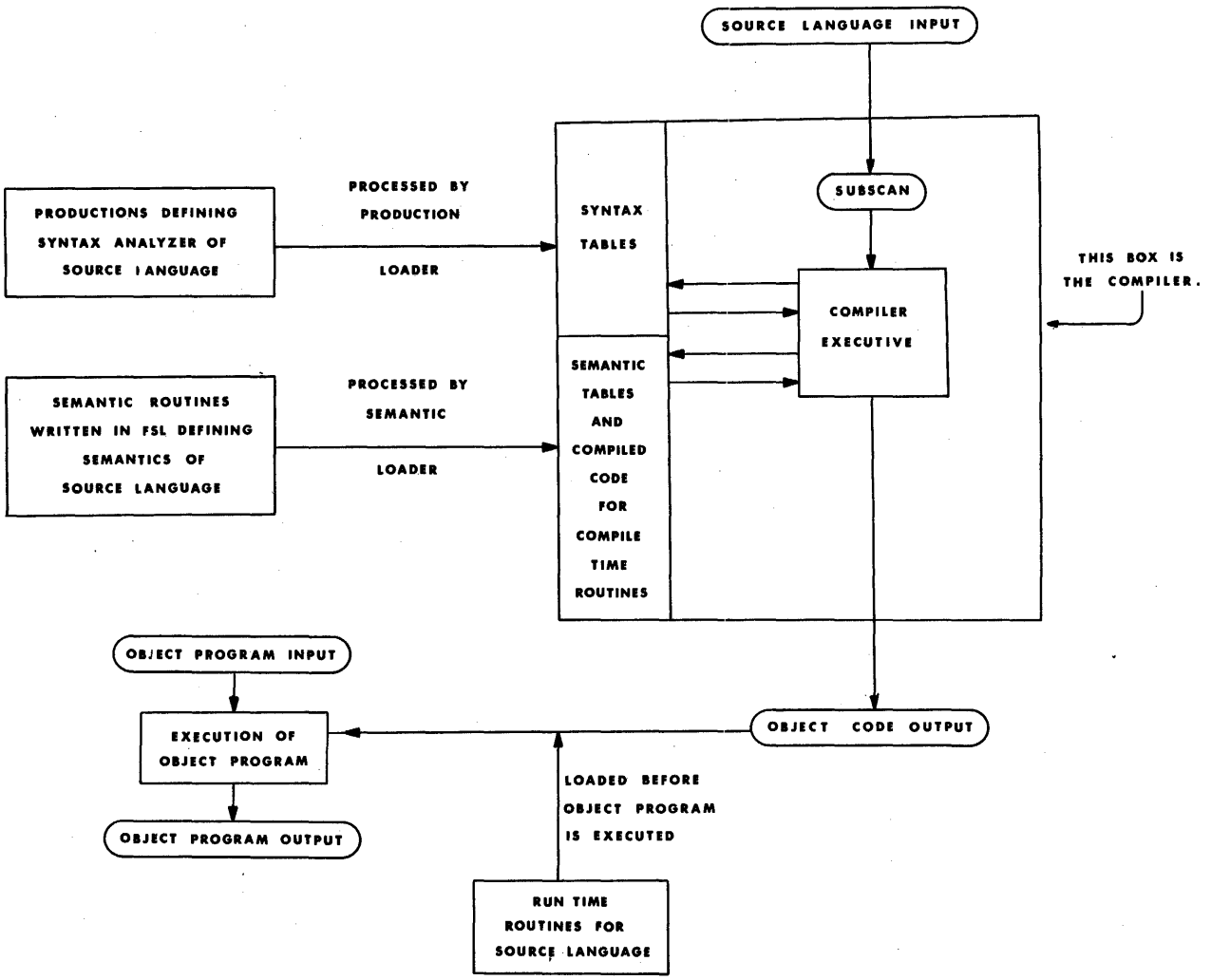


Figure 1. Flow of systems.

LABEL L5 L4 L3 L2 L1 | → R3 R2 R1
| ACTION *LINK

where the appearance of everything except L1 and the two vertical bars is optional. Each production tests for the presence of a particular configuration among the topmost syntactic units in the stack by attempting to match the pattern given by L5 L4 L3 L2 L1 against them. If a match is found and if → R3 R2 R1 occurs in the production then the configuration matching L5 L4 L3 L2 L1 is transformed into the configuration represented by R3 R2 R1. Then the ACTION is executed. If → R3 R2 R1 is absent and a match is found then the ACTION is executed and no transformation is performed. The only three actions we will discuss in this paper are the actions EXEC, SUBR, and RETURN. The actions SUBR and RETURN will be explained later. The action EXEC N, with parameter N, is a call to the semantic routine numbered N. This semantic routine may, among other things, alter the descriptions of the syntactic units involved in the stack transformation described by the production. For example, the description of an identifier may consist of the address of the run time location assigned to the variable that identifier represents. The description may also bear type bits telling the type of the variable (e.g., real, integer, formula). After the action is executed the link is examined to see if it is prefixed by a *. If it is then the subscan is called and the next primary syntactic unit in the source language string is recognized and

placed on top of the stack. Then control passes to the production whose label is given by the link. If a * does not precede the link then subscan is not called and control changes as in the previous case. In the event that the pattern L5 L4 L3 L2 L1 did not match the configuration of syntactic units at the top of the stack control passes to the next production in sequence. Also if the link is blank the action on the next line is executed and this process repeats until a nonblank link is found.

A sequence of productions may be organized into a closed subroutine by use of the actions SUBR and RETURN. The first production in such a sequence must be labeled and its label, say L, is the name of the closed subroutine. To call the closed subroutine starting at label L we execute the action SUBR L. When we wish to return from the subroutine we execute the action RETURN and control returns to the link of the production that called the subroutine originally. The control mechanism contains a push-down stack permitting recursive calls on the closed subroutines.

The general structure of the syntax analyzer for Formula Algol is as follows. The major units of the source language, such as statements and expressions, correspond directly to subroutines in the production language, which subroutines analyze the given major units. For example, there is a production subroutine called the "statement scanner," which is called every time a statement is expected in the source language. There is also a production subroutine called the "expression scanner," which processes expressions of all types and which is called every time an expression is expected in the source language. The statement scanner may call the expression scanner and, in fact, corresponding to the occurrence in the source language of statements which contain other statements as parts, the statement scanner may call itself. The flow of control through the syntax analyzer is governed by the structure of the source language program being analyzed. It is roughly true that the structure of such large production subroutines as the statement scanner and the expression scanner is the following. Upon entrance to the subroutine the first few characters of an expected source language construction are subjected to a sequence of tests which separate the various possible classes of constructions that may be encountered into cases. Corresponding to each case a transfer is made to a part of the routine which treats that case specially. This basic scheme of organization was first introduced by Evans in the writing of the Carnegie Tech Algol compiler[8] and the structure of the syntax analyzer for the Formula Algol compiler is basically an extension of it.

Because the flow of control in the syntax analyzer is directed by the constructions encountered in the source language it will be possible to use the following technique to explain various mechanisms found in the compiler. We will focus our attention on a critical subset of productions responsible for processing a given type of construction. This critical subsystem will always be embedded in a larger context but since the flow of control will never involve that context we may isolate the critical subsystem for study. This subsystem will involve calls on a set of semantic routines, and these semantic routines will be solely responsible for the compilation corresponding to the constructions which the critical subsystem processes.

An explanation of the primitives in the formal semantic language is given in Refs. 4 and 5. A complete summary of those primitives is not given in this paper, but the subset of primitives used below are accompanied by explanations in order to make the treatment self-contained.

A DETAILED EXAMPLE

Let us consider an example of the compilation of the assignment statement X ← A + B × C;. As we begin to process this statement control in the productions will be transferred to the statement scanner at label S1 where at entrance to the scanner the first character X has been recognized by the subscan and has been stacked as a syntactic unit I on top of the push-down stack. Furthermore, subscan sets the description of I to be the integer which is the relative address of its print name. The critical productions in the statement scanner which treat this case are as follows:

```
S1          I  |                 |              *S2
      . . . .
S2    I    ←   |  →      E  ←←   |  EXEC 9      *E1
      I    :   |  →              |  EXEC 91     *S1
      . . . .
```

The first production at S1 matches all statements which start with an identifier, and control is transferred to S2 after scanning the next character. At S2 a discrimination is performed on the second character and in the case of assignment statements the initial identifier I is changed to an E and control is transferred to EXEC 9 where a look-up in the symbol table is performed using the integer in the

description of I. This causes the retrieval of the location of the variable X which was assigned previously and stored in the symbol table while processing the declaration of X. It is conceivable that X was not declared and thus not stored in the symbol table. The table look-up procedure sets a signal in the event that it fails to locate an object during a table look-up and a test on this signal enables us to write a semantic error exit corresponding to the case where a variable is used but not declared in a source language program. Upon finding the entry corresponding to X in the symbol table the run time location of X, and its type (real, formula, etc.) are retrieved and the description of E in the stack is set to contain the run-time address, the type bits, and a bit to denote that the location rather than the value of X is desired. This description will be carried along as the associate of E until code is compiled to perform the assignment. Thus the FSL code for EXEC 9 looks as follows:

```
9 ↓   MARKJUMP [FIND];
        SIGNAL →
        RIGHT2 ← KEY + MODE0
          + TYPE + RELOC :
        FAULT 9 $ ↓
---------------------------------------------------
"FIND"
        KEY ← SYMBOL [ LEFT2 , $ , , ] ;
        TYPE ← SYMBOL [0 , , $ , ] ;
        RELOC ← SYMBOL [0 , , , $] ;
        JUMP [ < FIND > ]
```

Upon entrance to EXEC 9 we execute a mark transfer to a closed subroutine called FIND which performs the symbol table look-up using the integer given in the description in the LEFT 2 entry (same as L2 position defined above) in the push-down stack. LEFT2 is a variable whose value is this description. It is used in the statement KEY ← SYMBOL [LEFT2, $, ,] to locate the entry in the symbol table named SYMBOL which begins with the integer given by the value of LEFT2. Each line in the symbol table is of the form [integer, location, type, relocation base]. The relative position of the dollar sign $ among the commas indicates which of the entries in the located line we wish to extract. Hence the statement KEY ← SYMBOL [LEFT2, $, ,] extracts the location assigned to the variable whose internal integer is the value of LEFT2, and assigns this location to be the value of the variable KEY. If a zero occurs in place of LEFT2, as in the statement TYPE ← SYMBOL [0 , , $,] , the extraction defined uses the line previously selected saving the cost of an additional identical look-up. Thus for our example the routine FIND simply sets the KEY to the location of X, TYPE to the type of X, and RELOC to the relocation base of X (the relocation base is used to implement recursion and is too complicated to explain here). The statement JUMP [< FIND >] is a return to a mark transfer call. Returning now to the consideration of EXEC 9 we assume we have executed a mark transfer to FIND and have returned with either the signal set false to denote that the table look-up was a failure, or the signal set true and the variables KEY, TYPE, and RELOC set with the extracted values. The statement SIGNAL → RIGHT2 ← KEY + MODE0 + TYPE + RELOC : FAULT 9 $ is equivalent to the Algol statement <u>if</u> SIGNAL = TRUE <u>then</u> RIGHT2 ← KEY ∨ MODE0 ∨ TYPE ∨ RELOC <u>else</u> PRINT ('SEMANTIC FAULT 9') where MODE0 is a variable containing a bit denoting that a location rather than a value will be used. Executing this statement causes the logical union of the values of the variables to be stored as the description of the element in the R2 position of the stack in the event that the signal was true, and it causes a semantic fault to be printed otherwise.

At this point in the consideration of our example, control is returned from EXEC 9 back to the production that called it. This in turn causes another character to be scanned and control to be transferred to E1 which is the beginning of the expression scanner. The expression scanner contains two main parts, one starting at E1 which expects an operand, as would be the case, for instance, at the beginning of an expression, and the other starting at E2 which expects an operator or separator. Thus upon transferring control to E1 we will find the following set of productions:

```
E1    I        | →    E  |  EXEC 7      *E2
       . . . .
E2    <OP>  |              |  SUBR COM  *E1
```

At E1 the first production matches and control is transferred to EXEC 7 with the syntax units in the stack looking like E ↞ E | . EXEC 7 is roughly the same as EXEC 9 the main difference being that the description of the E in the RIGHT 1 position is set to contain a bit denoting that the value rather than the location of the variable is desired. So far X ← A has been scanned and converted to E ↞ E. We now scan the operator + and transfer control to E2 in the productions. The expression <OP> stands for a class of binary operators including the

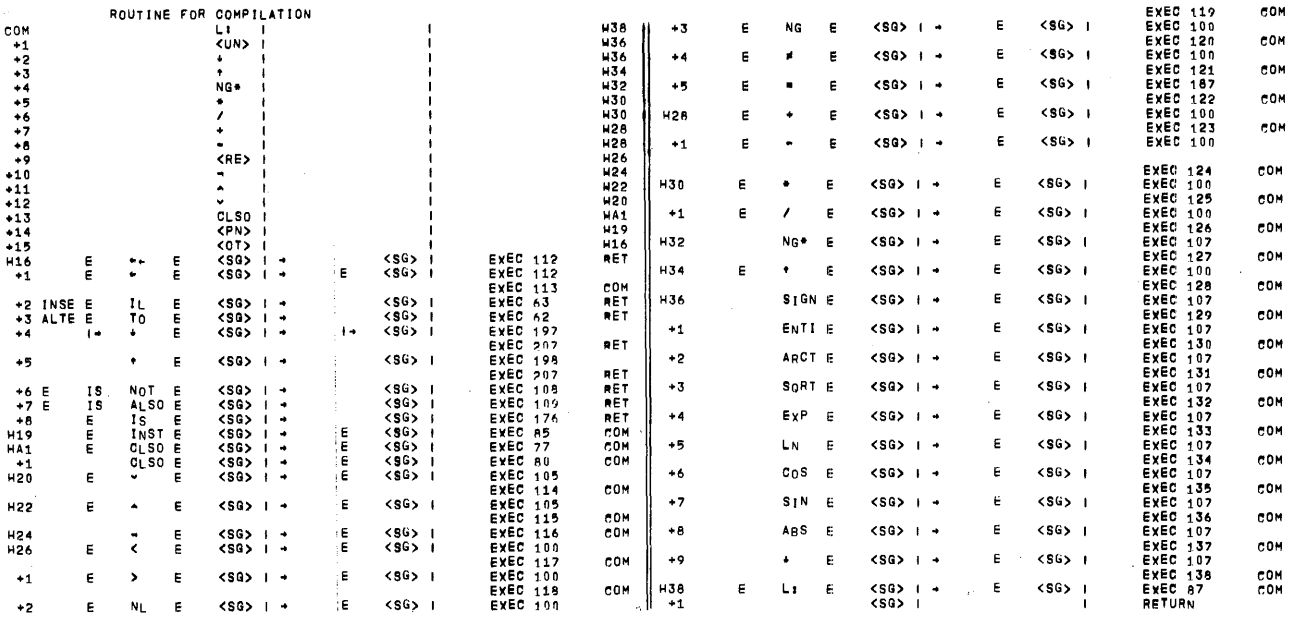


Figure 2. Subroutine COM.

plus sign so the production at E2 matches the + sign on top of the stack and control is transferred to the production subroutine labeled COM. This subroutine is given in Fig. 2. Notice that the expression <SG> in the LEFT1 position matches any arbitrary character.

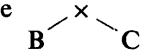
Subroutine COM is equipped with an operator precedence comparison mechanism for sorting on the hierarchies of operators so that, for example, in the expression A + B × C, code is compiled to perform the multiplication first and the addition second even though the order in which these operators are encountered in the syntax stack is the reverse. As we enter subroutine COM, production COM+7 matches and a transfer to H28 occurs. Nothing matches from H28 until the end, so control returns to the expression scanner which recognizes the next two characters and returns to subroutine COM with E ↞ E + E× | in the syntax stack. Then production COM+5 matches the stack, control passes to production H30, nothing matches until the end of subroutine COM, control returns to the expression scanner, two more characters are recognized, and a final transfer is made back to subroutine COM. At this point the configuration of the syntax stack is

$$E \twoheadleftarrow E + E \times E ; |$$

Here the expression <OT> matches the semicolon on top of the stack at production COM+15 and control passes to production H16. The first production to match the stack is production H30. This leads to the first instance of object code compilation in the processing of the statement. All previous actions up until this point have consisted of postponements. The compilation is accomplished by transfers to EXEC 100 and to EXEC 125 which compile code to multiply B and C. In the case of arithmetic operands CLA B is constructed. In the

MPY C

case of formula operands, code is produced to construct when executed the formula tree



The semantic routines used to accomplish this test the types of the operands, which types have been stored in the descriptions assigned by EXEC's 7 and 9, and they compile the appropriate code. At the completion of this compilation the syntax stack is

altered to look like E ← E + E;| because the terminal E×E has been replaced by a single E as is seen by inspecting production H30. The semantic routines also set the description of the topmost (rightmost) E to contain the type of the expression and the fact that the code compiled leaves the value of the expression in the run-time accumulator. Control now passes back to the beginning of subroutine COM for another iteration of the process. Subroutine COM will be seen to reenter itself iteratively until the entire expression is consumed, until code for it has been compiled, and until its external representation in the syntax stack has been replaced by E in the case of pure expressions and nothing in the case of assignment statements. We are now at the point where the syntax stack looks like E ← E + E;| and where we have reentered COM. On this pass production COM+15 matches and passes control to H16 where successive productions fail to match the syntax stack until production H28, at which point E + E is compiled by EXEC 100 and EXEC 123. The compile-time routines responsible for producing code detect the fact that code has been compiled leaving the value of the second operand in the run-time accumulator. Thus the code compiled is ADD A. Again the semantic routines analyze the types of LEFT2 and LEFT4 to determine whether code should be compiled to add numerical expressions or to add formula expressions. After compiling ADD A the stack configuration is changed to E ← E;| and control passes back to the beginning of subroutine COM. On this final trip through subroutine COM production H16 constructs code to perform the assignment of LEFT2 to LEFT4 and subroutine COM is exited with only the semicolon remaining in the syntax stack, the assignment statement having been consumed entirely. In the case of expressions, rather than assignment statements, an E is left upon exit in the RIGHT2 position with its semantic description set to contain its type and the fact that it resides in the run-time accumulator.

The strategy of subroutine COM comes from a well-known compiling technique for which no claim to originality is made. Both Floyd and Evans have used similar techniques in their Algol compilers.

FLEXIBILITY OF THE SYNTAX ANALYZER

We will now present examples showing how we can experiment with, extend, redesign or improve the syntax of the source language.

Suppose we want to add a new type of binary logical operator NOR (equivalent to the Pierce operator familiar to logicians and electrical engineers) and suppose we choose to denote it by the pair of characters ~ ∨ in the source language. Let's agree that in the expression ~A∧B ~∨ C ∨ D the NOR operator binds less tightly than ~, ∧, and ∨ so that fully parenthesized the expression looks like (((~A) ∧ B) ~∨ (C ∨ D)). We need to do four things to add this operator to the source language: 1) we declare the character pair ~ ∨ to be a primary syntactic unit for subscan; 2) we expand the definition of the class of operators < OP > to include ~∨ so that the production labeled E2 (above) will detect the presence of this new operator and will pass control to subroutine COM. The last two steps are additions to subroutine COM itself: 3) we insert the production

~ ∨ | | HNOR

after production COM+12 (cf. Fig. 2); and finally 4) we insert the production

HNOR E ~∨ E <SG> | → E <SG>
| EXEC K COM

after production HA1+1 (cf. Fig. 2). Here EXEC K must compile code for Boolean expressions using the NOR operator and it looks as follows:

K ↓ TEST[LEFT4, BOOLEAN] ∧ TEST[LEFT2, BOOLEAN] → CODE (RIGHT2 ← ~ (LEFT4 ∨ LEFT2)) : FAULT K ↓

This is our first example of the use of the code brackets CODE (. . .) . An expression contained in code brackets describes code to be generated and inserted into the object program. The test commands test the descriptions in the LEFT4 and LEFT2 positions to see if they contain bits denoting the type BOOLEAN.

From this example we see that we can 1) add operators and choose their hierarchies at will, 2) change the hierarchy of an operator without changing the operator itself, 3) by redefining an EXEC routine, change the meaning of an operator without changing its syntax, and 4) delete operators at will. This exemplifies the kind of experimental flexibility available to users of the FSL system.

The organization of the compiler is such that other kinds of additions, deletions and alterations may be performed in the syntax analyzer with ease. For example we may add a new type of variable by adding its declarator to various lists, by inserting a

production in the production subroutine that processes declarations, and by inserting tests and consequent compiling actions in the semantic routines which are called corresponding to expressions and statements using the new type of variable. It is also easy to add new types of statements to the statement scanner and to add new types of expressions to the expression scanner. The implementation of the list processing part of the Formula Algol language demonstrated to us the ease with which it was possible to extend the compiler. Unfortunately, space precludes an adequate description of this extension.

As a last example of a notational change we consider the evolution of the notation for formula patterns F INST P as defined in the first Formula Algol paper.[13] F INST P is a Boolean primary which tests whether the formula F is an instance of the formula pattern P. Later this notation was changed to F = = P and an additional type of test was added of the form F >> P to test whether F contains a subexpression which is an instance of P. In the case of changing INST to = = one merely had to substitute = = for INST in the productions. When >> was added it was possible with a minor correction to share the productions for = = to process >>. This correction consisted essentially of substituting a class symbol representing the set {= =, >>} for occurrences of = = in certain productions. The semantic routines associated with these productions were also altered to compile different code for the two different cases.

FLEXIBILITY OF THE SEMANTIC ROUTINES

Having treated some examples of the flexibility of the syntax analyzer we now turn our attention to corresponding properties of the semantic routines. The following example is intended to demonstrate the kind of experimentation that can be done with the compiling processes in order to improve both the compile-time efficiency and the parsimony of the formal description of the compiler. The example deals with a type discrimination problem encountered in the compilation of expressions involving unary and binary operators. For instance, when code is generated for A + B, the types of A and B must be checked.

REAL + REAL	is well defined, the operands are already complete, and the result is of type REAL.
REAL + BOOLEAN	is an illegal construction.
FORMULA + REAL	is well defined, but the right-hand operand must be made into a formula data term using a cell from linked list memory before a formula representing the sum can be constructed, and the result is of type FORMULA.

We shall describe two methods of implementing this type discrimination (which is the function of EXEC 100). The first is the most recent obsolete method and the second is its successor.

In the former method the set of operators was partitioned into a small number of equivalence classes. Two of these are the arithmetic operators +, −, ×, =, >, . . . and the Boolean operators ∨, ∧, ∼, ... For arithmetic operators, EXEC 100 checks the types of the operands for compatibility and sets a switch (MACHINE) to 2 if either operand is of type FORMULA, and to 1 if both operands are arithmetic. In the following version of EXEC 100, X7 is an address extractor.

```
100 ↓ TEST[LEFT4, BOOLEAN] ∨ TEST
        [LEFT2, BOOLEAN] → FAULT 100:
        RIGHT2 ← RIGHT2 ∧ X7;
      TEST[LEFT4, FORMULA] ∨ TEST
        [LEFT2, FORMULA] → MACHINE
        ← 2;
        ∼ TEST[LEFT4, FORMULA]
          → CODE( CONSTRUCT FORMULA
          [LEFT4]) $;
        ∼ TEST[LEFT2, FORMULA]
          → CODE ( CONSTRUCT FOR-
          MULA[LEFT2]) $ :
      MACHINE ← 1;
      TEST[LEFT2, REAL] ∨ TEST[LEFT4,
          REAL] → SET[RIGHT2, REAL]:
          SET[RIGHT2, INTEGER] $ $ $
```

As more operators and data types are added to a language this method becomes complex and inefficient both with regard to the space required to express the sequence of tests and the average time required to execute such a sequence. Therefore we have invented a successor to the above method.

The successor is described as follows. It is based on a single four-column table (DISCR) which may be preloaded. The first entry in a row of this table is a coded word which has three fields:

[TYPE1, OPERATOR, TYPE2]

Any combination of two types and an operator may be described in a single word. Unary operators have one of the two types fixed. The second entry indicates a compile time routine to be executed which makes the operands compatible. The third entry points to a compile time routine which actually compiles code for the expression. The final entry is the type of the result. DISCR may be initialized as follows:

```
REAL + REAL          none   ADD   REAL
REAL + INTEGER       none   ADD   REAL
FORMULA + REAL       CONS2  FADD  FORMULA
FORMULA ∨ BOOLEAN    CONS2  OR    FORMULA
NONE ~ BOOLEAN       none   NOT   BOOLEAN
```

When the syntax stack matches any production of the form

```
E <OP> E <SG> | → E <SG>
              | EXEC 189  COM
```

then the following code is executed:

```
189 ↓
        COMB ← (LEFT4 ×L12 + LEFT 3)
            × L6 + LEFT2;
        R ← DISCR[ COMB, $, , ];
        ~ SIGNAL → FAULT 189 :
        FINAL ← DISCR [0, , $ , ];
        TYPE ← DISCR [0, , , $ ];
        R ≠ 0 → MARKJUMP [R] $ ;
            JUMP[FINAL] $ ↓
```

Here L12 and L6 are shift constants, and LEFT3 contains a small but unique integer representing the operator.

In some cases we may experiment with the organization of compile-time processes to improve the quality of the object code produced, by which is meant we can reorganize some processes so that they produce less code which is more efficient. A small example of this is as follows. Certain patches of object code may be defined as nonexecutable because the flow of control may not enter them directly. For example, control must bypass a procedure declaration which may be entered only through a procedure call. If there are several adjacent procedure declarations then one may jump around each of them in turn or, preferably, one may jump around all of them simultaneously. The latter scheme is preferable because the object code requires less machine space, it runs slightly faster, and it looks less complex to the programmer trying to debug the system. The actual scheme for compiling these jumps has changed several times because we were able to try one method, tear it out, and try another both in the same day.

ORGANIZATIONAL EFFICIENCY IN THE COMPILER

When planning the structure of a compiler written in FSL we can take advantage of an organizational principle commonplace in programming, which states that when performing a class of operations which have certain common processing requirements we should, if possible, make a division of labor allowing the common processing requirements to be treated by a single shared routine. It is easy to apply this principle when writing EXEC routines since we can write a single EXEC routine performing labor common to several different compilation processes and we can share it in conjunction with other EXEC's to perform each separate process required. An example of this is EXEC 100 which is shared in the compilation of arithmetic expressions as is seen by looking at subroutine COM (Fig. 2). Another example is EXEC 160 which does everything common to procedures and blocks.

Production subroutines may also be shared. It may occur that certain syntactic constructs are used in different places in the source language with different semantics. For example, a list of identifiers can be used as a variable list in a declaration, as an array name list, as a formal parameter list, as a value list and as a specifier list. The productions in the syntax analyzer are written so that all identifier lists, no matter the context in which they occur, are processed by a common subroutine of the form:

```
ID          I | →  |  EXEC 190     * AID
          <SG> | →  |  ERROR 190     AID
AID         , | →  |                * ID
          <SG> | →  |  RETURN
```

As is seen, this production subroutine transfers control to EXEC 190 with the integer corresponding to the identifier on top of the stack. It does this for every identifier in the identifier list. In each of the different contexts of an identifier list it is necessary to process the identifier list in a different manner. To accomplish this, EXEC 190 is made into a variable capable of containing transfers to other EXEC's. For example, when, in FSL, the statement XEQ 190 ← XEQ 2 is encountered, it means the next time EXEC 190 is called, EXEC 2 will be executed. This will cause an identifier list to be processed as a variable list by the semantics. Similarly the statement XEQ 190 ← XEQ 3 will cause EXEC 190 to call EXEC 3 thus allowing an identifier list to be processed as a list of array names. By this mechanism one can treat the same syntactic

construct differentially in the semantics on the basis of context.

The addition of the XEQ construct to FSL is an example of the effect of feedback from the process of implementing Formula Algol on the design of FSL.

FORMULA MANIPULATION

It was decided to represent formulas inside the computer as trees or list structures built from cells taken from an available space list in a standard linked list memory. To add formula manipulation to the source language formula variables were introduced. In most cases the syntax already existing for numerical Algol was shared for formula manipulation. While no changes in the productions were necessary for this shared syntax, tests had to be added to the semantic routines to discriminate between numerical and formula compiling operations. For the new constructions added to the source language such as EVAL, = =, > >, and the extraction operator in patterns, additions were made to the productions and semantic routines were defined for them. Because most actions involving formulas are either interpretive at run time or involve manipulations which cannot be compiled into the object code as macros due to the size of the code involved, a set of run-time routines were constructed in machine code. These run-time actions constitute, more or less, a basic order code for formula manipulation. In effect, the compiler produces code for two machines, one an interpreter accomplishing formula manipulation and the other the hardware accomplishing numerical manipulation. For example, we saw (Fig. 2) that in subroutine COM EXEC 100 and EXEC 123 are called in sequence when we compile code to add two operands together, E + E. As we have seen previously EXEC 100 checks the types of the operands and sets a switch (MACHINE) specifying the machine for which we are to compile code. The structure of EXEC 123 is as follows:

```
123 ↓   MACHINE = 1 → CODE (RIGHT2
              ← LEFT2 + LEFT4):
          CODE (X1 ← LEFT2;  R0 ← '+';
            ACC ← LEFT4;
           MARKJUMP[CONSTRUCT
            FORMULA]; RIGHT2←ACC)
            $ ↓
```

Here the routine CONSTRUCTFORMULA is a basic operation of the formula manipulation machine which expects a right operand in X1, a left operand in the accumulator, and an operator in R0. Using cells from the list of available space it constructs a tree structure representing the sum of the operands and leaves the address of the head of this tree structure in the run-time accumulator.

The reader can now see how we could implement complex arithmetic by defining yet a third machine, which performs complex operations, and by extending the compiler by the same process used to accomplish the formula manipulation extension.

LIST PROCESSING

We will consider one example of list processing to try to convey some of the flavor of the mechanisms involved. Consider the statement INSERT [A, B, C] (AFTER LAST, BEFORE FIRST T) OF S. Here we assume that S contains a list (represented by a chain inside the computer). For the sake of specificity let S contain the list [V, T, V, V] where V and T have been declared of type SYMBOL. After the insertion statement is performed the list is to look like [V, A, B, C, T, V, V, A, B, C]. In a manner similar to that for formula manipulation a list processing machine is defined with an order code represented by a set of run-time routines. The compiler compiles a sequence of list processing instructions chosen from this order code corresponding to each list processing statement. Basic to the operation of the list processing machine is a push-down stack extant at run time called the *chain accumulator.* Most of the run-time list processing operations consist of manipulations on chains stored in the chain accumulator. The code produced by the compiler corresponding to the statement INSERT [A, B, C] (AFTER LAST, BEFORE FIRST T) OF S is a sequence of list processing operations whose mnemonics are as follows:

	Instruction	*Comment*
	STACK A	(on top of the chain accumulator)
	STACK B	(on top of the chain accumulator)
	CONCATENATE	(the top two chains in the chain accumulator)
	STACK C	(on top of the chain accumulator)
	CONCATENATE	(the top two chains)
	GO TO θ	
ρ:	CLA	symbol to denote *last*
	FIND POSITION	(this routine locates the last element of the chain

```
                    on top of the chain ac-
                    cumulator and stacks a
                    pointer to this element on
                    top of the chain accumula-
                    tor)
    PERMUTE         (we change the order of
                    the elements in the chain
                    accumulator so that the
                    pointer is moved into the
                    third position)
    CLA     T
    FIND POSITION   (locate the position of the
      MINUS ONE     cell in the chain on top of
                    the chain accumulator
                    which occurs directly be-
                    fore the first occurrence of
                    T in that chain; stack a
                    pointer to that position on
                    top of the chain accumu-
                    lator)
    PERMUTE         (as before, change the
                    order of the elements in
                    the chain accumulator so
                    that the pointer is moved
                    into the third position)
    PERFORM         (this routine performs in-
      INSERTIONS    sertions using information
                    saved in the chain ac-
                    cumulator)
    GO TO χ
θ:  STACK   S       (on top of chain accumu-
                    lator)
    TAKE            (replace S with the chain
      CONTENTS      that occurs as its contents)
    GO TO ρ
χ:  . . .
```

Let us now trace the effect of the execution of this code on the chain accumulator. We will adopt the symbolism that $| \phi$ represents the state of the chain accumulator before we start to execute the code. As we enter the code we build up the list [A, B, C] and stack it on top of the chain accumulator. This proceeds in the following steps. First we stack A on top of $| \phi$ producing $A | \phi$. Then we stack B on top of this producing $B | A | \phi$. Then we concatenate the top two chains on the chain accumulator producing $A \cap B | \phi$, where $\cap$ has been used as a symbol for the concatenation of chains. Next we stack C producing $C | A \cap B | \phi$, and then we concatenate again producing $A \cap B \cap C | \phi$. At this point the construction of the chain [A, B, C] is complete and control transfers to location θ in the code where we stack S producing $S | A \cap B \cap C | \phi$ and take its contents producing $V \cap T \cap V \cap V | A \cap B \cap C | \phi$. Control now returns to ρ where we compute and stack a pointer to the last element of the contents of S giving $o | V \cap T \cap V \cap V | A \cap B \cap C | \phi$. This pointer is moved to the third position in the chain accumulator producing $V \cap T \cap V \cap V | A \cap B \cap C | o | \phi$. A second pointer is now computed and stacked. It points to the position before the first T in the chain on top of the chain accumulator $o | V \cap T \cap V \cap V | A \cap B \cap C | o | \phi$. This pointer is also moved into the third position in the stack giving $V \cap T \cap V \cap V | A \cap B \cap C | o | o | \phi$. We could continue in this fashion computing and stacking as many pointers as we wish, each pointer corresponding to a place where an insertion is to be performed. We now transfer control to a routine which actually performs the insertions. This routine pops the chain $V \cap T \cap V \cap V$ from the top of the chain accumulator and inserts a copy of the chain $A \cap B \cap C$ after the position given by each pointer in the chain accumulator looping until all pointers in the chain accumulator are exhausted. The state of the chain accumulator after the execution of this statement is $| \phi$. Control in the code now passes to χ where the execution of the program continues. The reason for the existence of transfers in the code sample given is because the order of recognition of syntactic constructions in the insertion statement in the source language is the reverse of the order in which we utilize these constructions in the computation expressed by the code. Specifically we must stack S and compute its contents before we compute any pointers locating positions in the contents where insertions are to be performed. However, the constructions telling us where to make insertions are encountered in the source language before we encounter the expression telling us the object on which the insertions are to be performed. Floating addresses are used in the compiler to implement such reversals.

Since the semantics of the source language demands that all insertions be performed simultaneously we are forced to compute all locations where insertions are to be made before performing any insertions.

CONCLUSIONS

In this paper we have outlined the broad organization of the Formula Algol compiler. We have also presented examples exhibiting various proper-

ties of the FSL compiler writing system. We have not described completely or, in some cases at all, the implementations of declarations, switches, arrays, for statements, recursive procedures, block administration, formula manipulation or list processing. For a complete and detailed treatment of these the reader is referred to another paper by the authors, "The Implementation of Formula Algol in FSL."[14] The subject matter was chosen to reveal what we feel to be interesting techniques involving the use of a formal compiler writing system.

ACKNOWLEDGMENTS

The authors are deeply indebted to Professor Alan J. Perlis who guided and inspired our effort. Many of the creative and original ideas presented are his. However, we alone remain responsible for errors of style or content.

REFERENCES

1. R. Brooker and D. Morris, "A General Translation Program for Phrase Structure Languages," *Journal ACM*, vol. 9, p. 1 (1962).

2. J. C. Reynolds, "Cogent—A Compiler and Generalized Translator," Applied Mathematics Division, Argonne National Laboratory, internal paper.

3. R. Bolduc, T. E. Cheatham and A. Dean, "Preliminary Description of the Translator Generator System-I," Computer Associates, Inc. (Apr. 1964).

4. J. A. Feldman, "A Formal Semantics for Computer Languages," doctoral dissertation, Carnegie Institute of Technology (1964).

5. ____, "A Formal Semantics for Computer Languages and its Application in a Compiler-Compiler," *Communications of the ACM*, vol. 9, p. 3 (Jan. 1966).

6. A. J. Perlis, R. Iturriaga and T. Standish, "A Preliminary Sketch of Formula Algol," Carnegie Institute of Technology (July 1965).

7. R. W. Floyd, "A Descriptive Language for Symbol Manipulation," *Journal ACM*, vol. 8, p. 579 (1961).

8. A. Evans, "An Algol 60 Compiler," *Annual Review in Automatic Programming*, vol. 4, Pergamon Press.

9. P. Z. Ingerman, "THUNKS," *Communications of the ACM*, vol. 4, p. 55 (Jan. 1961).

10. K. Sattley, "Allocation of Storage for Arrays in ALGOL 60," *Communications of the ACM*, vol. 4, p. 60 (Jan. 1961).

11. A. Evans, "Syntax Analysis by a Production Language," doctoral dissertation, Carnegie Institute of Technology (1965).

12. R. Krutar, *FSL II*, Carnegie Institute of Technology, Computation Center, internal publication.

13. A. J. Perlis and R. Iturriaga, "An Extension to ALGOL for Manipulating Formulae," *Communications of the ACM*, vol. 7, p. 127 (Feb. 1964).

14. R. Iturriaga et al, "Implementation of Formula Algol in FSL," Carnegie Institute of Technology, Computation Center (Oct. 1965).

A PROPOSAL FOR A COMPUTER COMPILER*

Gernot Metze and Sundaram Seshu
Coordinated Science Laboratory and Department of Electrical Engineering
University of Illinois

INTRODUCTION

In recent years digital computers have been applied, with great success, to the automation of an increasing variety of tasks in the design of digital systems, from the printing of wiring tables and the drawing of logical diagrams to the optimization, according to certain criteria, of the layout of components and wiring, and even the actual computer-controlled production of subassemblies such as printed circuit boards or integrated circuits. Similarly, the design of circuits, especially those involving nonlinear elements, has been made easier by computer programs (e.g., which perform tolerance analyses). On the system level, the use of digital computers has been limited to tasks which are equally mechanical, such as programs which check for violations of fan-in, fan-out, and cascading rules.

More recently, languages have been developed which permit the simulation of a proposed system on an existing digital computer. Alternative system designs can be evaluated not only on the basis of performance statistics produced by the simulator, e.g., timing and utilization of machine components, but also by permitting the execution of programs written in the instruction language of the system being simulated.[1]

*Supported in part by the National Science Foundation under Grant GR-32 and in part by the Joint Services Electronics Program under contract number DA 28 043 AMC 00073(E).

The system designer, however, needs a language which is powerful enough to permit the description of the macroscopic structure of the system independent of the microscopic structure of its components. While the description of the system in this language may also be used for simulation purposes, the primary objective is the description of the relationships between system components in such a way that a compiler program can supply the detailed structure of the components, guided by certain design and optimization criteria which are stated explicitly or built into the program. Thus, the program should be a true compiler, and the system design language should permit a description of the system on a higher level than the languages proposed by Proctor,[2] Schlaeppi,[3] and Schorr.[4]

The linguistic aspects of a system design language, while interesting, have not been considered here, except that the language was developed in close analogy to the programming language FORTRAN, reflecting the feeling that computer programming and computer design are related fields. In particular, concepts such as modularity (subroutine structure), interfaces (subroutine argument linkages), parallel operations (multiprogramming), flow diagrams, etc., pervade both philosophies.

Just as FORTRAN translates arithmetic statements written in near-human language into a computer program, the computer compiler will translate a system description, given essentially in the near-human language of the programmer's manual, into a description of the hardware, e.g., ANDs, ORs,

NOTs, and FLIPFLOPs, and their interconnections.

GENERAL DESCRIPTION OF THE COMPILER

In comparison to manual approaches, the design of a digital system by compiler methods can be expected to be much faster and much cheaper, making it possible to examine (either externally or within the compiler itself) many alternative designs and select the one that is best according to some criterion. In particular, one could automatically examine the design for such features as speed, cost, or maintainability. If the input language is sufficiently powerful, the effect of adding special features such as buffered input/output, look-ahead controls, etc., can be examined with a minimum of changes to the specifications. Perhaps most important is the prospect that one good system designer can design the entire system, leading to a more uniform and more balanced result.

It is convenient to break the computer compiler program into two parts: a hardware-independent *system compiler* which reads the input language and produces an intermediate language output, similar to the assembly language output of most compilers, and a hardware-dependent *logic compiler* which reads the intermediate language and produces a detailed machine description in terms of the basic building blocks specified.

The system compiler incorporates several standard assembler features, such as "macro," "repeat," and "library," as well as the facility of interspersing intermediate language statements if desired. The intermediate language output is a microinstruction string for each subprogram (subcontrol), optimized according to a specified measure. Although the system compiler is otherwise hardware-independent, this measure may involve hardware cost. The microinstruction string output of the system compiler includes a specification of the time-hierarchy and is thus equivalent to a flow chart.

The intermediate language may also be used to drive a simulator program which permits experimental programming in the instruction language of the proposed system.

The specification and the development of the logic compiler is fairly straightforward conceptually. The Boolean minimizations required introduce bookkeeping problems but no other difficulties. The logic compiler is not discussed any further in this preliminary report.

The concept of a library subroutine enters the discussion of a computer compiler in two distinct ways. The conventional notion is similar to that of a subcontrol (e.g. arithmetic control, I/O control), but in addition open subroutines ("built-in functions" in FORTRAN terminology) may be used, such as algorithms for arithmetic operations. However, one could now have several algorithms which are equivalent in their final answers, say for division in two's complement representation, and ask the compiler to choose the algorithm which fits best with the rest of the design. Thus one may want to call for *any* subroutine from a class of subroutines which is identified by a class name. With a sufficiently rich library one could conceive of "dime a dozen" designs that one could choose from.

The proposed compiler is also a good research tool. Since designs can be produced simply, one could produce examples rapidly to study new design ideas. Finally, the concepts generated here might well suggest procedures for the synthesis of non-computer systems thus providing a formal basis for "systems engineering."

THE INPUT LANGUAGE

The description of a digital system involves two aspects: the global description, and the subcontrol (subprogram) description. The subdivision of the system into subcontrols is similar to the subdivision of a program into subprograms, and must be done by the system designer. (Thus we implicitly seek modular designs.) However, in contrast to conventional programming, subcontrols may operate in parallel, i.e., simultaneously, thus giving rise to the need of a global description of the system. Counterparts to these concepts will become necessary when multiprogramming compilers are written.

Global Description

The global description carries the special identifier

MACHINE xxxx

followed by the following types of global headers:

1. Definitions of global constants by the operation SYN (see register declarations below) which define word length, memory size, etc.
2. Declaration of subcontrols which may operate in parallel.
3. Information necessary for optimization, such as cost, time and other measures.

There are no program statements in the global description.

Subcontrol Description

Each subcontrol description has an identifier and the necessary header statements followed by the instruction statements, i.e., the program, which describe the subcontrol:

1. The *identifier* is a statement of the type

SUBCONTROL ARITH

where ARITH is a name chosen by the designer. (The subroutine linkage mechanism is further discussed under Subroutine CALLs below.)

2. *Register Declarations* are analogous to DIMENSION and COMMON statements in FORTRAN except that we follow the machine language convention and demand that even single bit registers be declared. There are five statement types in this category:

a) REGISTER A(L) defines a register A of L bits where L is an integer or a previously defined symbol. Individual bits in the register are referred to by subscripts which normally range from 0 to (L − 1). Other ranges of consecutive subscripts must be specified explicitly, as for example in

REGISTER A(−1,...,L − 2).

b) SYN (F,N) assigns the value N, which must be a positive integer, to the symbol F, which may then be used in register definitions and subscripts.

c) CONNECT (EAQ(−1,...,38)) = ES.A (0,...,19). Q(1,...,19) permits the concatenation of registers. The registers on the right must have been previously defined but need not be full registers. In the example above, Q(0) is not part of the extended AQ register.

d) EQUIV (FNCTN(0,...,9) = IR(3,...,13)) labels (a part of) a register by another name and is thus the inverse of CONNECT.

e) INTERFACE (ARITH) M,A,Q defines registers M, A, Q as interface registers in common between the current control and the subcontrol ARITH. The INTERFACE statement is similar to the FORTRAN COMMON statement but differs from it in two respects. First, several subcontrols in a machine may be operating simulataneously, which is not the case in present programs. Since the compiler would normally try to use existing registers for temporary storage, it must be aware of the interface registers which may be used by parallel controls. Secondly, for the convenience of the logic compiler as well as for readability, the alternate control with which the register is shared should be identified. Interface registers must be dimensioned by REGISTER, CONNECT, or EQUIV statements, and must be referred to by the same names, in each of the subcontrols which share them. However, the order in which they are listed in the INTERFACE statement is not important.

3. *Instruction Decoding.* The assignment of bit configurations for the various instructions is a task that is best left to the logic compiler. We therefore allow the design engineer to use mnemonics for instructions. There are two types of instructions involved. First we have the instructions that are to be decoded and obeyed by the current subcontrol. Second there are instructions to be given to other subcontrols (for example, main control may request a memory subcontrol to read or write a word). In the first case, we need to decode and jump to the appropriate control sequence. In the second case, we need only set up a configuration of bits in an appropriate register. In both cases, the function is undefined. We must, however, specify (to the logic compiler) the bits that are to be used to define the function.

a) The format of the decode and jump statement is

DECODE (IR(0,...,9))
HLT, LLS, LRS, JMP, JAN,...

where IR(0,...,9) is (part of) a previously defined register and HLT, LLS, etc., are mnemonics which must appear as location field symbols in the main program. The DECODE statement is itself part of the main program since it serves as a multi-way branch, analogous to a computed GO TO.

b) The format of a translation (or decoder) statement is

UNDF (IR(0,...,9)) RM, WM, RMW

Here IR is a previously defined register and RM, WM and RMW are instructions to be passed on to other subcontrols. The system compiler generates a decoder for each such UNDF statement. Each decoder is defined in detail by the logic compiler. UNDF is a header statement.

The mnemonics on the right of the parentheses in both statements must be single-valued Boolean functions of the bits that are enclosed within the parentheses. For example, consider the execution of the instruction REPLACE ADD MEMORY, which replaces the contents of the memory cell by the sum of the previous contents and the contents of the

accumulator. We need to set up first a READ MEMORY (RM) instruction and then a WRITE MEMORY (WM) instruction in the instruction register of the memory subcontrol. If FN is the function part of the main instruction register, we cannot write

UNDF (FN), RM, WM

for RM and WM are not single-valued functions of FN alone. Some control flip-flop is also involved and must therefore be defined as

REGISTER CN (1)
UNDF (FN, CN) RM, WM

The symbols that are used on the right must appear exactly once in the DECODE statement of another subcontrol to permit correlation by the logic compiler.

4. *Program Statements*, which may contain a label, include the following types:

a) Register Transfers. The gating of information from a register A to a register B is specified by

B = A.

The symbol on the right must either be a register or an undefined function. Partial register transfers are indicated by subscripting. Gating is assumed to be parallel.

b) Branch Statements.

i) DECODE, the counterpart to a computed GO TO, has been discussed in the preceding section.

ii) An unconditional branch is indicated by simply writing the symbol (without the words GO TO).

iii) A conditional branch is indicated by

IF (A(O) = 1) JMP

where the true exit is the statement labeled JMP, the false exit the next statement. The condition must be based on a single bit being 0 or 1.

iv) The WAIT statement is similar to the IF statement, except that the true exit is the next statement, and the false exit is the WAIT statement itself, e.g.,

WAIT (RQ = 1)

permits the subcontrol to go on to the next statement only after RQ has been set to 1.

c) SET and CLEAR permit individual bits, or entire registers, to be set to 1, or cleared to 0. Subscripts are allowed.

d) SUBROUTINE INSERTIONS are accomplished by writing the name of the subroutine, with the argument list in parentheses. Both library subroutines and programmer-defined subroutines are treated as macros. Subroutines may be called by their class name if the choice of the particular subroutine is to be left to the compiler.

e) Subcontrols call other subcontrols through the statement CALL. Since subcontrols may be parallel or sequential (see the following section), and one would like to be free to define them either way by means of global headers, we provide three formats for the CALL statement:

CALLS SUB(RQ)
CALLP SUB(RQ)
CALL SUB(RQ)

By convention the argument in parentheses (RQ) is the name of the request flag. The terminals S and P designate the CALL as sequential or parallel and override the global definition. In the simple CALL, the global definition prevails. In each case a string of statements which load interface registers follows the CALL statement, terminated by an ENDC. Consider the following example of a main control to core control CALL (Store Accumulator instruction):

```
CALL CORE (MCRQ)
MCAR = ADDR
M = A
MCIR = WM
ENDC
```

If the memory control is defined as sequential in the global headers the compiler produces the microinstruction string

```
GATE MCAR = ADDR
GATE M = A
DECODER 2 = WM
CONNECT DECODER 2 to MCIR
SET MCRQ
WAIT (MCRQ = 0)
```

If on the other hand the global definition states that the memory control operates in parallel with main control, the following microinstruction string

results:

```
WAIT (MCRQ = 0)
GATE MCAR = ADDR
GATE M = A
DECODER 2 = WM
CONNECT DECODER 2 to MCIR
SET MCRQ
```

5. *The Slash Notation.* One of the most common operations in a computer is to read a word from memory into a register or store a word from a register into memory. Therefore we invent a special shorthand notation for this purpose. The notation /REG/ refers to the memory location whose address is in register REG. Thus

```
IR = /P/
```

states that the word whose address is in the program counter P is to be read and loaded into the instruction register IR. Similarly

```
/ADDR/ = A
```

states that the contents of A are to be stored in the memory location whose address is in register ADDR.

Naturally the compiler must be given the interpretations of the two statements by means of macro definitions. This macro is given the special name MEMORY. Since the memory address and buffer registers are unique to the calling program, this memory definition must be part of the calling program. Alternatively it may also be defined in detail in the global headers as a macro with a local macro MEMORY (calling the global one) defining the interface registers.

PARALLEL AND SEQUENTIAL SUBCONTROLS

As remarked earlier, a subcontrol is similar to a subprogram. Thus one intuitively expects to use some type of LINK JUMP (or RETURN JUMP). Since a subcontrol may be called from several places (in the same or different controls) it appears intuitively necessary to store the calling address in some register. If such a procedure were followed, the subcontrol would have to interpret the contents of this register and return to the calling point. The FORTRAN analog is the ASSIGNED GO TO. This technique is aesthetically unappealing since the subcontrol has to know the various points from which it can be called—an impractical procedure for library routines. Also the notion of a parallel subcontrol has no exact analog in subroutines. An interrupt subroutine comes close but a more exact analogy is the communication between two computers. In both of these cases, the standard communication technique is the use of flags rather than LINK JUMPs.

Thus parallel subcontrols must be initiated into action by means of a flag flip-flop and must similarly indicate the completion of the action by a flag flip-flop. There appears to be no reason why these two flip-flops could not be the same. We label it the REQUEST flip-flop. By convention the CALL sets the REQUEST and the subcontrol clears it when it is through.

One would like to be able to write library subroutines (subcontrols) with the parallel/sequential consideration. The executive program (or in our case, the global headers) should decide whether the subcontrol is to be used as a parallel or a sequential subcontrol. The REQUEST convention permits one to achieve this objective.

There is one further distinction between parallel and sequential usage which must be mentioned. If a parallel subcontrol can be called from two (or more) other subcontrols then it should have two (or more) sets of interface registers and request flip-flops. If RQ1, RQ2,... are the different request flip-flops, then a subroutine can serve a fixed hierarchy of requests by the statement

```
WAIT (RQ1 + RQ2 + ··· = 1)
```

where + denotes the Boolean OR. Similarly, a scanning procedure can be arranged by using a string of IF statements. In order to make it possible to write library subroutines independent of the *number* of requests, a simple extension of the INDEFINITE REPEAT feature of macro compilers may be used, with the necessary information carried in the library call.

In a sequential subcontrol multiplicity of interface registers is not necessary. It may be used without harm, of course.

We may finally note one distinction between intercomputer communication and parallel subcontrols. If two computers are tied together, either computer may request action by the other (and conflicts are somehow resolved). In our case, however, the standard hierarchical structure of programming must be observed. If subcontrol A can call on sub-

control B, subcontrol B may not call on subcontrol A. Thus the problem is simpler.

TIME AND CONTROL HIERARCHIES

We have implicitly noted that there are two notions of hierarchy among subcontrols. The different subcontrols form a partially ordered set under the relation of extended CALL. As in conventional programming we insist that this relation define a true partial ordering. Beyond this fact, however, we are not too concerned with this logical hierarchy.

A second partial ordering, which is not ab initio a partial ordering but may be converted into one, is by time of operation. If two subcontrols may operate at the same time they are at the same level in this partial ordering independent of the logical hierarchy. They are parallel subcontrols in our earlier terminology. Consider for example a main control, a buffered input/output subcontrol and a memory subcontrol. Since I/0 is buffered, it may operate at the same time as main control. Since the I/0 subcontrol may call on memory subcontrol, the memory subcontrol may operate at the same time as main control. Thus the logical hierarchy is that shown at the left in Fig. 1, while the time hierarchy is the one shown at the right.

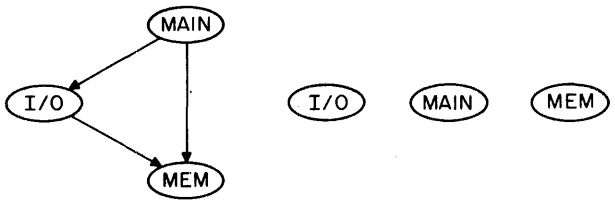


Figure 1. Hierarchy of subcontrols (Hasse diagrams): Logical (*left*); Time (*right*).

(Formally, "parallel" is a binary relation which we extend by transitivity. The partial ordering relation in the time diagram is "not parallel, and below in logical ordering.")

THE LIBRARY

The library material for the compiler should contain two classes of programs: subroutines and subcontrols. A subroutine is an "open" subroutine or a macro. Algorithms for arithmetic operations, incrementors, etc., come under this category. These algorithms are divided into types and are called by type names; the detailed choice is left to the compiler. Subcontrols are complete subprograms but are processed according to macro conventions, that is, they are stored in source language with dummy dimensions and dummy register names. They may contain such macro features as "indefinite repeat," "If True," "If False," etc. An example of the library call for such a library subcontrol is

```
ARITH     LIB AC5(A, Q, OV, N)
```

where AC5 is the identifier of the library routine, ARITH is the name assigned in the machine, A, Q and OV are registers and N is the dimension (defined in global headers).

It is sometimes necessary to label the "next statement while using the indefinite repeat directive. An example is a "scanner" which services requests in sequence. For this purpose we introduce the CONTINUE statement. An example follows:

```
          SUBROUTINE CORE (RQ, M1, MAD, MI, C1, AL, WL)
          REGISTER MAR (AL), MBR (WL)
          IRP (RQ, M1, MAD, MI, C1)
          REGISTER RQ(1), M(WL), MAD(AL), MI(1)
STM       IF (RQ = 0) C1
          DECODE (MI) RD, WR
RD        MAR = MAD
          .CORERD
          M1 = MBR
          CLEAR RQ
          C1
WR        MAR = MAD
          MBR = M1
          CLEAR RQ
          .COREWR
C1        CONTINUE
          IRP
          STM
          END
```

If there are three controls which wish to use this memory control in parallel, one may use the library call

```
LIB CORE ((RQ1, M1, MA1, MI1,),
          (RQ2, M2, MA2, MI2,),
          (RQ3, M3, MA3, MI3,), AL, WL)
```

In the subroutine, .CORERD and .COREWR are library subroutines which set up the signals for reading and writing core memory. We note that C1 is a repeated argument which has not been specified in the library call. Hence it becomes a created symbol, a different symbol for each repetition. On the other hand, STM is not an argument. Hence this symbol is assigned to the first occurrence. The repetition IRP uses simultaneous substitutions for all arguments. (This is the simple extension referred to earlier.) The CONTINUE statement is not

translated; its label is assigned to the "next" microinstruction.

It is easily verified that the "IF, CONTINUE" arrangement in the subroutine is in fact a scanner.

THE MICROLANGUAGE (OUTPUT LANGUAGE OF THE SYSTEM COMPILER)

The output of the system compiler is a preamble followed by a string of microinstructions. The preamble contains the information necessary for the logic compiler. Wherever possible, one would like to perform microoperations in parallel. For this purpose the system compiler will associate an ordered-pair level index with each microinstruction and specify in the WAIT field the ordered pair indices of the microsteps which must be previously completed. Thus the output becomes a description of the flow diagram.

The microlanguage is permissible in the source program as well (without the ordered-pair indices, of course). In fact, arithmetic algorithms have to be written in microlanguage. In the source program a switch to the microlanguage is initiated by

— MICRO

and terminated by

— COMPILE.

All arithmetic operations in the microlanguage are Boolean. The conventions are

A + B	A OR B
(−A)	NOT A (outer parentheses essential)
A *B	A AND B
A (+) B	A EXCLUSIVE OR B

Other operations can be added later. Subscripts are allowed, and the RANGE of symbolic subscripts may be specified.

An equality sign denotes a definition. If the variable on the left is a flip-flop, the quantity on the right decides whether the flip-flop is set (1) or cleared (0). Otherwise the equation is taken as a signal definition (decoder output for example).

Other microoperations (all self-explanatory) are:

1. GATE RA = RB
2. IF (BIT = 1 (or 0)) LABEL
3. OFF GATE
4. STOP

SAMPLE DESIGN OF A SMALL DIGITAL SYSTEM

In order to demonstrate the versatility and power of the input language, we present here the system design of a small digital computer with a sequential arithmetic subcontrol and a parallel input/output subcontrol which handles one-word transfers to and from memory.

The card format is essentially that of FORTRAN. A detailed discussion of the example will be found in the following section. The register layout and data paths are shown in Fig. 2.

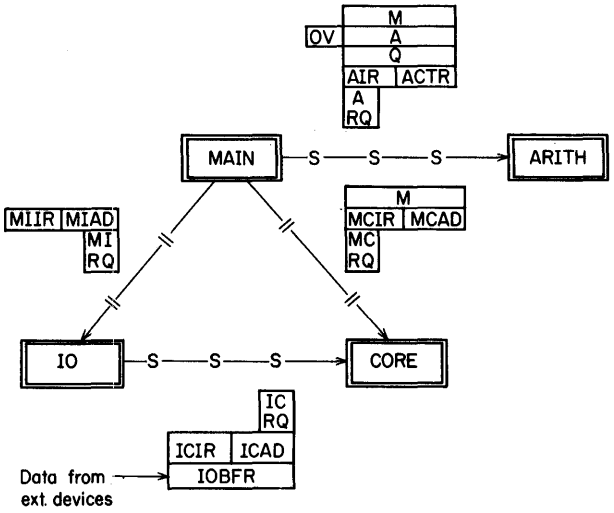


Figure 2. System layout and data paths in the sample computer.

```
      MACHINE CSLIAC
*     GLOBAL HEADERS
      SYN (WL,20), (DWL,39), (AL,10), (FL,10), (AAL,5), (AFL,3),
     1    (IOFL,1), (CFL,1)
*     LENGTHS OF REGISTERS ARE DEFINED AS FOLLOWS:
*     WL = SINGLE WORD, DWL = DOUBLE WORD, AL = ADDRESS, FL = FUNCTION,
*     AAL = ARITHMETIC ADDRESS, AFL = ARITHMETIC FUNCTION,
*     IOFL = INPUT/OUTPUT FUNCTION, CFL = CORE MEMORY FUNCTION
      PARALLEL (MAIN, IO), (MAIN, CORE), (IO, CORE)
*     MEMORY READ DEFINITION
      MACRO MEMRD (X,Y,AD,DT,IR,RQ)
*     X IS SOURCE REGISTER FOR MEMORY ADDRESS, Y IS DESTINATION REGISTER FOR
*     CONTENTS (OPERAND OR INSTRUCTION), AD IS INTERFACE ADDRESS REGISTER,
*     DT IS INTERFACE DATA (OPERAND OR INSTRUCTION) REGISTER, IR IS INTERFACE
*     MEMORY INSTRUCTION REGISTER, RQ IS REQUEST FLAG.
*     NAME OF MEMORY CONTROL IS CORE.
      CALL CORE (RQ)
      AD = X
      IR = RCH
      ENDC
      IFF (Y = DT)
      Y = DT
      ENDM
      MEMORY WRITE DEFINITION.  SAME ARGUMENT LIST
      MACRO MEMWR(X,Y,AD,DT,IR,RQ)
      CALL CORE (RQ)
```

```
      AD = X
      IR = WCM
*     IFF (Y = DT)
*     DT = Y
      ENDC
      ENDM
      END
      CONTROL MAIN
      REGISTER IR(WL), P(AL), RUNSW(1), CN(1),
     1         M(WL), A(WL), Q(WL), ACTR(AAL), AIR(AFL), ARQ(1), OV(1),
     2         MIAD(AL), MIIR(IOFL), MIRQ(1),
     3         MCAD(AL), MCIR(CFL), MCRQ(1)
      INTERFACE(ARITH) M, A, Q, ACTR, AIR, ARQ, OV
      INTERFACE (IO) MIAD, MIIR, MIRQ
      INTERFACE (CORE) M, MCAD, MCIR, MCRQ
*     DEFINE RIGHTMOST AL BITS OF INSTRUCTION AS ADDRESS PART
      EQUIV (ADDR = IR(WL-AL,...,WL-1))
*     DEFINE LEFTMOST FL BITS OF INSTRUCTION AS FUNCTION PART
      EQUIV (FNCTN = IR(0,...,FL-1))
*     ARITHMETIC FUNCTIONS
      UNDF (FNCTN) ALLS, ALRS, AADD, ASUB, AMUF, ADVF
*     I/O FUNCTIONS
      UNDF (FNCTN) IORD, IOWR
*     CORE FUNCTIONS
      UNDF (FNCTN, CN) RCM, WCM
*     SLASH NOTATION DEFINITION
      MEMORY Y = /X/
      MEMRD(X,Y,MCAD,M,MCIR,MCRQ)
      ENDM
      MEMORY /X/ = Y
      MEMWR(X,Y,MCAD,M,MCIR,MCRQ)
      ENDM
*     MAIN PROGRAM
EXIT  INCR (P,P,AL)
BEGIN IR = /P/                    FETCH NEXT INSTR
XEQ   CLEAR CN
      DECODE (FNCTN) HTR, LLS, LRS, JMP, JAN, LDA, STA, LDQ, STQ, SAD,
     1               LJP, JOV, ADD, SUB, MUF, DVF, INP, OUT, JIO
HTR   WAIT (RUNSW = 1)            HALT TRANSFER
      JMP
LLS   CALL ARITH (ARQ)
      AIR = ALLS                  LONG (ARITH) LEFT SHIFT
      ACTR = ADDR (AL-AAL,...,AL-1)
*     ACTR USES RIGHTMOST AAL BITS OF ADDRESS
      ENDC
      EXIT
LRS   CALL ARITH (ARQ)
      AIR = ALRS                  LONG (ARITH) RIGHT SHIFT
      ACTR = ADDR (AL-AAL,...,AL-1)
      ENDC
      EXIT
JMP   P = ADDR                    JUMP (UNCONDITIONAL)
      BEGIN
JAN   IF (A(0) = 0) EXIT          JUMP ON A NEGATIVE
      JMP
LDA   A = /ADDR/                  LOAD ACCUMULATOR
      EXIT
STA   /ADDR/ = A                  STORE ACCUMULATOR
      EXIT
LDQ   Q = /ADDR/                  LOAD Q
      EXIT
STQ   /ADDR/ = Q                  STORE Q
      EXIT
SAD   M = /ADDR/                  SUBSTITUTE ADDRESS
      M(WL-AL,...,WL-1) = A(WL-AL,...,WL-1)
      SET CN
      /ADDR/ = M
      EXIT
LJP   .INCR (P, P, AL)            LINK JUMP
      M = /ADDR/
      M(WL-AL,...,WL-1) = P
      SET CN
      /ADDR/ = M
*     PLANT CONTENTS OF P IN RIGHTMOST AL BITS OF MEMORY WORD
      .INCR (ADDR, P, AL)
      BEGIN
JOV   IF (OV = 0) EXIT            JUMP ON OVERFLOW
      CLEAR OV
      JMP
ADD   CALL ARITH (ARQ)            ADD
      AIR = AADD
      M = /ADDR/
      ENDC
      EXIT
SUB   CALL ARITH (ARQ)            SUBTRACT
      AIR = ASUB
      M = /ADDR/
      ENDC
      EXIT
MUF   CALL ARITH (ARQ)            MULTIPLY A BY M (FRAC)
      AIR = AMUF
      M = /ADDR/
      ENDC
      EXIT
DVF   CALL ARITH (ARQ)            DIVIDE (FRAC) AQ/M
      AIR = ADVF
      M = /ADDR/
      ENDC
      EXIT
INP   CALL IO(MIRQ)               INPUT VIA IO
      MIIR = IORD
      MIAD = ADDR
      ENDC
      EXIT
OUT   CALL IO(MIRQ)               OUTPUT VIA IO
      MIIR = IOWR
      MIAD = ADDR
      EXIT
JIO   IF (MIRQ = 0) EXIT          JUMP IF IO BUSY
      JMP
      END
      SUBCONTROL ARITH
      REGISTER M(WL), A(WL), Q(WL), ACTR(AAL), AIR(AFL),
               ARQ(1), OV(1), ES(1)
      INTERFACE (MAIN) M, A, Q, ACTR, AIR, ARQ, OV
      CONNECT (EAQ(-1,...,DWL-1) = ES.A.Q(1,...,WL-1))
      EQUIV (AQ = EAQ(0,...,DWL-1))
STRT  WAIT (ARQ = 1)
      DECODE (AIR) ALLS, ALRS, AADD, ASUB, AMUF, ADVF
ALLS  IF (ACTR = 0) EXIT
      .DECR (ACTR, ACTR, AAL)
      EAQ(-1,...,DWL-2) = EAQ(0,...,DWL-1)
      EAQ (DWL-1) = 0
      ALLS
ALRS  IF (ACTR = 0) EXIT
      .DECR (ACTR, ACTR, AAL)
      EAQ(1,...,DWL-1) = EAQ(0,...,DWL-2)
*     LEAVES POSITIONS 0 AND ES UNTOUCHED
      ALRS
AADD  .ADD2 (A,M,A,OV,WL)
      EXIT
ASUB  .SUB2 (A,M,A,OV,WL)
      EXIT
AMUF  .MUF2 (A,M,AQ,OV,WL)
      EXIT
ADVF  .NDVF (AQ,M,Q,A,OV,WL)
      EXIT
EXIT  CLEAR ARQ
      STRT
      END
```

```
        SUBCONTROL IO
        REGISTER IOBFR (WL),
      1          MIAD (AL), MIIR(CFL), MIRQ(1),
      2          ICAD(AL), ICIR(CFL), ICRQ(1)
        ICRQ IS REQUEST FROM IO TO CORE, MIRQ IS REQUEST FROM MAIN TO IO
        INTERFACE (MAIN) MIAD, MIIR, MIRQ
        INTERFACE (CORE) IOBFR, ICAD, ICIR, ICRQ
        UNDF (ICIR) IRC, IWC
START   WAIT (MIRQ = 1)
        DECODE (MIIR), IORD, IOWR
 IORD    IOBFRRD (IOBFR)              GATES INTO BUFFER FROM EXT. DEVICE
        CALLS CORE (ICRQ)
        ICAD = MIAD
        ICIR = IWC
        ENDC
        CLEAR MIRQ
        START
 IOWR   CALLS CORE (ICRQ)
        ICAD = MIAD
        ICIR = IRC
        ENDC
        CLEAR MIRQ
        .IOBFRWR (IOBFR)             GATES FROM BUFFER TO EXT. DEVICE
        START
        END
        SUBCONTROL CORE
        REGISTER MBR(WL), MAR(AL),
      1          M(WL), MCAD(AL), MCIR(CFL), MCRQ(1),
      2          IOBFR(WL), ICAD(AL), ICIR(CFL), ICRQ(1)
        INTERFACE (MAIN), M, MCAD, MCIR, MCRQ
        INTERFACE (IO), IOBFR, ICAD, ICIR, ICRQ
START   WAIT (MCRQ + ICRQ = 1)
        IF(ICRQ = 1) IOD
        DECODE (MCIR) RCM, WCM
 IOD    DECODE (ICIR) RCI, WCI
 RCM    MAR = MCAD
        .CORERD
        M = MBR
        CLEAR MCRQ
        START
 WCM    MAR = MCAD
        MBR = M
        CLEAR MCRQ
        .COREWR
        START
 RCI    MAR = ICAD
        .CORERD
        IOBFR = MBR
        CLEAR ICRQ
        START
 WCI    MAR = ICAD
        MBR = IOBFR
        .COREWR
        CLEAR ICRQ
        START
        END
    SUBROUTINE .RIPLADD (X,Y,Z,OFL,WL), CLASS .ADD2
    X = AUGEND, Y = ADDEND, Z = SUM, OFL = OVERFLOW FLAG
    RADIX COMPLEMENT ADDITION WITH RIPPLE CARRY
    REGISTER X(WL), Y(WL), Z(WL), OFL(1), SX(1), SZ(1)
    IFT (X = Z)
    REGISTER TX(WL)
    IFF (X = Z)
    EQUIV (TX = Z)
    -- MICRO
    C(WL-1) = 0
    D(WL-1) = 1
    RANGE I = 0, WL-1
    C(I-1) = X(I)*Y(I)*D(I) + (X(I)+Y(I))*C(I)
    D(I-1) = (-X(I))*(-Y(I))*C(I) + ((-X(I)) + (-Y(I)))*D(I)
    WAIT (C(-1) + D(-1) = 1)
    ASSERT (C = (-D))
    S(I) = X(I) (+) Y(I) (+) C(I)
    GATE TX = S
    OVS = C(0) (+) C(-1)
    GATE OFL = OVS
    IFT (X = Z)
    GATE X = TX
    -- COMPILE
    END
```

DISCUSSION OF THE EXAMPLE

The purpose of the global header subprogram with the identifier "MACHINE" is to provide maximum flexibility in the system design. One can change word length or memory size simply by changing one synonym. One can change from buffered I/0 to sequential I/0 by removing one statement. One can change from one type of memory to another by changing macros MEMRD and MEMWR. We note incidentally the convention ENDM as macro termination and the use of macros within macros.

The statements IFF and IFT mean "IF FALSE" and "IF TRUE" and cause conditional compilation of the next statement (MAP convention). All subroutines whose names start with a period are assumed to be library subroutines (or classes of them). The subroutines used have the following interpretations:

.INCR (SOURCE, DESTINATION, LENGTH): DESTINATION = SOURCE + 1

.DECR (SOURCE, DESTINATION, LENGTH): DESTINATION = SOURCE − 1

.ADD2 (OPERAND 1, OPERAND 2, DESTINATION, OVERFLOW, WORD LENGTH): 2's complement addition.

.SUB2, .MUF2, .NDVF2 have the same arguments as .ADD2 and refer to 2's complement subtraction, multiplication and non-restoring division.

.IOBFRRD AND .IOBFRWR are subroutines to activate external I/0 devices to read into and write out of the IO buffer register, respectively. Similarly .CORERD and .COREWR generate signals to read from and write into the core memory.

Since we have adopted the IPLV execution convention for GO TO, certain statement tags are not admissible. These are: DECODE, CALL, MACRO, ENDC, ENDM, IF, IFF, IFT, IRP, END, WAIT, CLEAR, SET, REGISTER, INTERFACE, PARALLEL, etc.

Special Points to be Noted

Main Control: ACTR is a counter register for arithmetic control. RUNSW is a flip-flop controlled by console switches. When it is off (RNSW = 0), the instruction HTR stops the main control. However, the I/0 and memory controls may continue to run. In SAD and LJP we could have used a read/modify/write procedure if the memory control had such capability. For example, if RMW is the appropriate core memory instruction, the appropriate string for SAD is:

```
CALL CORE (MCRQ)
M = A
MCAD = ADDR
MIR = RMW
ENDC
```

One would define such a string by a macro in the global description.

Note that register M appears in the interface between MAIN and ARITH, as well as in the interface between MAIN and CORE. This is permissible in this case since MAIN calls ARITH sequentially.

Arithmetic Control: We should note the formation of the long EAQ register by the use of CONNECT. The sign bit of Q is excluded in this long register and the sign bit of A is extended by one bit (ES).

I0 Control: We note the use of CALLS when subcontrol CORE is called. In the global definition, IO and CORE have been defined as parallel as they could operate at the same time. However, when CORE is called by IO, we have to wait for the CORE control to finish. In reading from an IO device, we have to signal MAIN that the word has been written into memory; in writing to an IO device, we need the word from memory to write. Thus in both cases a WAIT (ICRQ = 0) is necessary after the initiation of a memory request. If a simple CALL were used, the ENDC would have to be followed by such a wait, whereupon the first WAIT (ICRQ = 0) produced by the compiler becomes redundant. (Incidentally, in such a case the .IOBFRRD must be inside the CALL string.) We also note that the IO/CORE interface register IOBFR is being used also as IO/EXTERNAL DEVICE interface. While the compiler itself will never use an interface register between parallel controls for any other purpose, the programmer may choose to do so. The compiler will bring this fact to the attention of the programmer but will not label it as an error.

Core Control: We note that the initial wait string assigns a higher priority to IO than to main control. If a scanner type of arrangement is desired we should replace the first two statements by

```
START    IF (ICRQ = 1) IOD
STRT1    IF (MCRQ = 0) START
```

and return to STRT1 instead of START after RCI and WCI. Clearly one could, by additional IF statements, permit more complicated scanning procedures (such as a 2:1 priority for IO over MAIN). Also we could have saved some writing through the use of IRP.

Library Subroutine Usage: An example of a library subroutine is shown under .RIPLADD. The subroutine belongs to the class .ADD2 and is called by that class name in the subcontrol ARITH. The class definition is contained in a table within the library directory. We note a new statement ASSERT. This statement is an assertion known to the programmer but difficult to detect by program. The assertion is passed on to the logic compiler for its use.

CONCLUSION

The purpose of this paper was to present a proposal for a computer compiler system of programs. As of the time of writing this paper, the system of programs is not available and hence no experimental data can be provided. The example given establishes the following facts:

1. The input language is simple and versatile.
2. The input language is complete. That is, one can describe any existing computer unambiguously in this language.
3. The language is translatable. That is, there appear to be no conceptual reasons why the input cannot be algorithmically translated to produce optimized hardware designs.

The basic concept proposed here is not entirely new. Similar work has been reported earlier by Proctor[2] and Schorr.[4] The present paper differs from the earlier proposals in several respects. First the philosophy is different. The basis of each decision has been user convenience rather than linguistic structure. In fact we have chosen to disregard the linguistic aspects. By the same token, the user is not required to specify any more information than he absolutely has to. For example, no registers that are not directly referred to in the input need be

defined. MACRO, SUBROUTINE, LIBRARY and conditional compilations are new features (an elementary MACRO was used by Schlaeppi[3]). The use of global headers and the notion of parallel/sequential CALLs to subcontrols introduces a flexibility that was not previously available.

The compiler (when completed) will optimize more extensively than is humanly possible, maintaining the modularity specified in the input. Thus the system compiler will attempt to merge microinstruction strings both between instructions (of the object machine) and within an instruction, inserting conditional branch statements where necessary. The hardware compiler will attempt as much Boolean minimization as practical. The assignment of bit configurations to instructions and the combination of different decoders (within one subcontrol) are places where substantial gains are expected.

One last distinction which is conceptually trivial, but to us important, is that our designs will be asynchronous. It is our claim that asynchronous computers are faster and more reliable in addition to being more maintainable. Design difficulty, which has in the past been the main disadvantage, is eliminated by the computer compiler.

REFERENCES

1. M. S. Zucker, "LOCS: An EDP Machine Logic and Control Simulator," *IEEE Trans. on Electronic Computers*, vol. EC-14, pp. 403–416 (June 1965).

2. R. M. Proctor, "A Logic Design Translator Experiment Demonstrating Relationships of Language to Systems and Logic Design," ibid, vol. EC-13, pp. 422–430 (Aug. 1964).

3. H. P. Schlaeppi, "A Formal Language for Describing Machine Logic, Timing and Sequencing (LOTIS)," ibid, pp. 439–448 (Aug. 1964).

4. H. Schorr, "Computer Aided Digital System Design and Analysis Using a Register Transfer Language," ibid, pp. 730–737 (Dec. 1964).

A BUSINESS-ORIENTED TIME-SHARING SYSTEM

G. F. Duffy and W. D. Timberlake
International Business Machines Corporation
Systems Development Division, Poughkeepsie, New York

INTRODUCTION

The purpose of this project was twofold. First, to gain systems and operating experience with a remote terminal, time-sharing system. This would help to define the needs of future time-sharing applications. Second, to achieve productive use of time-sharing in some of IBM's current operations.

In general, man attempts to solve complex problems in a step-by-step manner, basing his next step on the results of the preceding step. Problem solvers have continually sought tools to aid them in their tasks. Several decades ago most of these tools (slide rules, calculators, etc.) were under direct control of their users (see Fig. 1). Because turnaround time was not excessive the problem solver could essentially devote his entire energies to the situation at hand in real-time. Unfortunately these tools were not powerful enough.

The computer emerged as an extremely powerful tool; however, it was too fast and expensive to be used efficiently by one individual in a real-time mode. Thus, the jobs had to be processed sequentially by the computer. Turnaround time increased and the problem solvers had to time-share their efforts among several tasks. This tends to be an inefficient use of human talents. This inefficiency, in part, has led to the current interest in time-sharing —whereby the computer serves multiple users simultaneously.

The early development in time-sharing was oriented to the scientific, rather than the business user. This was due, in part, to a lack of knowledge and consequently a lack of interest by potential business users. In spite of this, the Systems and Procedures Department was interested in fostering the use of time-sharing by business users.

In 1962, we learned of the Administrative Terminal System that was being developed by IBM's Advanced Systems Development Division, San Jose, California. Since this system seemed suitable for our use we started making plans to install ATS in the IBM Systems Development Division Laboratory in Poughkeepsie.

Two approaches were possible: select one major application area or select different application areas. Although the former alternative would have been easier to implement the latter course of action was chosen for the following reasons. More experience would be gained. Initial acceptance would tend to be enhanced by a more modest beginning in each area. The system application could be easily expanded at a later time by the addition of more terminals. Finally, failure of a given application would not significantly impact the entire system.

The task of selecting different application areas, and justifying the system was simplified because we were able to obtain three prototype terminals for experimentation and demonstration. The benefit of having something tangible to show to potential users of time-sharing cannot be overemphasized.

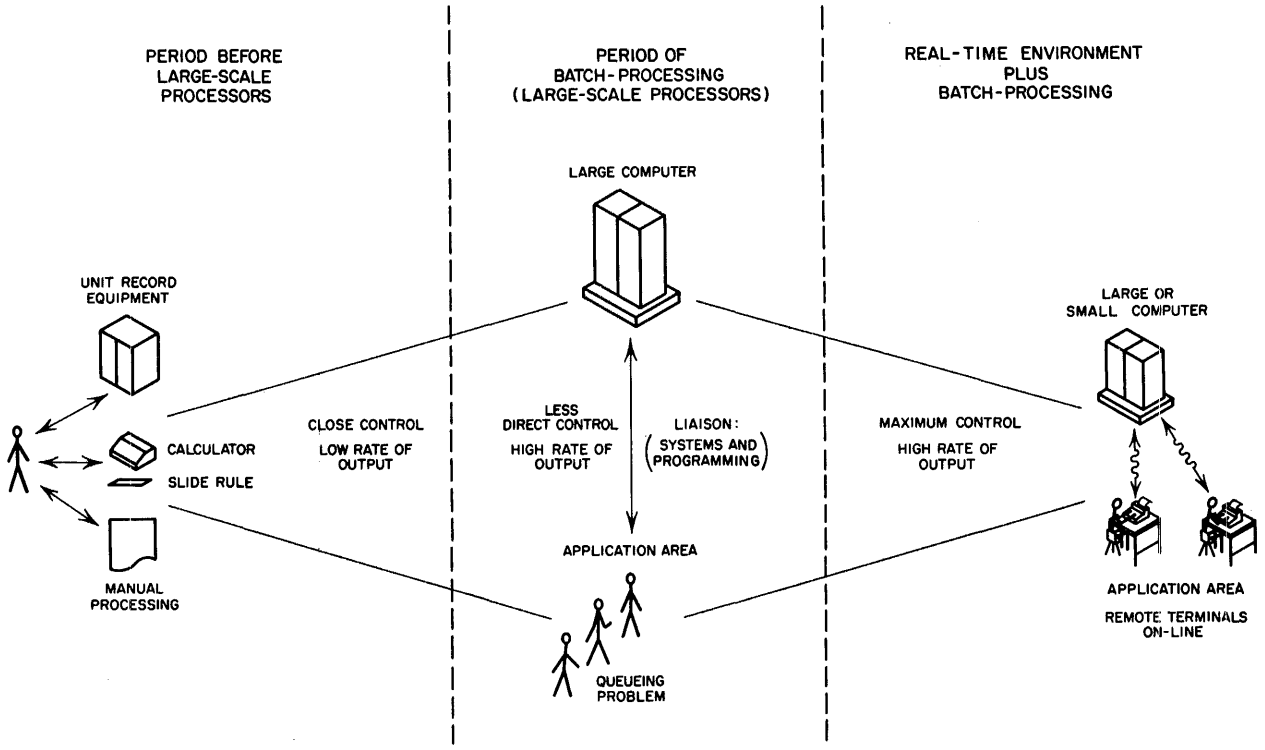


Figure 1. Man-machine relationship.

SYSTEM DESCRIPTION

ATS is a real-time, multiprogrammed, time-shared, remote terminal system that can be run on either a 1440 or 1460 IBM computer (see Fig. 2). The hardware consists of a processing unit, disk storage, multiplexor channels, typewriter terminals, magnetic tape drives, card reader and punch, and on-line printer(s).

The system permits many operators to simultaneously perform different data processing tasks. These tasks include data (or text) entry, immediate correction, storage, retrieval, updating, formatting, and transmission.

At entry time a backspace-correction feature can be used to ensure that error-free data is retained in storage. This is done by backspacing over incorrect keystrokes and re-keying the correct strokes. Data can be updated by using add, replace, and delete functions. Text can be formatted at output time into various specified formats. (This paper was prepared with the help of ATS.) Figure 3 is an example of text as it was entered into ATS, and as it was subsequently formatted and printed.

Data can be transmitted or recorded on various output devices. These include the originating terminal (or any other terminal on the system that provides a message transmission capability), on-line printer, on-line card punch, or magnetic tape. The on-line printer may be standard or supplemented with the upper and lower case print feature. The data recorded on magnetic tape can be used for further processing (off-line in regard to ATS), off-line printing (standard or with upper and lower case characters), or off-line card punching. (See Fig. 4.)

Input media into the system come primarily from the remote terminals, but punched cards or magnetic tape can also be used.

The software consists of a core resident control program and various disk resident service programs. In addition to accepting data from, or transmitting data to, one or all terminals, the control program maintains lists of work in progress, calls the service programs into core as necessary, and handles all disk I/O operations.

Core storage is used for various lists and tables used by the control program and also as core buffer areas for terminals. Core storage allocation consists of 5900 positions for control program and system subroutines, 2700 positions for active service programs, 1400 positions for tables and lists, and up to 6000 buffer positions (see Fig. 5).

As an option, some portion of upper memory

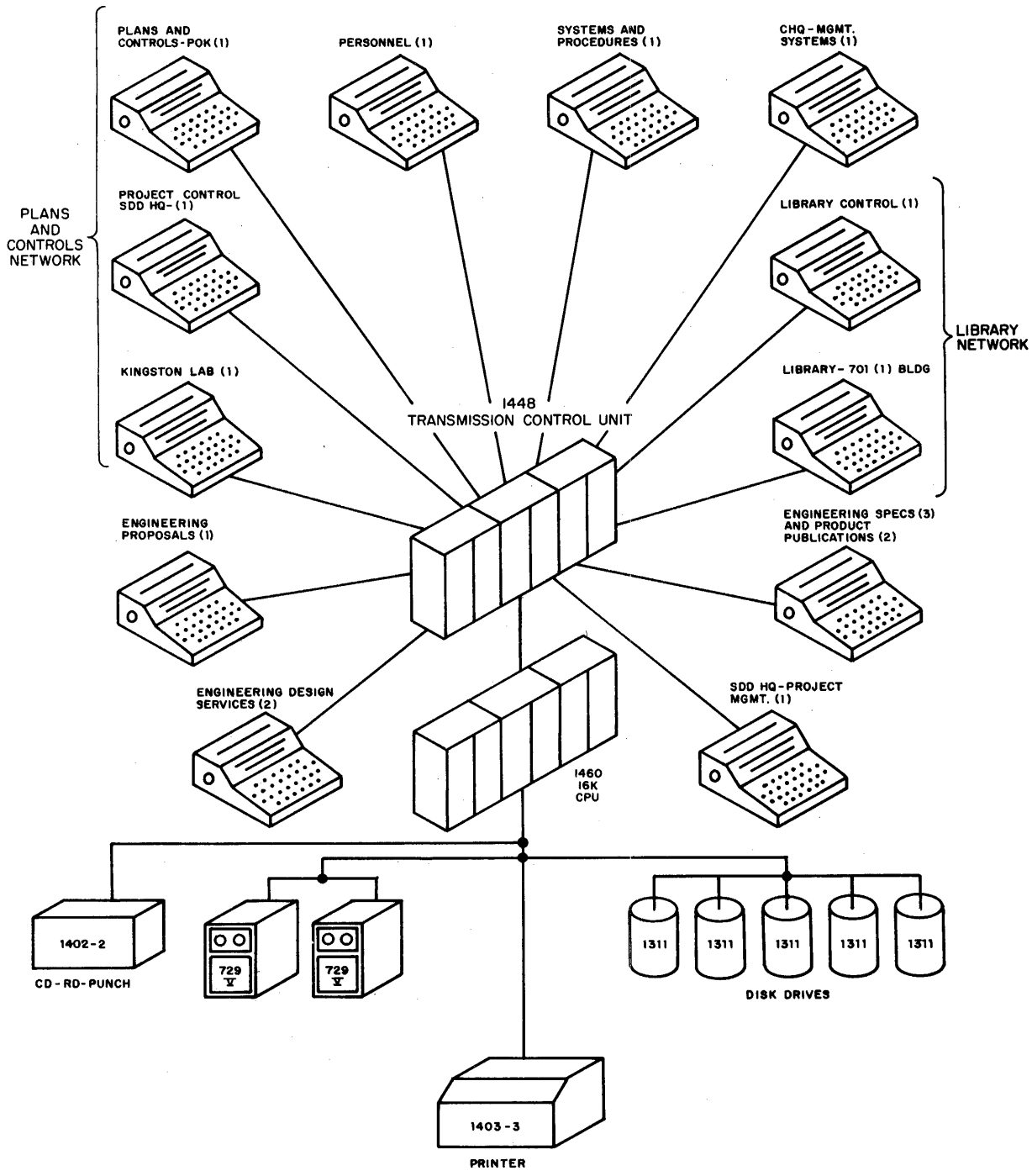


Figure 2. Configuration of the system.

can be used to contain an active peripheral program. Thus ATS can perform an additional task: that of providing a peripheral operation (e.g., tape-to-printer or card-to-tape) concurrent with terminal activity.

Data flows from a terminal into core blocks (100 characters long) that are dynamically assigned at the time of need. When the core block is filled it is written onto disk "working storage" and the core block is released for further assignment. Working storage is defined as that portion of disk containing documents currently being prepared or updated. A

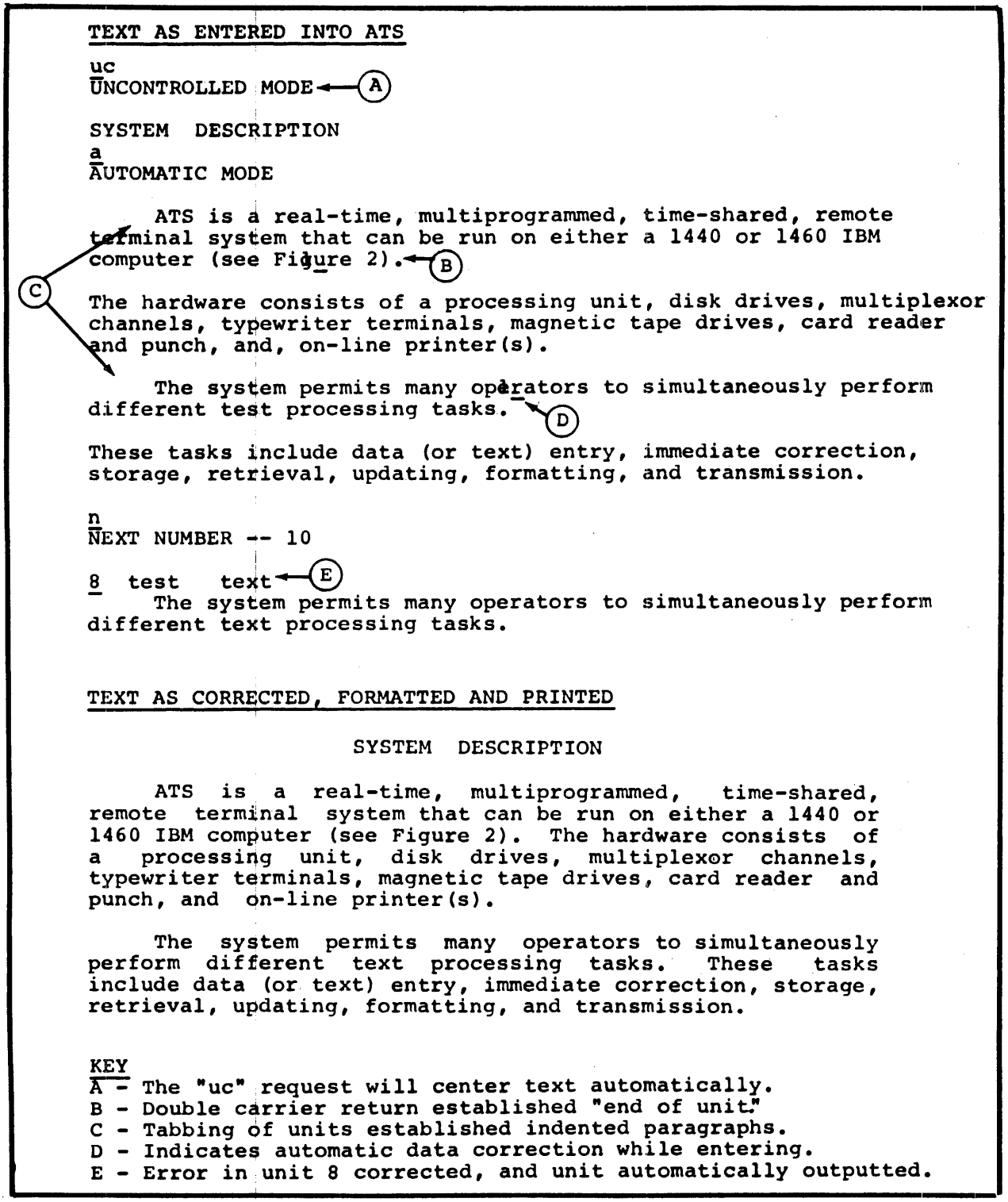

TEXT AS ENTERED INTO ATS
uc
UNCONTROLLED MODE
A
SYSTEM DESCRIPTION
a
AUTOMATIC MODE
ATS is a real-time, multiprogrammed, time-shared, remote terminal system that can be run on either a 1440 or 1460 IBM computer (see Figure 2).
B
C
The hardware consists of a processing unit, disk drives, multiplexor channels, typewriter terminals, magnetic tape drives, card reader and punch, and, on-line printer(s).
The system permits many operators to simultaneously perform different test processing tasks.
D
These tasks include data (or text) entry, immediate correction, storage, retrieval, updating, formatting, and transmission.
n
NEXT NUMBER -- 10
8 test text
E
The system permits many operators to simultaneously perform different text processing tasks.
TEXT AS CORRECTED, FORMATTED AND PRINTED
SYSTEM DESCRIPTION
ATS is a real-time, multiprogrammed, time-shared, remote terminal system that can be run on either a 1440 or 1460 IBM computer (see Figure 2). The hardware consists of a processing unit, disk drives, multiplexor channels, typewriter terminals, magnetic tape drives, card reader and punch, and on-line printer(s).
The system permits many operators to simultaneously perform different text processing tasks. These tasks include data (or text) entry, immediate correction, storage, retrieval, updating, formatting, and transmission.
KEY
A - The "uc" request will center text automatically.
B - Double carrier return established "end of unit"
C - Tabbing of units established indented paragraphs.
D - Indicates automatic data correction while entering.
E - Error in unit 8 corrected, and unit automatically outputted.

Figure 3.

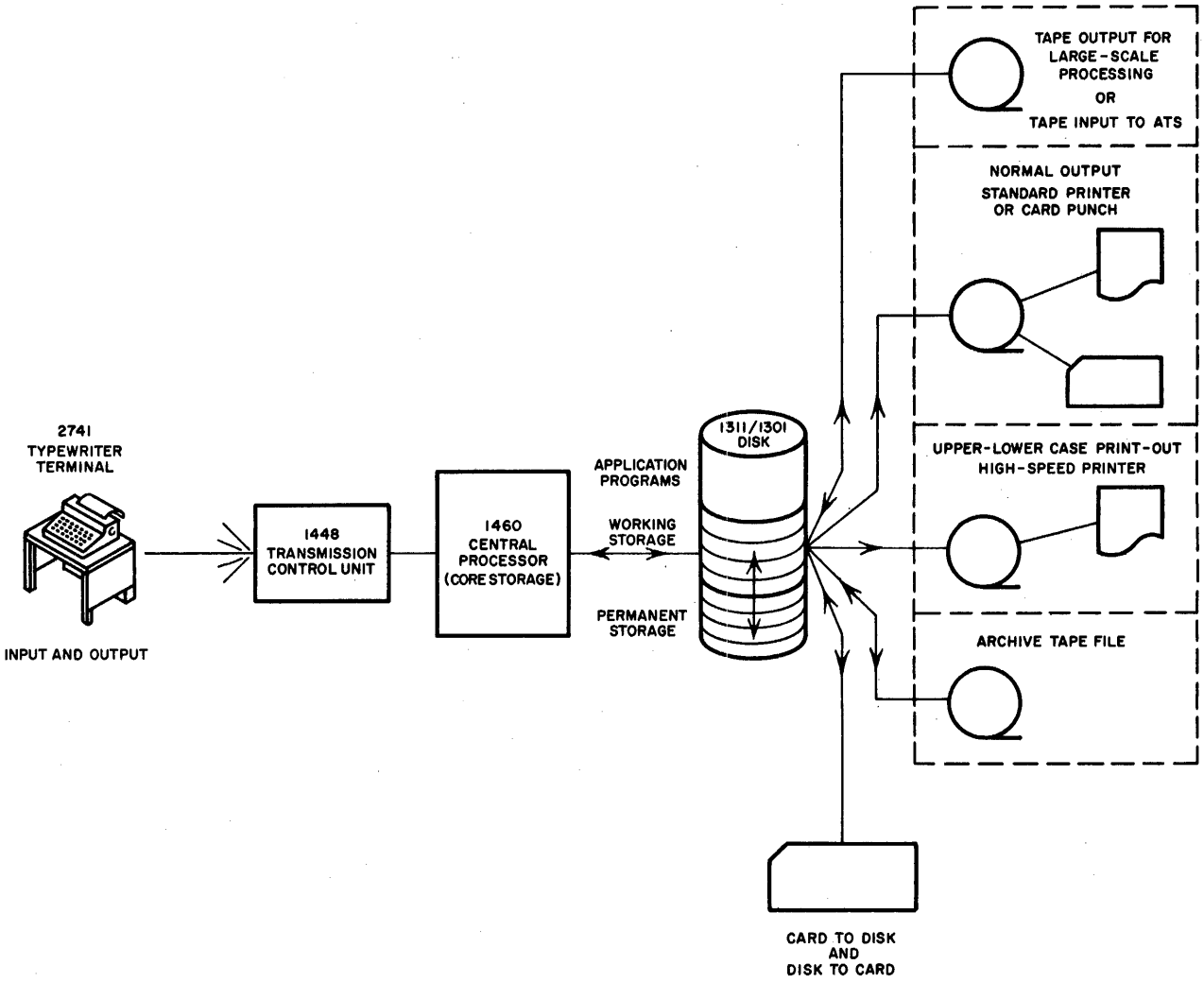


Figure 4. Administrative terminal system—Work-flow schematic.

document in this part of storage is essentially private to a given terminal.

In contrast, a document in "permanent storage" can be copied into private working storage by any terminal operator who has the requisite identifying information. Thus a document can be created in working storage from any terminal, stored in permanent storage from the same terminal, and later copied from permanent storage into the working storage of any terminal.

The operation of the terminal is relatively simple. Many commands are designated by as few as two keystrokes. A list of the common ATS functions follow:

- Clear working storage. This erases working storage.
- Delete stored document.
- Erase line(s) of data from working storage.
- Erase characters in current line of data in working storage.
- Retrieve document from permanent storage.
- Retrieve a previous line of data from working storage.
- Insert a line into working storage.
- Substitute one phrase with another.
- Move line(s) from one portion of working storage to another.
- Print working storage. (This command can take several forms. The printing can be performed on either a printer or a terminal. It can be formatted or unformatted.)

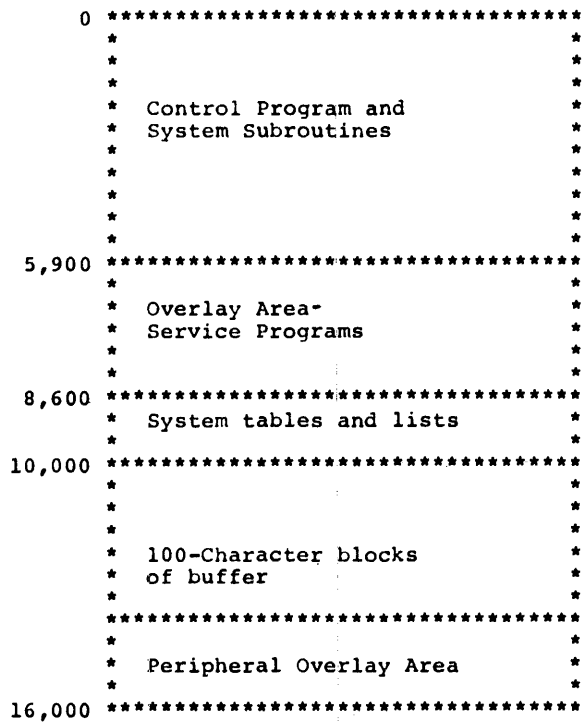


Figure 5. Core storage allocation.

- Store a document.
- Transfer a document to/from magnetic tape or cards.
- Receive a message sent from another terminal.
- Transmit a message to another terminal.

In addition to the common functions, some provisions available for automatic formatting are:

- Page length specification.
- Page width specification.
- Right margin justification.
- Automatic page numbering.
- Line centering.
- Line skipping.
- Heading/footing. (Automatically generated at top/bottom of each page.)
- Page skipping.
- Keep specified portion of text together on a page.

APPROACH TO SYSTEM IMPLEMENTATION

The events leading to actual system implementation include the following.

Original Proposal

As a result of "hands-on experience" gained from the initial 3 terminals, which were demonstrated to more than 25 different areas, we proposed installation of 17 terminals within 11 application areas.

In order to minimize cost, ATS was to share a 1460 computer, which was originally intended for peripheral printing only. Consequently, the initial proposal estimated a cost and time savings that would more than cover the cost of the purchased equipment.

In retrospect, the decision to install ATS in many areas, rather than one large one, appears to have been sound. Each application presented new problems, new benefits, and ideas for future system experimentation.

The proposal was accepted by management and the equipment order was placed. See overall ATS schedule (Fig. 6).

Planning and Training

As shown on the ATS schedule (Fig. 6) it was necessary to start in-depth systems analysis approximately four months prior to installation of the 1460 ATS. The three original terminals were tied to a "borrowed" computer and were used as much as possible during this initial analysis.

The objectives derived from the first analysis included:

- Train terminal operators in those areas receiving terminals.
- Define goals for each ATS application.
- Write procedures and design forms for jobs to be done after installation of the system.
- Train computer operators to operate an ATS system.
- Determine storage allocations for each terminal.

After installation of the 17-terminal 1460 Administrative Terminal System, the feasibility phase of the systems analysis began. A target date of three months for concluding feasibility studies was set. Questions to be answered included:

- What type of applications are best suited to ATS?
- Which jobs could ATS best perform with respect to reduced turnaround time, cost savings, and manpower savings?
- What human factors must be considered in operating in a real-time environment?
- What additional procedures are needed to operate a real-time system?
- What is the future potential of ATS in the areas studied?

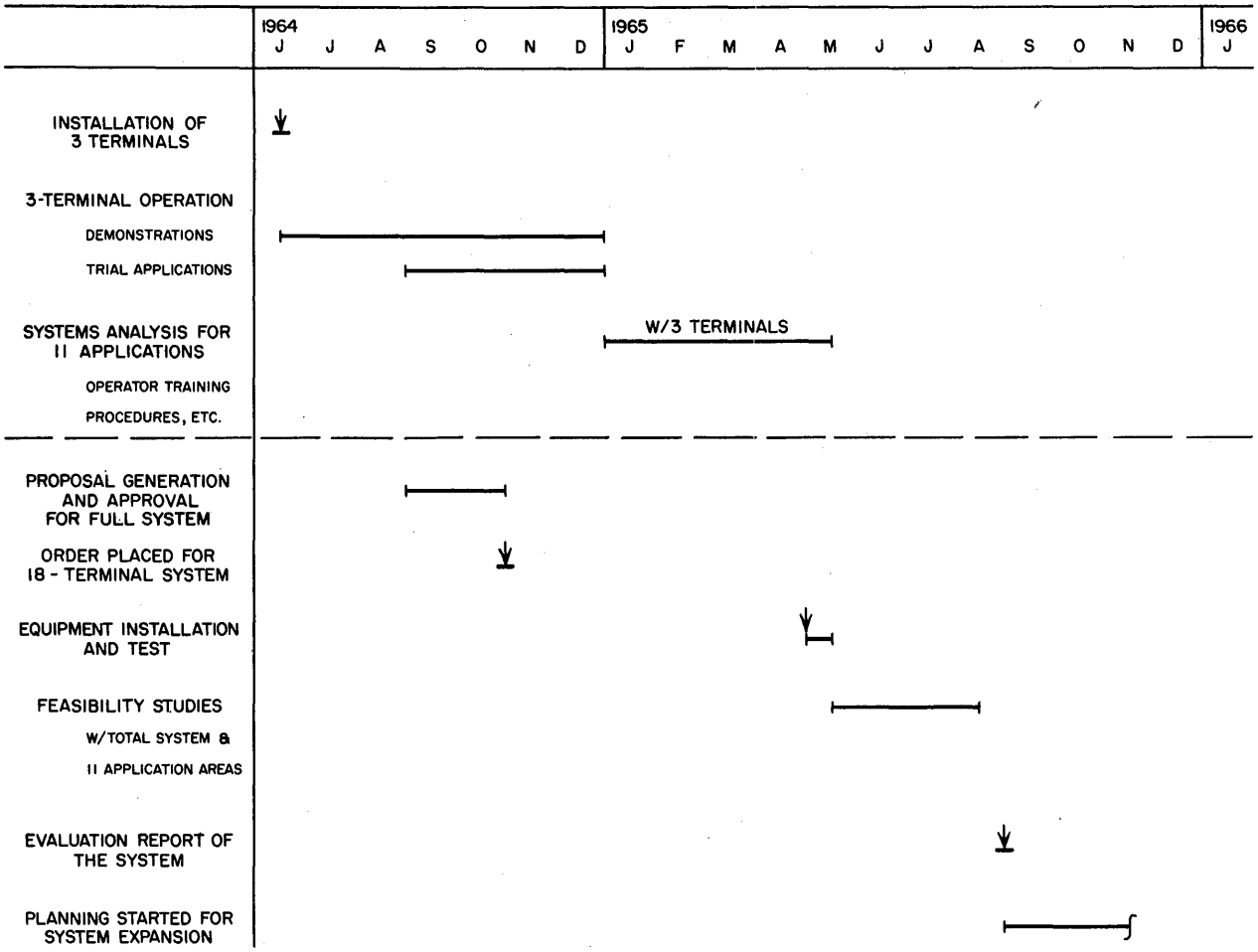


Figure 6. ATS implementation schedule.

- How much off-line effort will be generated from ATS activity?
- Which direction should ATS expansion take relative to system configuration, programming subroutines, hard-copy vs displays, etc.?

After installation of the equipment, the feasibility studies began, and a "Users Conference" was held to review ground rules and operating procedures.

Coordination

An important aspect of system implementation was the type of coordination and control required to operate a remote terminal system. Close communication was essential among the following areas:

Computer Center:	ATS computer operators, and off-line equipment operators
Customer Engineering:	terminal customer engineers, and system customer engineers
Telephone Company:	maintenance of lines between terminals and Computer Center
Application Areas:	17-terminal operators, and operating management in each area
Systems & Procedures:	systems man for each operating area

Computer operators had to become accustomed to dealing with many customers via the terminal and telephone. They also had to learn that operating in a time-shared environment does not permit arbitrary manual intervention, such as pushing the stop button without advance warning. Requests for on-line or off-line processing had to be clearly defined by the terminal operator so that each request could be satisfied. The computer operator also had to act as a clearing agent for all customer engineer

calls. Terminal operators and their managers had to adjust to the condition of being on-line with a computer. The insulation previously provided by the systems analyst and/or programmer was considerably reduced.

A Terminal Operator's Guide was written to define all personal contacts involved, and "what-to-do" in various situations, such as requesting off-line processing of ATS documents, asking for ATS time after prime shift, etc.

Communication breakdowns were frequent during the first months of operation, but by the end of the feasibility phase communications between the various areas was no longer a problem. It was then that the applications people recognized that (from a user viewpoint) real-time systems afford better communications and control than was possible under the batch-processing mode of operation.

TYPICAL APPLICATIONS OF ATS

A summary of the specific areas selected, the tasks performed, and the benefits derived by each user of ATS is discussed in this section.

Engineering Design Services

This area is responsible for the release, and status control, of Engineering Changes (E/C's). In addition to preparing tabular-type data on E/C's, this group prepares Installation Instructions for each release. The use of ATS has decreased turnaround time between receipt of paperwork and drawings from the Engineer, and the release of formal Engineering Changes.

Task Performed

- Weekly status reports (on System/360 E/C's).
- E/C status histories.
- Design automation cycle time reports.
- Systems installation instructions.

Benefits Derived Using ATS

- Reduced turnaround time for review of E/C's, and provided faster release of formal changes.
- Reduced time to update documents and reports.
- Eliminated keypunching, card-handling, and 1401 processing.
- Produced cost and manpower savings.

Potential Uses of ATS

- Maintain development machine structure, and switch to production records after release. (P/N's, Qty, Descrip., where used, etc.)

Engineering Proposals

Requests for price quotes (RPQ) from IBM customers presents two problems. The RPQ workload is high at irregular times, and the requested due dates usually allow the minimum time for preparing a proposal. ATS has allowed the Engineering groups to manipulate stored text into the unique form required for each RPQ proposal. After making minor changes to the standard text, and inserting special information requested by a particular customer, a finished manuscript can be obtained.

Task Performed

- Prepare "Boilerplate" proposal—standard sections stored for easy retrieval and revision.
- Prepare "Unique" proposals—generate unique portions of text, and bring in standard sections as required. The standard portion of each proposal represents about 75% of total text.

Benefits Derived Using ATS

- Reduced typing and editing—only changed portion of standard text requiring editing, or retyping.
- Increased accuracy of proposal data.
- Reduced turnaround time for proposal response to customer, because of the rapid method of updating and editing existing copy.
- Provides camera-ready output.

Library Services

This application involves converting library control processes to a real-time situation. In addition, the feasibility of a library network is being explored for the purpose of centralizing the common library functions of several facilities. Two terminals are now being used for this purpose.

Task Performed

- Book order processing—after data is entered, the book order is printed on the terminal, the total order list is updated, and a status report output is generated.
- Book holdings—master file by shelflist, title and author.

Benefits Derived Using ATS

- Eliminated keypunching, card-handling, etc.
- Permitted rapid access to central files.

Reduced turnaround time on processing orders and answering inquiries.
Produced cost and manpower savings.
Proved feasible for use in larger-scale library networks.

Potential Uses of ATS

Library announcements.
Subscription control (e.g., periodicals).
Library procedures.
Cataloging and subject index.
Book circulation cards.

Plans and Controls and Documentation Controls

The Plans and Controls (P&C) area is responsible for project planning, control processes to measure plans vs performance, and for issuing of status reports.

Because of the shared responsibility for systems design at various IBM laboratories, and the existence of an overall P&C manager at divisional headquarters, the feasibility of an ATS network for common P&C needs is being explored.

The Documentation Control area is responsible for the distribution and status of Engineering Specifications. The status of specifications, i.e., approval or disapproval, is maintained through ATS.

Task Performed

Prepare engineering specifications.
Prepare functional and performance specification status reports.
Perform keypunch simulation (project plan update).
Establish communication network for:
SDD Laboratory—IBM Poughkeepsie
SDD Laboratory—IBM Kingston
SDD HQ—IBM Harrison
Prepare "hot" exception reports (e.g., systems status).
Prepare "hot" manpower status reports.

Benefits Derived Using ATS

Established central files that are easily retrievable in hard-copy form.
Maintained control files on a real-time basis.
Provided rapid communication of critical reports to SDD HQ.
Proved feasibility of larger SDD communication network for facility control functions.
Resulted in a high degree of accuracy.
Minimized turnaround time between final update and transmittal of reports and specifications.
Reduced manpower and cost for the preparation of control documents and specifications.

Divisional and Corporate HQ—Procedures and Planning

Task Performed

Preparation of, and file maintenance on:
Procedures
Procedures Distribution Lists
Organization Directory
Document Index
Card-Image Input for Off-line Processing
General Memoranda

Benefits Derived Using ATS

Provided rapid access to stored data.
Minimized turnaround time to update and output new files.
Elimination of keypunch, card-handling, etc.
Reduced typing and editing.
Increased accuracy.

Potential Uses of ATS

Systems and procedures programming.
Legal department documentation.
Corporate policy revisions.
Contract preparation.

CONCLUSIONS

This section discusses the problems encountered in implementing a time-sharing system, an overall summary of significant benefits experienced by ATS users, most promising applications of ATS, and summary of accomplishments.

Problems Encountered in Implementing ATS

A list of the common problems encountered in implementing ATS follows:

1. Human factors, habit patterns, etc., have to be considered with respect to both operation of the terminal and designing new procedures.
2. It was found that storage requirements exceeded available storage. A larger disk file is planned to increase each terminal's storage allocation to at least one million characters.
3. Systems analysis of terminal applications has to be very flexible, in order to permit changing of existing procedures and formats.
4. Close coordination and control is required, especially during the first several months, between the critical areas involved in system implementation, i.e., Computer Center personnel, customer

engineers, operating managers, terminal operators and systems analysts.

5. The selection and training of terminal operators requires special techniques. Efficient terminal operation for most applications is facilitated by a combination of secretarial skills and a logical thought process.

6. While reductions in turnaround time can be readily defined, a better means is required to define cost and manpower savings. This is because these savings usually overlap service-type departments.

Summary of Significant ATS Benefits

The following is a list of benefits experienced by one or more of the various ATS users:

1. Proved to be a flexible means of entering text, tabular and card-image data into the system, with minimum time required to maintain up-to-date files.
2. Provided real-time access to updated files.
3. Provided the ability to establish common function networks for constructing central files from various sources, and also allowing rapid communication of critical memoranda.
4. Reduced turnaround time between keyboard entry and desired output.
5. Provided a flexible means of obtaining output data, i.e., on-line or off-line.
6. Provided the ability to capture data in machine readable form for large-scale processing.
7. Eliminated transcribing, keypunching, card-handling and card-to-tape operations.
8. Increased accuracy of text or tabular data, since unchanged portions are not subject to retyping errors.
9. Reduced editing time.
10. Cost savings as a result of reducing repetitive operations as well as reducing intermediate steps (e.g., control points, delivery, distribution).

Most Promising Type Applications of ATS

The most successful applications of ATS are those that take advantage of the greatest number of the following ingredients, especially when overlapping the three categories listed:

1. *Need for capabilities such as:*
   a) File maintenance on a "real-time" basis, at frequent intervals.
   b) Information retrieval on a "real-time" basis.
   c) Hard-copy requirement, either on-line or off-line.
   d) Minimum turnaround, entry-to-output.
   e) Card-image preparation for batch processing (keypunch simulation).
2. *Network of common function terminals.*
   a) Access to central files from various facilities.
   b) Rapid hard-copy communication between facilities.
   c) Merging of various facility files for access or processing.
3. *Bonus items.*
   a) Form letters and normal correspondence.
   b) Frequently changing lists, tables, manpower charts, etc.

Summary of Accomplishments

Six months of operating experience proved ATS to be feasible in 8 of the 11 application areas. Some terminals have been reassigned with increased emphasis on proved production applications.

Although ATS can be justified for many "single" functions, it is more easily justified when it can be used for various functions within an area.

As expected, much learning has taken place. Our Computer Center is in a better position to operate a time-sharing system. A number of people have been exposed to ATS and the concept of time-sharing. Others have had considerable terminal operating experience.

Extremely important is the fact that our systems people have increased their knowledge of time-sharing immeasurably. We can now avoid many of the earlier mistakes we made in planning, ordering, installing and operating ATS. For example, we would now expect new terminal operators to be apprehensive when first exposed to the system. But we would also expect the majority of them to become enthusiastic supporters in a short period of time.

We would expect operating management to be somewhat negative when time-sharing is first proposed to them. Generally this position changes to one of expectation that time-sharing can cure all problems. Certainly it cannot do this, but we can anticipate the attitude.

We are now in a better position to specify to de-

signers of future systems what was good about ATS. We can also specify improvements that became apparent to us during this period.

As a forerunner of things to come ATS has served in an outstanding fashion. We are presently planning for the next system which will be superior to ATS partially because of the experience gained from ATS.

In summary, our experience with ATS has solidified our previous conviction that time-sharing systems can, and will in the future, assume a major role within data processing. If current interest (as expressed by word of mouth, publications, and computer manufacturers' plans) are any indication, this major role is fast approaching.

ACKNOWLEDGMENTS

ATS was designed and developed at the Advanced System Development Division, San Jose, California. Mr. Michael Nekora was principally responsible for ATS development and implementation and worked closely with us during the initial installation of the system at the Poughkeepsie Laboratory.

The total systems analysis and programming efforts during the implementation of ATS applications were accomplished through the cooperative efforts of S&P personnel, computer operators, and the operating people involved in the design of each terminal application.

"NEVER-FAIL" AUDIO RESPONSE SYSTEM

Bruce Dale
Honeywell EDP Division
Wellesley Hills, Massachusetts

INTRODUCTION

The Northwestern Bell Traffic Rating System is a real-time audio response dual computer system which must be on-line continuously. The system allows any Northwestern Bell long-distance operator to interrogate a remotely located computer system for a long-distance toll rate. The 3000 operators are widely scattered over an area covering 10% of the country. The toll message response consists of an amount, plus a word denoting whether the rate is for a "station-to-station" or "person-to-person" call. These conditions are ideal for using audio response for the remote display of information, rather than any of the commonly used hardware terminals.

The Traffic Rating System concentrates an operation that was formerly done by widely dispersed Rate Operators. This concentration has a disadvantage in that the operation is now vulnerable to the single failure of any part of the computer system. To keep the operation going continuously, a Systems Monitor is necessary to 1) detect failure of the on-line operation, and 2) transfer the operation to a standby computer system. In this way the system "heals itself" without recourse to human intervention. The necessity for such a capability is becoming widespread as more computer systems are installed which must be on-line all the time.

The building and installation of a real-time system requires special procedures to deal with the unique problems involved. Some of the problems in putting together the Traffic Rating System were:

1. Liaison between the customer and the manufacturer over a distance of half the country.
2. Development of two completely new units for the system—one from a subcontractor who was also a great distance from the manufacturer.
3. The system had to be in operation less than a year after the contract was signed, resulting in the parallel debugging of new hardware along with the software.
4. Once on-line, the system had to operate continuously. It could not be taken off-line for any reason, and so no long shakedown of the system was possible.

These and many other problems had to be solved, through close liaison, careful planning, and tight scheduling. The successful installation of the system was the one purpose of the project. Of potentially greater value was the gaining of experience which will provide the capability to meet the growing demand for this type of system.

PURPOSE AND OPERATION OF THE TRAFFIC RATING SYSTEM

Manual Rate Quotation

When a long-distance call is made from a telephone booth, the operator must determine the ini-

tial deposit—the rate for the first three minutes. To do this, she calls a Rate Operator and gives her the first six digits of each of the two 10-digit telephone numbers involved. The Rate Operator then goes through the following procedure:

1. Using the six digits, she looks up in a table the Vertical and Horizontal Coordinates for the call's origin and terminal points. These are grid coordinates, similar to latitude and longitude.
2. She then calculates the differences between the two Vertical Coordinates, and the difference between the two Horizontal Coordinates. This gives her a measure of the two legs of the right triangle whose hypotenuse connects the two locations.
3. Another table enables her to find the airline distance between the two locations.
4. Finally she enters the appropriate rate table with the distance, and gets the initial rate, which she relays back to the operator handling the call.

The above process occupies two operators, their positions and the connecting circuit for an average of 45 seconds. Times as long as two minutes have been recorded. In an effort to improve the service while at the same time reducing the costs involved, telephone companies are investigating ways of automating the rate quoting process.

Automatic Rate Quotation

The Northwestern Bell Telephone Company serves an area in the Upper Midwest that covers approximately 10% of the United States (Fig. 1). Late in 1964, the company requested bids on a real-time computer system which would:

1. Be accessible to any of the 3,000 operators in the Northwestern Bell territory who handle calls requiring an initial rate quotation;
2. Accept rate inquiry digits from an operator;
3. Calculate the rates, using information previously put in memory, such as Vertical and Horizontal Coordinates, location time zones, rate tables, etc.;
4. Inform the operator of the resultant rate;
5. Operate continuously without fail; and

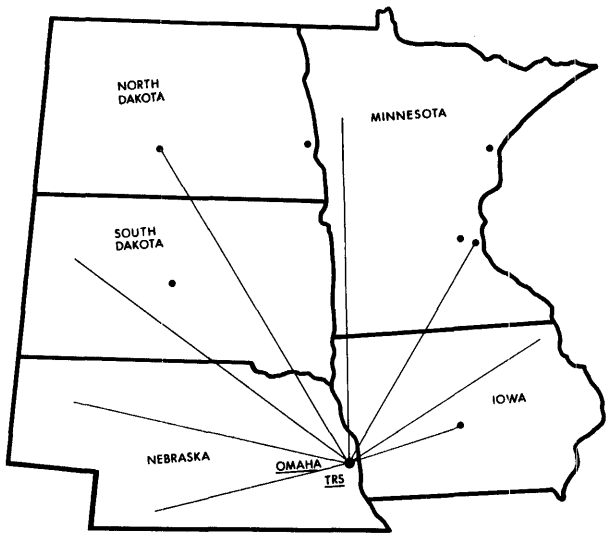


Figure 1. The Northwestern Bell Traffic Rating System in Omaha serves 3000 operators in five states.

6. Handle the peak hour load of 5,000 inquiries over 24 trunks with less than 60% of the total central processor capacity.

System Input

Input to the automatic system would come from keysets which were already an integral feature of the operators' positions. These keysets send digits in a form which are received and decoded by a Multi-Frequency Receiver. The computer system would interface with 24 trunks, each having a Multi-Frequency Receiver. (See Fig. 2).

System Output

The answer to an inquiry would consist of an amount in nickel increments from $0.00 to $3.00 and/or control or information words. In all, the replies could be made up from less than twenty different words.

The possible output devices were cathode-ray tubes, teleprinters, or audio response units. Each device had certain advantages, as shown by Table 1. The slight disadvantages of limited vocabulary and low speed of the Audio Response Unit were more than offset by its negligible cost and minimum maintenance requirements. With audio response, the system responsibility of the computer system manufacturer would be concentrated in one location, and not be spread among 3000 remote sites.

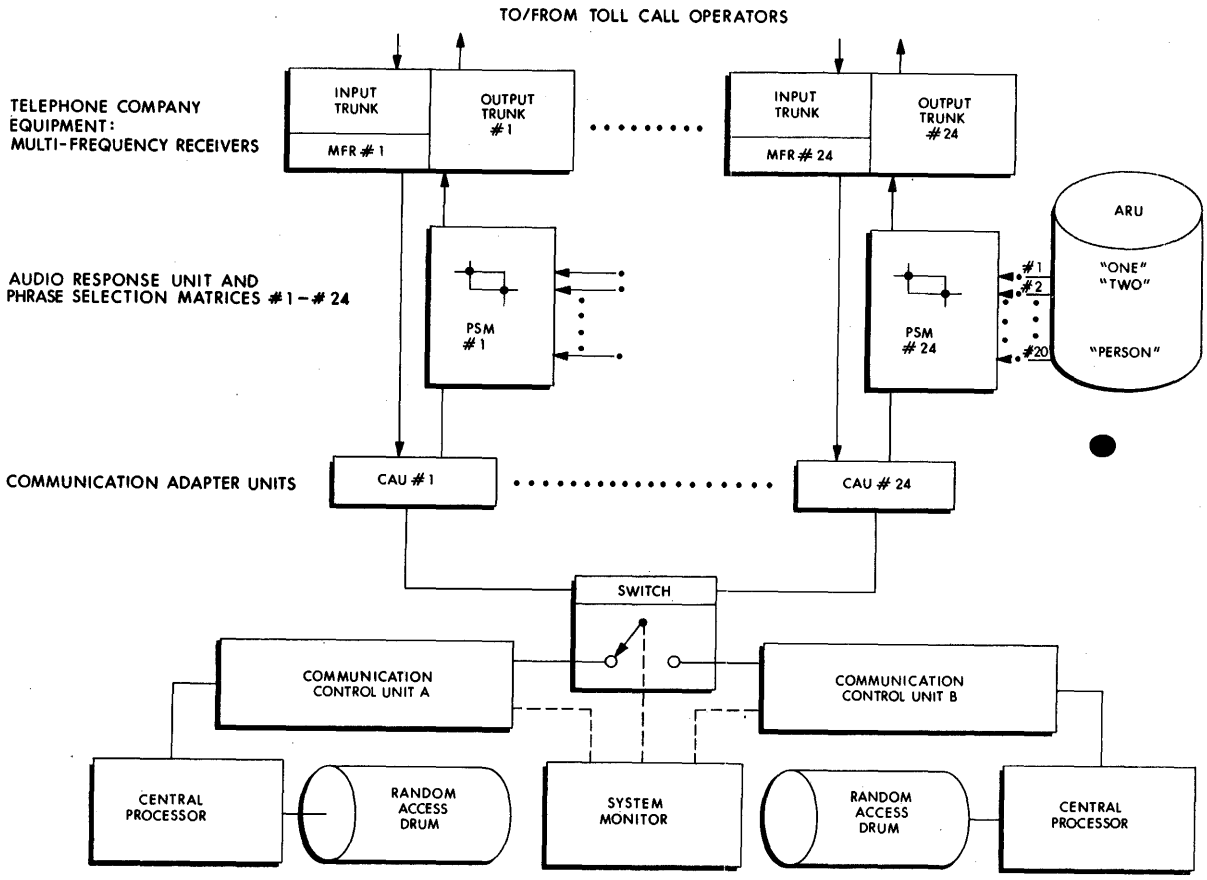


Figure 2. Diagram of Traffic Rating System.

Table 1. Ratings of Terminal Output Devices

Device	Cost	Vocabulary	Speed	Installation	Maintenance	Human Factors
Cathode-Ray Tube	High	Unlimited	High	Difficult	Moderate	Excellent
Teleprinter	Medium	Unlimited	Medium	Difficult	Moderate	Poor
Audio Response Unit	Very low*	Limited	Low	None	None	Good

*Cost is that of central site audio equipment divided by 3000.

Traffic Rating System Operation

In November of 1964, Honeywell proposed a Traffic Rating System to meet the requirements set forth by the Northwestern Bell Telephone Company in their bid request.

To obtain an initial rate from the proposed system, an operator performs the following steps.

1. She keys in a three-digit access code, thus tying her position to one of the 24 trunks in Omaha. The system recognizes that the trunk has been "seized," and returns a "beep-beep" tone from the Audio Response Unit to the operator.

2. Upon hearing the beep-beep tone, she keys in the 12 digits, plus a 13th digit to signify whether the call is station-to-station or person-to-person. The system calculates the rate in less than 0.1 second—a procedure that took a Rate Operator 45 seconds. The Central Processor then sends the audio track addresses for the reply to the trunk's Phrase Selec-

tion Matrix. At intervals of 0.5 seconds, the matrix directs the chosen words to the output trunk, and the reply goes to the operator.

3. The operator hears the reply, such as "eight-five-station-eight-five." The reply *states* the requested rate, *confirms* the rate through repetition in case the operator did not understand it the first time, and *confirms* that the rate is for a station-to-station call. This form of reply reduces the chance of operator error or misunderstanding.

Benefits of the Traffic Rating System

A comparison between the manual quotation procedure and the automated system shows the following advantages of the Traffic Rating System:

1. *Service.* The customer waits 15 seconds or less instead of 45 seconds.
2. *Cost.* The Rate Operators and their positions are available for other tasks. This saving is partially offset by the Traffic Rating System rental, and by the cost of equipment to handle the additional long-distance traffic coming to Omaha.
3. *Accuracy.* The chances for error in determining the rate are greatly diminished.

On the cost basis alone, the system will justify itself many times over. While the other advantages are difficult to assess quantitatively, they are highly significant to an organization which must answer to the general public in its function as a public utility.

The basic advantage of the Traffic Rating System is the concentration of the widespread manual rate quotation operation into an efficient centralized activity. Efficiency implies the reduction of redundancy. A system without redundancy becomes extremely vulnerable to failure. A single mishap to the Traffic Rating System destroys the rate-quoting capability for the entire Northwestern Bell area. Such an event must be guarded against at all cost. The Traffic Rating System must "never fail."

"NEVER-FAIL" SYSTEM REQUIREMENT

The financial justification for installing the Traffic Rating System is the availability of the Rate Operators and their positions for other tasks. This means that no manual backup would be available in case any component of the system malfunctioned. The real-time system must also be a full-time system, on-line 24 hours per day and 7 days per week.

The Traffic Rating System requires the functioning of all four major components: Communications Control Unit, Central Processor, Random Access Drum, and Audio Response Unit. The Phrase Selection Matrices, Communication Adapter Units, and associated telephone company equipment are all modular—one unit for each of the 24 trunks served. A malfunction in a modular unit affects only one trunk, while a malfunction in a major component cripples the entire system. Therefore each of the major components is duplexed. Switches enable any component to be switched into or out of the system almost instantly. This capability protects the system from failing from any single malfunction.

This capability solves one problem, but brings in more problems. Now that the switches are available, these questions arise:

- *When* should a switch be thrown?
- *Which* switch is thrown?
- *Who* (or *what*) throws the switch?

Obviously a switch is thrown when the system comes to an unprogrammed halt. Still, the system could be running when only a portion of the system malfunctioned. Numerous transient errors could occur which, while individually correctable, collectively indicate an incipient malfunction. Therefore the running of the system is not the only criterion which determines when a switch should be thrown.

Which switch is thrown depends on which of the major components malfunctioned. If this is not immediately apparent by a control panel display, then additional time must be spent in either finding the bad unit, or trying each switch in turn to see if the system resumes operation. As an alternative to these procedures, *all* switches may be thrown for any malfunction, thus replacing all major units in an effort to get the system back on the air as quickly as possible. Should another malfunction occur before the first one is remedied, then which units are malfunctioning must be determined.

The Traffic Rating System operates in an unattended environment, since none of the usual data processing personnel are necessary for its operation. Other personnel are located too far from the system to be of timely assistance. Therefore, the throwing of the switch must be done automatically, acting upon information as to the status of the major units of the system.

Neither of the Central Processors are used to perform the status evaluation and activate the switch, since they are part of the system which could fail. An independent judgment is necessary to determine

whether a system has failed. This requirement is fulfilled by the System Monitor, which also has the capability of activating the switch to bring in *all* the standby components.

Design of the System Monitor

The primary purpose of the System Monitor (Fig. 3) is to detect malfunctions. The degree of monitoring could be as simple as the detection of a system halt, to the complex sensing of all circuits of a unit (as on the engine of a Saturn rocket). The more complex a System Monitor becomes, the greater the probability that: 1) the System Monitor interferes with the efficiency of the system that it monitors; and 2) the System Monitor itself malfunctions. Considering also the high cost of increasing monitor complexity, the development concentrated on the simpler design concepts.

The design was also affected by the purpose and operation of the Traffic Rating System. These were considered in finding answers to the following questions:

- Does all information in the system have to be saved or transferred from the on-line Central Processor when it is switched to off-line?
- How are malfunctions detected which do not halt the system?
- What is the allowable interval between the occurrence of the malfunction, and detection? Does the detection have to be instantaneous?

When a system requires that all information that has entered the system be saved, the usual procedure is to have two computers receive and operate on all

Figure 3. The System Monitor.

incoming data. An example of this operation is the New York Racing Association Tote System, where both Honeywell computers receive real-time information on every bet made at the Aqueduct, Belmont, and Saratoga Tracks. However, in the Traffic Rating System each inquiry is independent and complete—it has no connection with any other inquiry before or after it, and the receipt of the reply completes all processing for that inquiry. If an operator does not get a reply within seconds after keying her input, or if she does not understand the reply for any reason, all she has to do is disconnect, and re-initiate the inquiry. Since only one computer is necessary for input to the Traffic Rating System, the back-up computer could be used for regular data processing.

Malfunctions which do not halt the system could be found while running dummy inquiries through the system, testing all possible program loops, and ascertaining the correctness of the replies. This check should not be run too frequently, since it might degrade the system's capability to handle live inquiries. Transient errors sometimes indicate that the system is close to a marginal condition, and should be "peaked up." A cumulative record of such events as correctable drum read errors will provide for the detection of incipient malfunctions.

The interval between the occurrence and detection of a malfunction was balanced against the time required to execute the switching procedure. This time was variable over a small range, since it depended upon the state of the system which was to go on-line. Since instantaneous detection of a malfunction was not required, a query-response scheme rather than a continuous signal method was advocated. This required the system being monitored to send a periodic response in reply to an outside query, rather than passively send a continuous signal.

System Monitor Operation

The System Monitor is designed to perform the following functions:

1. It sends a Check character to both Central Processors at regular time intervals.
2. It expects to receive an OK character in return, before a new Check character is due to be sent.
3. If the System Monitor does not receive the OK character from a system in time, it sounds the visual and audio alarms. It may initiate the switching procedure, depending on whether the on-line or back-up system has the malfunction.
4. If the back-up system is to be brought on-line, the System Monitor sends a message to that system, so that it may be prepared to accept the trunks when they are switched. After a short time interval, it switches the trunks.

A Monitor program in each of the Central Processors works with the System Monitor. This program returns the OK character within the alloted time span, unless it is blocked from doing so by a system malfunction or through its own intent. Optional variations to improve the quick detection and remedy of malfunctions are listed below:

1. The program does not return the OK character immediately, but holds it up until nearly the end of the interval. Should a malfunction occur during this time, the System Monitor detects the nonreturn of *this* OK character rather than the *next* one. With this variation, the detection time is 50% of what it would be otherwise.
2. The Monitor program runs a dummy inquiry through the system, testing all the major components except the Audio Response Unit. If the result differs from that previously computed, the program blocks the return of the OK character. The frequency of the dummy inquiry check is either a function of the current traffic load or a constant. In either case, care is taken to make sure that the system capability is not affected by running the dummy inquiry too frequently.
3. The Monitor program ascertains that the System Monitor is sending the Check characters periodically. This closes the loop to make sure that the System Monitor is functioning correctly. If a Check character is not received, the Monitor program displays this information on the teleprinter attached to the system.
4. When an OK character is deliberately blocked from being sent to the System Monitor, the Monitor program displays the reason for its action on the teleprinter. This enables the malfunction to be more readily identified.
5. The program causes a time signal to be printed on the teleprinter periodically. This evidence of operation is reassurance that the system has not somehow malfunctioned without giving any alarm. Should a malfunction occur, the printed record gives the most recent time at which the system can be presumed to have been operating correctly.

The Monitor program in each Central Processor thus provides a great amount of flexibility in the detection and identification of malfunctions. This flexibility would be difficult and expensive to achieve with additional System Monitor hardware.

Secondary System Capabilities

The primary back-up capabilities are handled by the System Monitor when it switches all off-line major components to the on-line system in case of any malfunction. During the period while the malfunction is being repaired, a second malfunction could occur in the (new) on-line system. There are manual switches which allow a functioning Traffic Rating System to be assembled, assuming that the malfunctioning components can be identified and the two components are not identical. These switches are located on the System Monitor Control Panel. The Panel also displays the current status of all components—which system it is that controls each component.

Audio Response Unit Monitor and Switch

The output on each of the 20 tracks of the two Audio Response Units is continuously monitored. If a track output falls below a given threshold, audible and visual alarms on the System Monitor are activated. If the malfunctioning unit is on-line, the back-up unit is automatically cross-switched with it.

Power

The one element that affects every portion of the Traffic Rating System is the power supply. The threat of power failure is eliminated through the following steps:

1. A motor-generator unit is installed to assure uninterrupted power to the Traffic Rating System in case of outside power failure.
2. The power supplies within the system are either duplexed and automatically switched, or else they can be switched manually. That portion of the system which must always be available is fed by two pairs of automatically-switched power supplies (one pair for the Audio Response Units and the Phrase Selection Matrices, and the other pair for the remainder of the modular units, plus the System Monitor and the switches). A regular power supply is normally connected to each of the two groups of major components, each group consisting of a Communications Control Unit, a Central Processor, and a Random Access Drum.
3. A power switch allows any identical pair of major components to be cross-switched between the two regular power supplies. With this switch, two nonidentical malfunctioning major units from different groups can be fed from one power supply, while the other supply handles the functioning system. A power supply is handled as a major component, being able to be switched back and forth between systems as the need indicates.
4. Each cabinet in the system, and each of the drawers in the cabinet is provided with interlocks to enable them to be isolated from the rest of the system when undergoing repairs.

Other modifications are made to break up the system so that the effect of any malfunction is isolated, and not propagated throughout the power supply network.

Future Applications

The computer state-of-the-art is heading toward a level of sophistication in which large electronic systems will monitor their own operation, detecting malfunctioning components and replacing them without human intervention. The System Monitor and duplexed components of the Traffic Rating System makes it one of the systems which is leading the way toward this goal.

IMPLEMENTATION OF THE TRAFFIC RATING SYSTEM

Planning

Immediately after Honeywell signed the contract for the Traffic Rating System, an organization was created to handle the project. An Engineering Project Director was appointed to coordinate all the activities relating to the subcontracting, building, and testing of the complete system. Coordination between the customer and the Engineering Project Director was the responsibility of the Project Manager, who was appointed from the marketing home office. The Project Manager was the center of a network of communication lines. It was his job to collect and interpret the needs of the customer, examine the effects of different system approaches, expedite the paperwork, issue progress reports to interested parties, and in general be informed on all aspects of the project.

Scheduling

The initial phase of the project included the setting up of a master schedule, which would end at the

cut-over date. Progress toward determining that date was made from three directions simultaneously —merging the requirements of the Northwestern Bell management with the capability of their programming staff and of Honeywell's Engineering Department. The software was PERT-charted, the hardware development was scheduled in detail, and a cut-over date was agreed upon. This date was less than a year after the contract for the Traffic Rating System was signed.

Design and Test of New Equipment

When the project started, the Audio Response Unit and the System Monitor had been functionally specified. The responsibility for any additional design work necessary for the Audio Response Unit was given to the subcontractor. The System Monitor was designed and built by Honeywell.

When each equipment prototype was ready, a series of component tests were run on the individual units. Only when these were passed was the unit connected to other components for system test.

The recording for the Audio Response Unit presented a unique problem. There was no quantitative way to determine whether the quality of the voice recording was satisfactory. Can a high voice be understood better than a low voice? Should the voice be that of a telephone operator, or that of a professional voice specialist? Should the phrases be spoken in a flat voice or with a rising or falling inflection? Even spectral contour plots were used to determine the solution to such dilemmas as these.

The System Monitor also had its share of problems. A complex program was written to test the effect of all possible inputs to the System Monitor. Timing intervals were measured, changed, and measured again. Concurrent with this test, the Monitor program was being written and debugged. Through close cooperation among the two programmers and the engineers, the final checkout of the System Monitor hardware and software was done in parallel.

System Test

The complete traffic Rating System was assembled and tested at the Honeywell plant. The system was arranged in exactly the same way that it would be installed at Northwestern Bell. Twelve of the 24 trunks were installed to test the interface of the system, and to allow live multiple input and output. By setting up the complete system at the Honeywell plant, there was little chance of an oversight occuring at a place and time in the future which could be more difficult to handle by Honeywell engineers.

As quickly as possible, one of the two systems was made available to the Northwestern Bell programming staff, who used it to debug their programs. They had planned their program writing so as to parallel the engineering schedule—working from the "inside out" by first doing that portion of the inquiry program involving only the central processor, and ending with the complex control program which handles a variety of inputs and outputs to the central processor. Thus, through detailed program planning and some extra effort at a time when it could be spared, the final operating program was completely debugged on the entire system only a week after the system itself had been debugged.

Acceptance

A total of 12 Multi-Frequency Receivers were connected to the system. Each receiver was connected to a keyset, and the 12 keysets became 12 telephone operator positions. (Actually, each Multi-Frequency Receiver could simulate two trunks to the computer—so the complete environment could be simulated.)

After the system was turned over to the Northwestern Bell programming staff for their final debugging, an extensive series of tests commenced for checking both hardware and software. These consisted of 2 million rate computations that had already been calculated as part of a regular batch process run on a computer. This was followed by a series of inquiries which were put in simultaneously on the 12 keysets, with the results being checked against precalculated answers. When both tests were completed to the satisfaction of NWB management, the system was shipped and reassembled on site in Omaha, where the same tests were run through the system again.

SUMMARY

Despite the difficulties imposed by distance, time, and new equipment, the Northwestern Bell Traffic Rating System began operating on schedule on November 15, 1965. It has been in continuous operation since that time, 24 hours a day, 7 days a week.

APPLICATION OF COMPUTER-BASED RETRIEVAL CONCEPTS TO A MARKETING INFORMATION DISSEMINATION SYSTEM

James J. Gatto
Westinghouse Electric Corporation
East Pittsburgh, Pennsylvania

INTRODUCTION

It is a practical philosophy of many industrial advertisers to direct promotional material and product information to customers and prospects on a selective basis. This selectivity entails direction of mail to recipients so that each receives literature in accord with his established needs and interests. Selectivity affords a considerable reduction in material, addressing and postage costs, and insures that the right information is sent to the right person at the right time. In addition to enhancing the personal touch in the process, the likelihood is increased that the recipient will study and retain every mailing sent to him because he recognizes that these are about subjects in which he is likely to have an active interest.

The size, complexity, and number of mailing lists are rising in proportion to the population growth, market changes and new product development. The economics of keeping abreast of this rise, while validating current lists and accommodating changes, demand a direct mail program in which one of its essential features is the maintenance of accurate and complete mailing lists at minimum cost and maximum efficiency. Industrial advertisers are becoming acutely aware of the need for having at their disposal a large set of combinations of criteria for selecting the audience to which they wish to communicate. This essential feature must be provided by a direct mail program having inherent simplicity and very high speed. In addition, a well-designed mailing program can relieve the industrial advertiser from demanding deadlines and tight scheduling, and provide him with a facility for consolidating large, independent mailing activities.

For many industrial advertisers, a mechanical maintenance and search mailing program does not have the capacity nor capability to handle large mailing lists efficiently or economically. The situation has prompted these organizations to consider electronic data processing and printing as a solution to their direct mail problem.

A computerized marketing communications direct mail system, whose shape and form is based upon a current application of information retrieval techniques, has been developed to satisfy the information dissemination requirements of large industrial advertisers.

MARKETING INFORMATION DISSEMINATION SYSTEM REQUIREMENTS

The essential phases of current direct mail programs are the collection and processing of input data, updating and searching of the data file, and labeling or imprinting, gathering and distribution of publication material. It is helpful to view this mailing cycle as a closed-loop system.

The mailing cycle is initiated by the field salesman in response to a customer's need and interest for information regarding one or more of the com-

pany's products, services and policies. The field salesman prescribes the input data and enters an order form for a new customer, or he enters a change notice for a customer presently being serviced, but who requires a change in status. The forms are forwarded to a control center where the input data are translated and entered on a visual, mechanical or electronic data file. The file is interrogated when a mailing is desired, and the resultant answer set is usually a list or lists of mailing labels. Each label is affixed to or imprinted on an envelope which in turn is manually or mechanically packed with the specified literature. The envelopes are weighed, proper postage is applied, and distribution is made through regular mailing channels.

In reviewing the mailing cycle, certain basic considerations become evident:

1. The format design of the order and change notice forms requires careful attention to details. The salesman's participation in completing the forms must be minimized. Any instructions on the forms must be precise, since it is not possible to contact every salesman personally to explain their functions. In addition to being simple and economical, the forms must easily accommodate new data entries.

2. A control center must serve as the coordinating and control body of the mailing program. The personnel should be familiar with all phases of the mailing cycle. The center's functions should include the translation of input data, preparation of search specifications, scheduling of production runs, validation of errors and changes, and administration of the labeling or imprinting, gathering and distribution processes.

3. The update procedure must be economical, simple, and must provide a means of validating the update entries to maintain accurate lists.

4. A large set of combinations of criteria for selecting the customers to which a particular marketing effort is to be directed must be provided. The search specifications must be in natural language notation to permit anyone with minimal training to produce a finished set of query statements.

5. The data file must be structured to easily accommodate additional mailing lists.

6. Mailing lists must be sorted by zip code to comply with postal regulations.

7. The answer set is usually a list or lists of mailing labels. However, the facility must be provided in which the answer set may be counts, partial lists, and lists which reveal all the data on each customer in the data file.

The above considerations suggest the use of an electronic computer as the catalyst of a direct mail program. In fact, the value of a computerized system in today's complex mail scene is that it substantially improves the speed, economy, flexibility, generality and capability of a direct mail program in comparison with visual or mechanical systems. A computer-based mailing program can promote new concepts of marketing possibilities. The idea here is that a properly designed computer package can provide the industrial advertiser with the ability to communicate with all intermediate points between the company and its audience, and internally within the company. In this light, the direct mail program now becomes a marketing information dissemination system.

A marketing information dissemination system is characterized by the present and future mailing requirements, computer retrieval logic, the company's policies and practices, and the amount of capital investment. Industrial advertisers demand that a marketing information dissemination system satisfy three basic requirements which are input compatibility, file generality and output versatility. These are described as follows:

1. Input Compatibility. Provision of a communication channel for use by anyone in the organization with a legitimate need to reach employees, distribution outlets, customers, and prospects or potential customers. To meet this requirement, the system must provide:

a) An input data recording scheme which is flexible, logical, economical, convenient and easy to administer.
b) Addition, cancellation and maintenance of names and descriptions of recipients from a variety of sources.
c) A coordinating and control group whose responsibility is to process input data, prepare search specifications, schedule and conduct query and maintenance computer runs, and validate errors and changes.
d) Classification and coding schemes for the recipient's industrial and functional roles, and for the organization's products, discount and commission schedules, renewal parts, and special publications, and so on. It is essential that

some accommodation of the existing schemes be made with minimal requirements for technical reevaluation.

2. File Generality. Automation to the extent permitted by technology and economics. The system must provide:

a) Complete and readily accessible mailing lists for employees, distribution outlets and customers.
b) Expansion capability for accommodation of marketing information dissemination requirements for other locations within the organization.
c) Retrieval of mailing label lists or other answer sets by any combination of criteria as specified.
d) Zip code and other special sorting on all lists.

3. Output Versatility. Preparation of reports to meet the marketing information dissemination requirements for buying data, sales promotion material, renewal parts, discount and commission schedules, and special publications. The system must provide:

a) Variety of computer output displays in prescribed formats to include mailing label lists, lists with gathering instructions, lists displaying selective data, and list counts.
b) Labeling, gathering and distribution facilities.

A MARKETING INFORMATION DISSEMINATION SYSTEM

The general model of a marketing information dissemination system is illustrated in Fig. 1. The model is divided into three phases:

- Phase I is the collection and processing of input data and queries.
- Phase II is the updating and searching of the master file.
- Phase III is the labeling or imprinting, gathering and distribution processes.

Phase I—Collection and Processing

Input Data. One of the most positive ways to assure regular list maintenance is to require some individual to "sponsor," or be accountable for, every name added to a mailing list. In most organizations the sponsor is usually a field salesman; however,

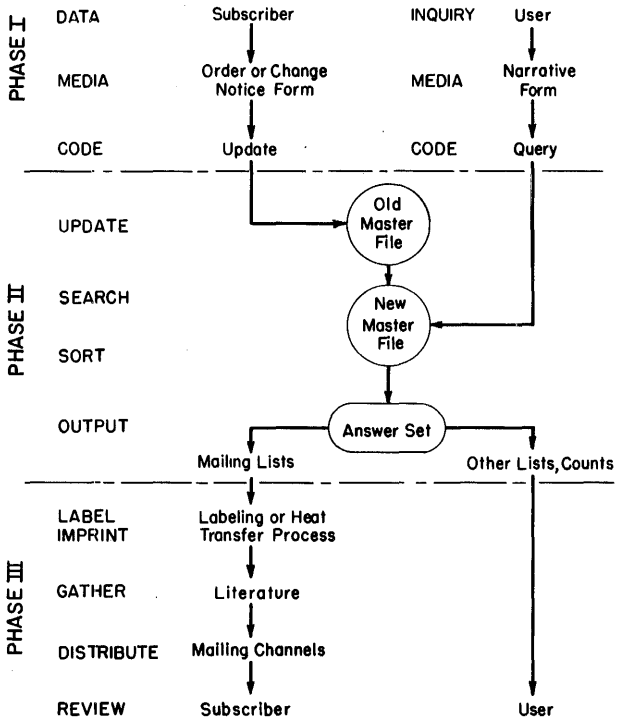


Figure 1. Dissemination system model.

there is a requirement that anyone with a legitimate need should have the ability to enter a name to the mailing list. The sponsor reviews his list of names periodically and submits an order form for any new subscriber who has expressed a need for information regarding one or more of the company's products, services and policies. In addition, he submits a change notice form for every subscriber presently serviced by him, but who requires some change in status. A subscriber, in this context, is a recipient of literature through the mailing program and is, specifically, either an employee, distributor, customer or prospect.

The input data are divided into five categories:

1. Identification
2. Classification
3. Profile
4. Address
5. Key

The embodiment of these categories for a particular subscriber originates a record for that subscriber as illustrated in Fig. 2.[1] Each category is described by one or more terms and each term is composed of an attribute which relates to one member of a class of descriptions, and a value which particularizes the description.

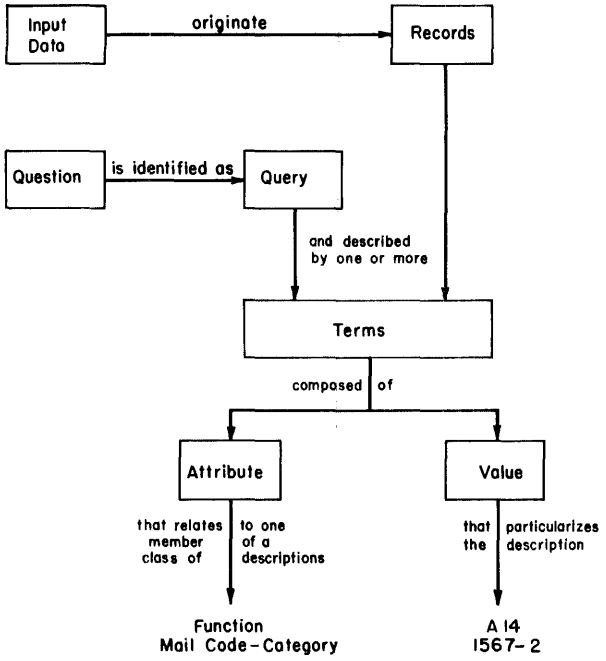


Figure 2. Input data and inquiry structure.

The *identification* consists of a unique numeric term which identifies a particular subscriber's record. This number remains unchanged throughout all subsequent dealings with him, and its coding structure is dependent upon the expected number and natural grouping of records. The *classification* consists of terms such as sponsor, industry, function, and any other which may be included in search specifications. For example, persons of interest may be all customers who belong to the textile industry and who function in the role of vice-president. "Textile Industry" and "Vice-President" are, then, values of the classification. The *profile* reveals the subscriber's publication needs and interests. It is composed of numeric terms selected from existing publication groups to include customer discount schedules, distribution outlet agent commission and discount schedules, handbooks and other special publications, renewal parts data, and product information. Each term in the profile is structured on a mail code-category basis. A mail code is a numeric representation which uniquely identifies a company's product, service or policy, and its category is a numeric representation which particularizes the kind of information such as a price list, descriptive bulletin, dimension sheet, selling policy, application data, or sales promotion piece. The *address* is composed of terms such as firm name, individual's name and title, street address, city, state and zip code. The *key* consists of the identification term and selected classification terms, and is included in the mailing label. One of its primary roles is to link the mailing label with the subscriber's record.

The medium for reflecting new subscriber input data is the mailing list order form, which is a simple, single-part form. This is designed primarily to present a clear and logical format to the sponsor to minimize his participation in its completion. In addition, the form features a nonrigid format in which new data entries can be easily accommodated. The fact that a specific effort is required to enter profile data should encourage the sponsor not to oversupply the recipient with literature. The sponsor submits a change notice form to report a change in status of a subscriber currently being serviced by him. This form differs from the order form in that the identification number is entered on the change notice form by the sponsor whereas this number is preprinted on the order form. The change notice form reflects both the addition and deletion of terms.

A catalog assists the sponsor in entering the proper data values on either the order or change notice forms. The catalog is prepared in a language natural to the sponsor and provides an opportunity for adequate explanation and instructions. Since the catalog is separate from the forms, periodic reissues of the updated version of the catalog can be made without disturbing their format. A typical catalog consists of codes for: customer industry and function, customer discount schedule, distribution outlet classification and function, distribution outlet discount and commission schedule, employee function, products, renewal parts and special publications.

The implementation of a marketing information dissemination system to replace an obsolete visual, mechanical or electronic file system poses a few problems with regard to the initial input data. In most cases, it is desirable to require that the sponsors resubmit new input data rather than convert data on a current file system. The cost of the latter is usually prohibitive and, in addition, the question arises as to whether the data on the current file system are accurate or complete, especially if the company's sales organization has recently been re-structured. The fact that the data resubmission program is imminent invariably disheartens the field salesman since his available time is usually at a premium. However, once he is convinced, and some are never convinced, that the new system will assist him in better serving his present customers and providing

him with new prospects, then the field salesman is more willing to contribute his time and effort. In addition, he now has a golden opportunity to rid his list of nonproductive names. It is imperative that the sponsor be required to submit his order forms within a specified time interval to minimize data obsolescence. Futhermore, once the sponsor is informed of the new system, he becomes less attentive to the old system's maintenance program, and the timing aspect becomes even more critical.

A mailing list control center is established to coordinate and control the new system's operations. Its role during the initial input data phase is to check the incoming order forms for accuracy and completeness and to batch the forms in groups which are specifically numbered to control them. The forms are keypunched by an outside service since they have the capacity to perform this task in the shortest possible time. As soon as sufficient keypunched data becomes available, a test file is established to assist in the training of the center's personnel in all phases of the new system. This is an important step since, in most cases, their exposure to computer operations has been very limited. The personnel selected to supervise the center's functions should have a natural affinity for "system" concepts.

After the initial data file is established, the new system is run concurrently with the old system until debugging of the new system is completed. At some point in time, the new system is declared operational and the old system is eliminated. The mailing list control center activity now enters Phase I of the marketing information dissemination system. The center's functions in Phase I include the transferring of data from the order and change notice forms to punched cards, preparing and batching queries, and scheduling computer runs.

The logical requirements for file maintenance consist of three update events:

1. Insertion of a new file record
2. Changes in an existing file record
3. Cancellation of an existing file record

A new file record is initiated by receipt of an order form at the mailing list control center. The data is transferred to punched cards in prescribed field locations, and the number of cards is primarily a function of the number of profile terms entered on the form. Each card is assigned a predetermined set of codes. A change to an existing file record is initiated by receipt of a change notice form which reflects new values for those terms to be altered. These values, together with the subscriber's identification number and pertinent codes, are keypunched in the prescribed field locations. It is important that field locations be a natural consequence of the order and change notice format designs.

A cancellation of a record is effected by simply writing the word "cancellation" across the face of the order form. A single card is keypunched with the subscriber's identification number, pertinent codes, and the word "cancellation."

Queries. A question may be posed by anyone who has an interest in obtaining any logical or arithmetic implication of the file content. A question is interpreted as a request to locate and retrieve a reference to or data about a record or collection of records, and is composed of a narrative and a set of statements. The narrative expresses the logic of the query in a language natural to the user or questioner, and the set of statements prescribes the search and report specifications. The intent is to provide a medium whereby an occasional user can reliably express his desires or comprehend what is meant if they were formulated by a professional logician.

The nature of the data and its intended use set the basis for a marketing information dissemination system design, and the retrieval logic must be modified somewhat to obtain the best results within this framework. A reference retrieval package has been developed and is currently being applied to a marketing information dissemination system. The retrieval logic provides an unusually fast and efficient scheme, and features a flexible multilevel query capability. The query notation is designed to allow the expression of the search specifications in such form that minimal training is needed to permit the user to produce a finished set of statements for routine query processing of the master file. The language permits queries of great depth and provides output reports which are flexible and complete. The reports may be in the form of simple counts, list of mailing labels or lists with selected data.

A query postulates the existence of terms or attribute-value combinations, for example, mail code-category (attribute)—1201-1 (value), or sponsor-53468, function-B24, and so on. It also requests that a search be conducted to locate any record which responds to the logic of the query. The language consists of a set of primitives or elemental query statements used to manipulate data. The primitives are:

1. *Search Primitives* examine each record in the master file to determine the relation of the record

values to those values or arguments cited in the search specifications, as:

a) EQUAL—search for equal value.
b) PICK—search for equal value. (This is a special primitive which is explained later.)
c) NOTEQ—search for unequal value.
d) EQGR—search for equal or greater value.
e) EQLS—search for equal or lesser value.
f) GR—search for greater value.
g) LS—search for lesser value.

2. *Result Primitives* move partial answers from level to level and designate final answers, as:

a) TEST—move partial answer to next higher level.
b) CLEAR—erase previous result.
c) ANSWER—designate result for output.

3. *Output Primitives* initiate the desired counts and lists, as:

a) COUNT—accumulate a count of 'hits.'
b) LIST—list respondents of 'hits.'
c) POST—output special information with list respondents.
d) SELECT—select particular terms of 'hits.'
e) DUMP—list all terms for every record.
f) NOTE—explanatory comment ignored by logic system.
g) END—end of search and output logic.

There are six different levels at which query search and result primitives, except PICK, may be entered; 1 is the highest and 6 is the lowest level. Odd numbered levels are designated AND levels and even numbered levels are called OR levels. The levels are provided to permit the development of complex logic which may span the entire level range, or any partition, from the lowest to the highest level designation. The PICK primitive and the output primitives do not require a level designation. A few examples will be considered to illustrate the form of the query narrative and the structure of the query statements.[2]

In the first example, the following query is posed: List all customer mailing labels whose records mention either mail code-category 2181-3, or 1882-1 or 1924-2, and industry 487 and function B38. The search specifications responsive to this narrative are:

PRIMITIVE	LOGIC	LEVEL	ATTRIBUTE	VALUE
EQUAL	OR	2	PROFILE	2181-3
EQUAL	OR	2	PROFILE	1882-1
EQUAL	OR	2	PROFILE	1924-2
TEST		2		
EQUAL	AND	1	INDUSTRY	487
EQUAL	AND	1	FUNCTION	B38
ANSWER		1		
LIST				
END				

The values of 2181-3, 1882-1, 1924-2, 487 and B38 denote the arguments of the search specifications. The TEST routine evaluates the OR results; ANSWER prepares the final results for use as output; and LIST calls for the preparation of reference citations which are, in this case, a list of mailing labels. The remainder of the search specifications are rather self-evident. The key point in the example is that the user has the ability to reach a customer, or any subscriber for that matter, by his industry, function and product interest or any combination thereof.

In actual practice, each specification line consists of additional information which includes a sequence number, query number, record group code and the user's initials. The query narrative and statements are entered on separate forms which are designed for easy translation of data from the form to cards. The narrative and search specifications are printed at the beginning of each separate output listing.

In the above query the resultant answer set is a list of mailing labels. In addition, there is a requirement to output a mailing label and its corresponding profile values which match the set of profile arguments cited in search specifications. The collection of profile values displayed with a mailing label is called a 'pick' list which permits a simplification of the gathering process. The primitive called upon to produce the pick list output format is the PICK primitive, which is similar to the EQUAL primitive, except that it allows a more rapid interrogation of a record profile and provides a more convenient procedure for structuring the output data.

To illustrate the use of the PICK primitive, consider the following query: List all customer mailing labels whose records mention mail code-category 1418-1, 1806-2, 1401-1, and/or 1204-2. The search specifications become

PRIMITIVE	ATTRIBUTE	VALUE
PICK	PROFILE	1418-1
PICK	PROFILE	1806-2
PICK	PROFILE	1401-1
PICK	PROFILE	1204-2
END		

There are many occasions where special publications, which are not listed in the catalog, must appear as members of the pick list for convenient gathering. The POST primitive is especially suited to perform this function. To illustrate its use, consider the following query: List all customer mailing labels which mention industry 311 and function A46 and post cover sheet CS-1. If industry 311 and function A46 and sponsor 21346 are mentioned, then post CS-1 and additional instructions INST-1. The search specifications responsive to the narrative are:

PRIMITIVE	LOGIC	LEVEL	ATTRIBUTE	VALUE
EQUAL	AND	1	INDUSTRY	311
EQUAL	AND	1	FUNCTION	A46
ANSWER		1		
POST				CS-1
EQUAL	AND	1	SPONSOR	21346
ANSWER		1		
POST				INST-1
END				

In this example, either CS-1 or CS-1 and INST-1 would appear as members of the pick list for each mailing label in the answer set.

Even though the internal logic is intricate and detailed as necessary for the exploration of the master file, one can observe from the examples cited that the external language is quite direct and intelligible.

Phase II—Updating and Searching

File Characteristics and Organization. The totality of subscriber records composes the master file. Each record in the file is structured identically which permits the addition of new records to be a matter of routine.

The number of subfiles within the master file is primarily determined by the natural grouping of records. As mentioned earlier, publication material may be directed to anyone of three classes of recipients, these being employees, distribution outlets, and customers or prospects. Hence, the master file is logically organized into a base structure consisting of these three subfiles; however, the capability is provided for the addition of any other subfiles. The establishment of a new subfile or its incorporation into the base structure is primarily an economic consideration and depends upon the subfiles' anticipated frequency of use. Each subfile is identified by a file number and its records are ordered by subscriber serial number. The file number and the serial number form the record identification number.

The number of records in the customer subfile usually far exceeds the number of records in either the employee or the distribution outlet subfiles by an order of magnitude. However, the employee and distribution outlet files are usually interrogated more frequently than the customer file by an order or magnitude. Hence, it is more economical to combine the employee and distribution outlet subfiles on one tape and maintain the customer subfile on another.

Record Format. The subscriber's record format, illustrated in Fig. 3 consists of a matrix record and an address record. The records are maintained separately to provide for the most effective use of storage locations. All terms which are contenders for search specifications are structured in the matrix record, whereas all the terms descriptive of a mailing label are maintained in the address record.

The matrix record consists of the identification, classification and profile descriptions. The primary role of the identification term is to uniquely identify a particular subscriber record. The number is composed of the subfile number, which permits the in-

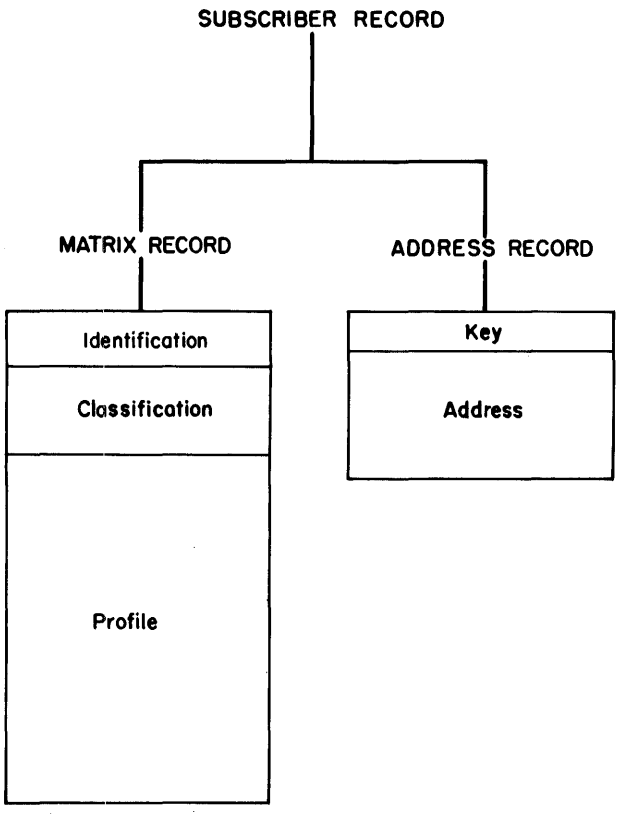


Figure 3. Subscriber record structure.

terrogation of a subfile on a selective basis, and the subscriber's unique serial number. The number appears in the address record where it serves to key the address record to the corresponding matrix record.

The classification consists of terms such as sponsor, function, industry and any other term which may be a contender for search specifications. The organization's requirements for obtaining specified implications of the data file prescribe the kind of terms to be incorporated in the classification. Each term may be single- or multiple-valued and is assigned a particular alphabetic or numeric code for querying. It is essential that existing schemes be accommodated to minimize the organization's requirements for technical reevaluation.

The profile comprises mail code-category terms. The mail code and category values are numeric and their range of values depends upon the organization's product, service and policy classification schemes. Whether the master file is being updated or queried, each profile value being processed is converted by a simple algorithm to a binary representation (L—pattern length) which is slotted into a particular matrix location (Y—matrix location) with a particular pattern (P—pattern characterization). Consequently, there exists a unique pattern length, characterization and location, or YLP, for each profile value.

The address record consists of the address and key, and is called a mailing label when it is printed out as a member of an answer set. A typical mailing label is illustrated in Fig. 4. The first line represents

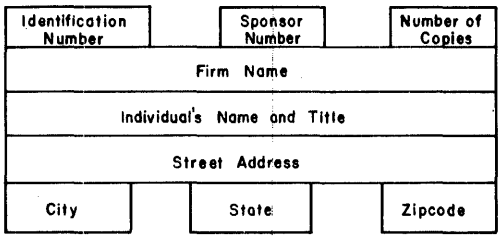


Figure 4. A typical mailing label.

the key terms and the last four lines show the address terms. The key terms are dependent upon the requirements of the organization, but should include terms which link the matrix record with the address record, identify the subscriber's sponsor and provide useful gathering data. All terms in the address record are single-valued.

Updating. As mentioned earlier, an update event may originate one of three file maintenance requirements, these being: addition of a new record, deletion of an existing record, or changes in an existing record. Each update event may initiate one or more of four basic update routines depending upon the nature of the update data, as:

1. Cancel a record.
2. Delete a term.
3. Add a term.
4. Overlay a term.

The second and third routines are particularly suited to handle updating of the profile terms and to accommodate multiple-valued terms in the classification. Figure 5 illustrates a method of portraying

Mail Code	Category 1	Category 2	Category 3	Category 4	
1501	A		D	A	

Figure 5. A method of portraying profile update data on a change notice form.

profile update data on a change notice form. The data are interpreted as: Add (A) profile values 1501-1 and 1501-4, and delete (D) profile value 1501-3 from the subscriber's record. This scheme requires a minimum effort on the part of the sponsor preparing the form, facilitates keypunching, and relegates the conversion of the category description to a digit by the most natural medium—the computer. The fourth routine is designed to update all single-valued terms in the subscriber record, except the profile terms.

The file is updated before it is searched so that the answer sets reflect current information. A list of card errors is automatically prepared during the updating process. After these cards are validated at the control center, the input data forms are returned to the respective sponsors. The sponsor discards the carbon copy which he had retained as a temporary record and replaces it with the original input data form. In addition to the list of card errors, a list of changes is automatically prepared so that the control center can make a one-to-one correspondence between the changes which were to be made and those which had actually occurred.

Searching. The control center prepares the queries and schedules the computer runs. The matrix records which satisfy the search specifications are matched with their corresponding address records,

and the resultant answer set may be a list of mailing labels, list of counts, list of mailing labels with corresponding pick values, special list with selected data, or any combination of these answer sets. The capability to prepare as many as 10 separate lists during one computer run is provided to yield an economical scheduling program. The batching of queries and scheduling of computer runs play an important role in decreasing query cost as query activity within the organization increases. A charging rate for any query may be established by automatically assigning a numerical weight to each primitive in the search specifications, summing up the total weight, multiplying this figure by a predetermined number of dollars, and then multiplying this dollar value by a factor which varies directly as the number of labels in the answer set. This provides a convenient means for justifying the cost of the query to the user.

It is advisable to supplement regular queries with statistical queries to obtain information which is useful in improving operations and in anticipating future needs. For example, a count of the number of profile entries having a specified value would be useful in controlling the number of corresponding publications to be printed at a future date.

Sorting. Since companies with large mailings are required to sort their lists by zip code, it is important that this facility be incorporated into the system. In addition, the system must have the facility to sort on other fields such as the sponsor's number and the subscriber's last name. For example, prospects responding to the company's ads or releases are candidates for its product information and promotional material. Every three months a list may be extracted from the customer's subfile by last name, first two initials and firm name. This list is sorted by the customer's last name, and the names of the verified prospects are matched against the list manually, weeding out duplicates. The option for adding the remaining prospect names to the customer subfile is then initiated by informing the appropriate sponsors to contact the prospects. If a prospect shows interest in the company's products, services and/or policies, then the sponsor will prepare an order form in his behalf. The procedure is economical and provides a linking of the company's marketing information dissemination system with its inquiry program.

Output. Since the number of mailing labels in the answer set ranges from several hundred to tens of thousands, depending upon the search specifications, the computer printout format must be designed to minimize printing cost and yet be compatible with the labeling, gathering and distribution processes. Three types of formats will be presented, but these are by no means exclusive. The type I format is a list of mailing labels, and the type II and III formats are lists of mailing labels with corresponding pick values. In the illustrations to follow, each format appears on a standard printout sheet which is perforated into four equal horizontal strips to permit universal and economical usage of each sheet.

An example of a type I format is illustrated in Fig. 6. The maximum number of horizontal mailing

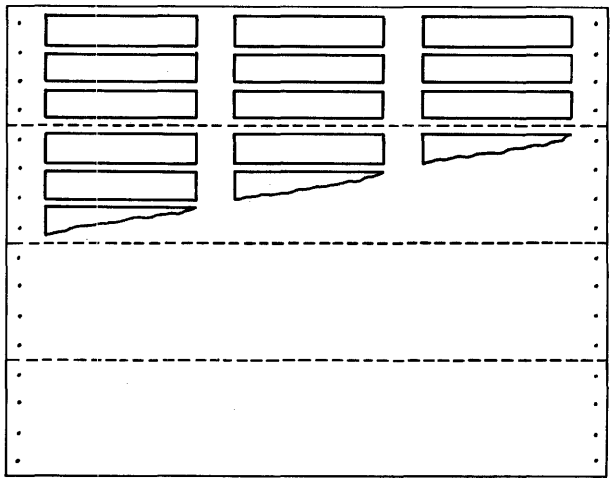


Figure 6. Type I of printout format.

labels is primarily determined by the number of printer keys, type of printout paper and the labeling or imprinting process employed. In this example, nine mailing labels may be accommodated on each strip or, in total, 36 labels on each sheet. The type I format is applicable to either the employee, distribution outlet or customer subfiles.

Figure 7 depicts an example of a type II format. This format is responsive to the answer set generated by interrogating the customer subfile with a PICK primitive having a large number of arguments. The pick list and the corresponding mailing label are printed in a prescribed arrangement on a strip. This arrangement, which is designed in conjunction with the characteristics of a window envelope, has advantages which will be discussed later. The number of pick values which can be listed on each strip depends on the dimensions of the strip,

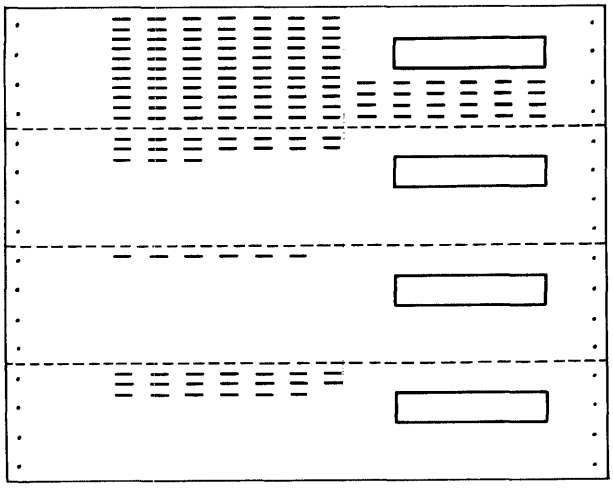


Figure 7. Type II of printout format.

the size of the window envelope, the readability of the pick values and the practical limitation of 'stuffing' the envelope, that is, whether there is more publication material than the envelope's design permits. If the number of pick values exceeds the strip's capacity, then the mailing label is repeated on the next strip and the pick list is continued.

An example of a type III format is shown in Fig. 8. This format is responsive to the answer set generated by interrogating the employee or distribution outlet subfiles with a PICK primitive having a small number of arguments. The pick list is printed under the corresponding mailing label in the arrangement illustrated. If the pick list exceeds the allotted number of possible pick values, then the mailing label is repeated on the next strip and the pick list is continued. The type III format has the advantages which will be discussed later.

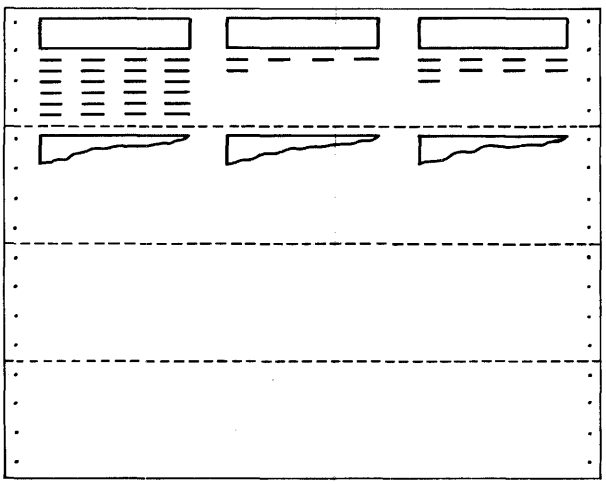


Figure 8. Type III of printout format.

Each sheet of each of the types is headed by a numeric code which portrays the list and page numbers. This is useful information when multiple lists are to be printed. In addition, the query narrative and statements are printed at the beginning of each list along with a count of the number of labels in that list.

Phase III—Labeling, Gathering and Distribution

Labeling or Imprinting. There are several excellent processes for affixing or imprinting labels from the computer printout to an envelope or a publication. A labeling machine performs the first operation, and a heat transfer machine effects the second; in more recent machine designs, one machine performs both functions. Imprinting offers a more professional result than labeling; however, the imprinting requires special heat transfer printout paper which is about three times the cost of printout paper used in the labeling process. The machine can label at a rate of 15,000 units per hour and imprint at a rate slightly less than this. Even considering the machine setup time, it is unlikely that the labeling or imprinting process will be a bottleneck in the total system's operations. The labeling or imprinting process is applicable only to the type I listings. Type II and III listings are initiated in the gathering process which, of course, is also applicable to type I.

Gathering. The gathering process may be manual or mechanical, and the selection of one or the other depends primarily upon economic considerations. For subscriber files less than 100,000 records, the cost of a mechanical gathering machine is usually prohibitive. The manual gathering process will be discussed in this section.

The customer type I listing usually consists of more than one list. Each list of mailing labels is responsive to search specifications in which the EQUAL primitive has one profile argument which relates to, in general, a sales promotion publication. Gathering of this publication is accomplished by mechanically inserting the piece in an envelope which has been labeled or imprinted with one of the mailing labels on the list. If the recipient is to receive two different sales promotion publications, his mailing label appears on two lists, and he will receive literature under two separate covers. The reason for this procedure is that the field salesman wants to direct specific material to a customer under individual covers to promote the personal touch.

The employee and distributor outlet type I listings are treated in the same manner as the customer type I listing except that the mailing label is affixed to or printed on a publication rather than an envelope. The publication is then manually placed in a particular bin which is the collection point for all publications to be distributed in bulk to a location within the organization. This scheme permits an economical distribution of literature.

The type II listing is put through a bursting machine which stacks the strips and maintains the zip-code order. Each strip is affixed to a standard cover letter which introduces the contents of the packet.

The publications to be gathered, sometimes referred to as buying data, are stacked separately by profile value. These stacks are in the same order as the PICK primitive arguments in the search specification. The publications corresponding to the pick values on the cover letter are manually gathered and inserted into a window-envelope. The scheme eliminates any labeling or imprinting process, provides a record of publications contained in the envelope for tracing purposes, and reduces confusion in the gathering process. For example, the matching of the pick list label to an envelope label is eliminated. In addition, the cover letter adds a professional touch to the customer's packet.

The type III listing is put through a bursting machine which stacks the strips. The stacks of strips are then slit, resulting in three separate stacks of mailing labels and corresponding pick lists. The publications corresponding to the pick values on each gathering slip are manually gathered and stapled to the slip. This packet is then placed in a particular bin and distributed, with other packets, to a location within the organization.

Distribution. The envelopes and bulk shipments are weighed and the proper postage is applied. Mailings are made through regular channels. The recipient reviews the literature he has received and contacts the sponsor if he desires additional service.

CONCLUSION

A marketing information dissemination system has been proposed to meet the information and communication requirements of large industrial advertisers.

Such a system is in current operation at the Westinghouse Electric Corporation at a cost reduction of two to one over the previous direct mail system. The file was established and is maintained under the Reference Retrieval System, also developed by Westinghouse.[3] The IBM 7094 computer configuration serves as the catalyst of the system and IBM 360-30 computer provides the output medium. The master file consists of 60,000 customer, 5,000 distribution outlet and 8,000 employee records. These can be analyzed and specific mailing lists prepared at an equivalent search rate of 100,000 records per minute. The equivalent search rate is derived by considering that a certain number of queries can interrogate the file in the time it takes to pass the file tape. If these queries were supplemented with an additional simple query, which is defined as an EQUAL, ANSWER, and LIST logic structure, then the system is no longer tape bound. It is this additional query that establishes the incremental query rate which specifies the equivalent search rate.

Large industrial advertisers are becoming increasingly aware that electronic data processing and printing are the solution to their information dissemination requirements. The computerized system discussed in this paper improved the speed, economy, flexibility, generality and capability in comparison with visual and mechanical systems already in use by approximately 100 percent.

ACKNOWLEDGMENTS

The author wishes to thank Dr. P. B. Henderson for his encouragement and assistance, Messrs J. G. Bishop and F. E. Cabron for their technical contributions, and Mr. A. M. McKinney for editing the manuscript.

REFERENCES

1. P. B. Henderson, "System Specifications for Data Retrieval," First International Meeting of Operations Research Society of America, Honolulu, Sept. 1964.

2. ——, "On the Design of Data Systems for File Analysis and Information Retrieval," Eleventh Annual International Meeting of the Institute of Management Sciences, Pittsburgh, Mar. 1964.

3. ——, "A Theory of Data Systems for Economic Decisions," doctorate thesis, Massachusetts Institute of Technology, June 1960.

A NEW LOOK IN A PERIPHERAL EQUIPMENT DESIGN APPROACH

Earl Masterson
Terminal Equipment Group, Honeywell EDP
Waltham, Massachusetts

DEVELOPMENT PROGRAM OBJECTIVES

The initial work of the development group was to produce a basic line of peripherals which would be a major step in freeing the company of its dependence upon vendors. The first product was to be a line of high-speed printers, the second a card reader, and the third a card reader/punch. An Optical Bar Code Reader was in parallel development under a contractual commitment. This machine later developed into a line of machines which, strictly speaking, cannot be considered basic peripherals.

Each machine's specifications were written to at least equal competitor's specifications. Rather than try to exceed the functional specifications, it was decided to place a great deal more emphasis on reliability and serviceability.

It was felt that every effort should be made to make a reliability breakthrough. The great advances made in solid-state circuits, particularly with regard to reliability, has caused an unbalanced condition between the central processor and the peripheral equipments.

Because of an ever-increasing rate of computer systems sales, the importance of company independence in the peripheral area became more immediate.

CONSIDERATIONS LEADING TO THE FORMULATION OF A DESIGN APPROACH GUIDE

The importance of a successful development program depends upon many factors, and the implications of these many factors must be carefully weighed before a satisfactory design approach can be stated. For example, there is probably no one best design for a given product. The design must be considered in the light of its total environment. Described below are a number of factors that went into the formulation of our design approach guide:

Machine Specifications. The machine specifications were the only factor in our favor at the start of this program since they did not dictate a major breakthrough in speed or other functional performance. This left us free to choose between the design approaches of our competitors and something altogether different. In the following paragraphs we will show why we chose the second alternative.

Development Time Cycle. The short overall program schedule required that serious consideration be given all methods which could possibly lead to shortening the overall development and manufacturing start-up cycle. One of the very time-consum-

ing development problems typical of electro-mechanical design is that of the rather lengthy cycle of design, life test, redesign, life test, etc. Some method of reducing this time had to be found. The successful search for ways to eliminate moving parts and to use more "off-the-shelf" mechanical and electro-mechanical components was a great help in this area.

Engineering Background. Assignment of this rather large development program to our group also had its effect on other parts of the engineering organization. For example, while a few of our associates in the circuits, logic, systems, and software groups had worked with our tape drive development group, most were more accustomed to working within the fixed restrictions of vendor devices. This involved the selection of the vendor equipment, studying its characteristics which were normally fixed and known, and then designing the necessary interfacing software and hardware to work effectively with the device. We were now forced to impose on the other groups within engineering the problem of design where initially nothing is fixed or known and where every separate approach or change has far reaching and interacting results.

Manufacturing Considerations. Our manufacturing organization had extensive experience in the production of large quantities of electro-mechanical tape drives and had an even more substantial background in the production of electronic equipment. In view of this, we felt the time was not inappropriate to emphasize an electronic rather than the traditional highly mechanical peripheral equipment design.

Field Service Background. Because of the complexity of the electronics involved in modern high-speed computers, Field Service personnel with heavy emphasis on an electronics capability have been and will continue to be employed. We felt that the Field Service personnel would have fewer problems in familiarizing themselves with our new equipments in view of the accent on electronics.

Reliability Breakthrough. Electro-mechanical peripheral equipment is an important part of all computer systems and is becoming an even larger part each year. (See Fig. 1.) While thousands of successful computer systems are in operation with many thousands of electro-mechanical peripherals, it is doubtful that anyone would agree that the reliability of the peripherals has reached the point where they are balanced with the reliability of the modern solid-state central processor. Computer

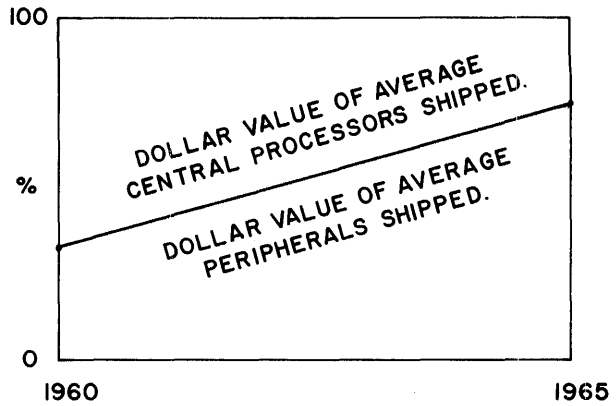


Figure 1. Change in data processing systems. The chart indicates the shift in importance of peripherals in the average system in a five-year period in our company.

circuit developments have progressed in a relatively short period from a time when a single flip-flop required four vacuum tubes to a point where sixteen flip-flops can be produced on a single tiny chip. Component development in the area of electro-mechanical design has not had this kind of improvement. It often seems that the typical design engineer errs in the direction of overconcern for manufacturing costs, not reliability. Where so much of our equipment is on rental, reliability is even more important and it seemed that a new evaluation should be made in developing the design approach. We realized that this program provided the once-in-a-lifetime opportunity to make a breakthrough in a design approach for reliability because there was no requirement to "warm over" an old product line.

It might appear that a study of other successful electro-mechanical industries could provide a design approach guide which would make a major improvement in peripheral equipment. While this study did not uncover an industry with an equivalent design requirement, it did prove to be a very worthwhile appraisal; and while it may not have shown a direction to proceed, it did show directions that should not be pursued.

The automotive industry appears to be based on a very successful application of what might be called an electro-mechanical design approach. It might seem that hiring automotive engineers and adopting automotive design principles would be a way to insure development of a successful line of electro-mechanical peripheral equipment. Our study indicated that this would be far from a wise decision. For example, it appears that the average automobile

is in operation somewhere between 5 and 10% of the time, whereas many of our peripheral devices are in use two and three shifts a day, six days a week. The average car in the United States is driven approximately 12,000 miles per year. One indication that this is accumulated on a very low duty cycle rate can be seen by the fact that if one were to drive at a rate of 60 miles per hour for only 8½ days, it would accumulate 12,000 miles. This rather dramatic difference in use factor shows that the adoption of automotive design principles would be very unwise. In other words, while automobile troubles and maintenance requirements seem to occur very infrequently, they would occur 10 to 20 times as often if the same principles were used for peripheral equipment operating on a three-shift basis.

DESIGN APPROACH OBJECTIVES

The foregoing was weighed and considered in developing a design approach guide which would be readily understood and used to advantage by the many individuals contributing to the program. Some of the objectives may sound as though we set the goals too high. A later summary will indicate that achievement of these goals was surprisingly complete.

Fewer Moving Parts. One of the obvious ways to eliminate manufacturing and maintenance problems with electro-mechanical equipment, and to achieve reliability is simply to eliminate moving parts. It was surprising with this as a goal, how many parts were eliminated compared to the typical approach for similar machines. Engineers received much more credit for simplifying a design than for solving problems in a complex design.

Less Mechanical, More Electrical. When considering a design approach, it can be seen that a particular function can in many cases be performed either electrically or mechanically. When only manufacturing cost is considered, the decision is quite often made to use a mechanical approach. However, when we considered such other key factors as production quantity, tooling, life testing, the electrical-electronic capability of the organization, the machine reliability, and the expanding state-of-the-art in electronics, it became quite apparent that we should use nonmechanical techniques as much as possible; and to our pleasant surprise, the manufacturing costs were found to be comparable.

Choice of Moving Parts. If a particular function cannot be performed electrically, and must be done mechanically, there is also a good decision and a poor decision as to the kind of moving parts that should be used. We established a scale of desirable types of movements and believe that of all the types available, flexing is perhpas the most trouble-free and gives the longest life. For example, a quartz crystal in an oscillator circuit actually changes its physical shape every cycle; in other words, a mechanical flexing occurs that may take place millions of times per second and may run trouble-free for many years.

Another example is found in loudspeaker design where motion may be a sizeable fraction of an inch and occurs thousands of times per second. This indicates that a flexure spring operating well within its design limits should provide a very satisfactory answer to the need for reciprocating mechanical devices.

Reasonable-speed rotary motion appears to be the next lower level of desirable form of mechanical motion and very satisfactory bearings can be provided at least for the life of the subject equipment. Perhaps the next lower level of satisfactory mechanical motion could be termed pulsating rotary motion. This can be thought of as having some of the characteristics of both of the first two motions, but imposes an additional problem of bearing design. The next type of mechanical motion, as we go down the scale of desirable motions, is a sliding or rubbing motion. This in general, is not a very desirable motion from a trouble-free life standpoint. It is usually very dependent upon a satisfactory solution of many physical guiding, metallurgical, and lubrication problems and, even then, does not generally lead to a life quite satisfactory for our needs.

The least satisfactory of all moving parts, perhaps, are those that impact with each other. Here, two or more surfaces are brought together under shock conditions and satisfactory operating life will vary greatly depending upon many factors. Later examples will show that by establishing design objectives, the more desirable types of mechanical motion can usually be employed.

"Off-the-Shelf" Mechanical Components. When a mechanical design became mandatory, we attempted whenever possible to use standard "off-the-shelf" components such as sealed ball bearings. This was done for all simple rotary applications and also for primary elements in eccentric systems as substitutes for cams and followers. This change from the typical approach greatly reduces the amount of life testing required; lowers the product

cost; requires less tooling for manufacturing; and in general, shortens both the development and manufacturing times.

Elimination of Field Lubrication. Early in the development of our new design approach, it appeared that our objectives would make possible the elimination of field lubrication. Although some may argue that this is not an important objective, experience tells us that devices that do need lubrication inevitably receive too much, too little, or the wrong kind of lubrication. In addition, lubrication in the presence of dirt, in particular card or paper dust, can create a very abrasive mud. Therefore, it seemed not only important but mandatory to pursue this as a design objective.

List of 27 Don'ts. Reproduced below is the list of "27 Don'ts" which was given to each designer engineer. It was written early in the program and used as a guide for developing equipment to be manufactured for service and use in the total environment found in the data processing business. Most items were listed because of concern about reliability, development time, maintenance cost, manufacturing cost, and quality control. Some were listed to eliminate design decisions based on the emotional "we did it this way before," and to insure pursuit of more challenging and imaginative routes. In either case, the list was provided as a guide and people were cautioned that if a rule had to be broken, it should be done with the full knowledge that difficulty might develop.

"As a matter of last resort in a data processing machine design, use:"

1. Field lubrication
2. V belts
3. Miter or bevel gears
4. Worms and gears
5. Helical gears
6. Chain drives
7. Three or more bearings in line
8. Needle bearings
9. Motors with centrifugal starting switches
10. Linear ball or roller bearings
11. Bearing length-to-diameter ratios of less than 5:1
12. Switches in low-level circuits
13. Shims
14. Surface plate and height gauge assembly techniques
15. Slides requiring parallel ways
16. Ball bearings in short-stroke oscillating applications
17. Low-force interposer or actuator parts depending on side guiding
18. Cams and followers
19. Long compression springs
20. Cap screws everywhere
21. ABEC7 bearings everywhere
22. Glue, cement, paste, and tape
23. Tight tolerance on sheet metal parts
24. Trapped belts
25. Open switches
26. "Ship in bottle" assembly techniques
27. Interacting adjustments

Our design engineers' first reaction, as can be expected, was that their hands were completely tied and they had nothing to work with. It was soon shown by means of frequent design review meetings that for every "Don't" there was a very satisfactory substitute.

EXAMPLES OF MORE DESIRABLE ELECTRO-MECHANICAL COMPONENTS

The following examples are for dynamically operating components and do not necessarily apply to more-or-less static devices such as operator controls:

Flexure vs Pivots

A flexural member operating well within its stress limits, as mentioned previously, seems to be one of the longest-lived mechanical elements known. An excellent substitute for pivot bearings in a reciprocating mechanism, therefore is a protected flexure spring. It has the very desirable characteristics that it requires no lubrication, is not sensitive to dirt, paper, or card dust, has substantially no friction, and does not generate heat. It does have a spring restoring action, but this is usually of no consequence and sometimes may even be desirable. It seems almost a paradox that a spring is rated so highly when everyone has seen broken springs. Broken springs are inevitably the result of improper design, improper manufacture or improper use, all of which can be controlled. The stresses in a spring can be readily calculated in most cases, and, in all cases, can be life-tested. Proper manufacture can be controlled by normal quality control procedures. Improper use can be controlled by proper design. For example, we have a rule which states that each

flexure spring system must have limit stops which prevents accidental overstressing. In many cases, the rule has even been extended to flexure subassemblies to further insure against accidental damage. Figure 2 shows how a flexure spring can be substituted for a pivot and how simple limit stops can be designed to prevent overstressing.

However, while this system provides essentially parallel motion, it does not produce pure straight line motion; in many cases though, the deviation is so slight that it may be ignored. For example, in our card punch, the interposing system is carried on four two-inch flexure springs. The total stroke of the interposer system is 0.1 inches and the resulting deviation from a true straight line motion is 0.002 inches.

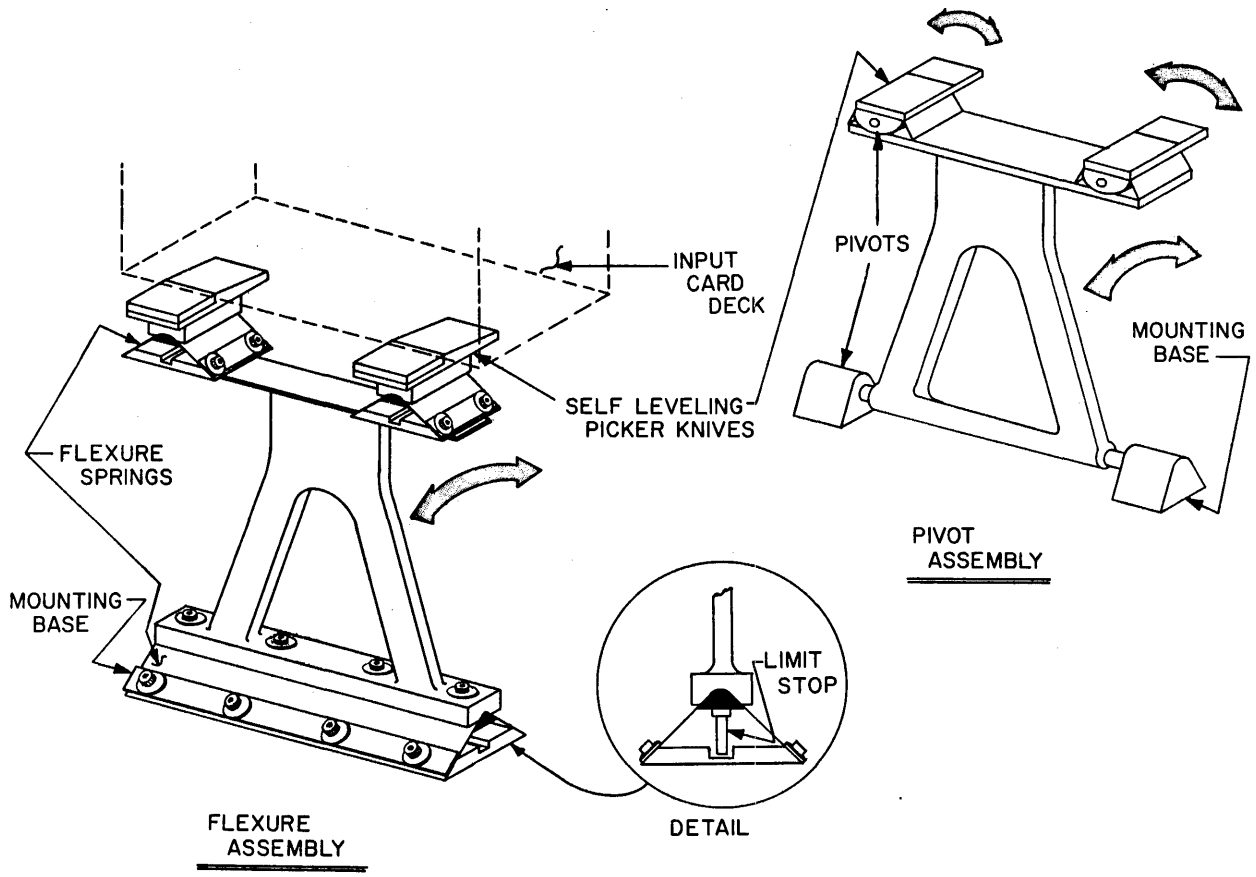


Figure 2. Flexures vs pivots. Comparison of flexure springs and pivots applied to the same card feed picker knife assembly.

Parallel Motion Flexure vs Slides

Mechanical elements which normally require slides to provide parallel motion can be greatly improved by using parallel motion flexure springs. As noted previously, slides are not very high on the preferred list of mechanical motion. It is very difficult to prevent metal-to-metal rubbing contact and the system is usually very dependent upon lubrication. By using parallel flexure springs (Fig. 3) it is possible to completely eliminate sliding friction, lubrication, and heat. In addition, the system will never develop play or clearance during its operating life.

Sealed Ball Bearings vs Sleeve Bearings

Sealed ball bearings usually result in a higher product cost than simple sleeve bearings. It is our belief that this additional cost is easily offset by reduced maintenance. The total cost of the downtime for a failed bearing will probably pay for the additional cost of providing ball bearings throughout a machine. Sealed ball bearings in most applications have the added advantage of requiring no

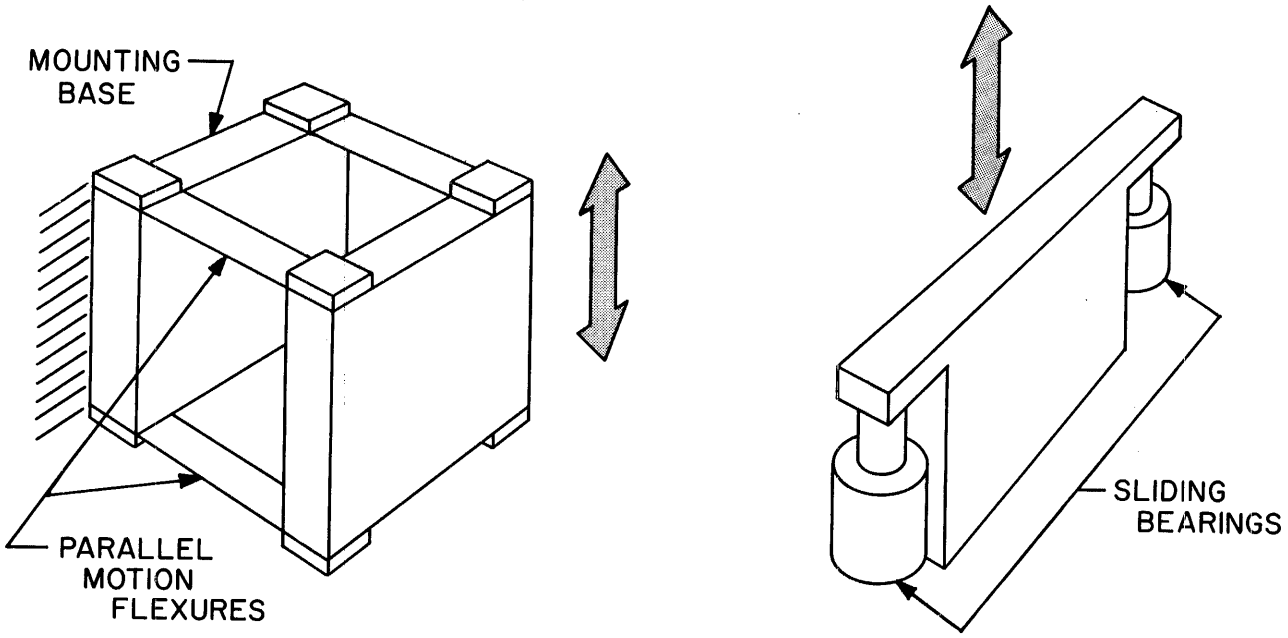


Figure 3. Parallel motion flexures vs slides. Comparison of two methods of supporting and guiding an assembly which is in reciprocating motion.

field lubrication, having very low friction, producing very little heat and possessing constant characteristics throughout their life. Sealed ball bearings are good examples of "off-the-shelf" mechanical components in which a solution to all the problems of metallurgy, surface finish, and lubrication required for a predictable life has already been found by the bearing manufacturer.

Eccentrics vs Cams and Followers

Cams and followers (Fig. 4) have the desirable characteristics of being able to convert rotary motion into controlled reciprocating or linear motion. Cams and follower systems in our applications, unfortunately, have many undesirable characteristics. For example, since the basic shape of cams and followers is cylindrical, the contact between the two is essentially a line contact. While this is somewhat modified in practice by the elasticity of metals, the fact remains that the unit pressure can be exceptionally high. A single cam and follower provide positive drive in one direction only and must rely on a return spring for drive in the opposite direction. A system with two cam surfaces to provide drive in both directions may eliminate the return spring, but in so doing must become a very precise assembly. Other disadvantages of cams and followers are that they require lubrication, and bearing problems sometimes develop because of the exceptionally high speed of the rather small diameter cam follower. To date, we have found it possible to provide all our reciprocating drive needs by using sealed ball bearings in eccentric drive systems. This allows designs with no lubrication, no return spring, low unit forces and "off-the shelf" package assemblies. The time displacement curve of an eccentric system is normally very close to a sine wave. In our applications we have found this potential disadvantage to be a very minor price to pay for the many advantages.

Flat Belts and Pulleys vs Gear Trains

Gear train power transmission systems providing rotary motion to a number of shafts characteristically have the disadvantage of requiring very accurate shaft center locations, lubricants and relatively high-cost components. Modern, thin, flat nylon belts running on smooth pulleys (Fig. 5) solve many of these problems. The cost of the entire system is much lower, center locations are much less critical, the system requires no lubrication, and problems of tolerances of all kinds can be absorbed in one spring-loaded belt-tightening idler. The life of a properly designed system of this type is exceptionally long. The reason for this is that the belt is very thin and pliable and readily wraps around the var-

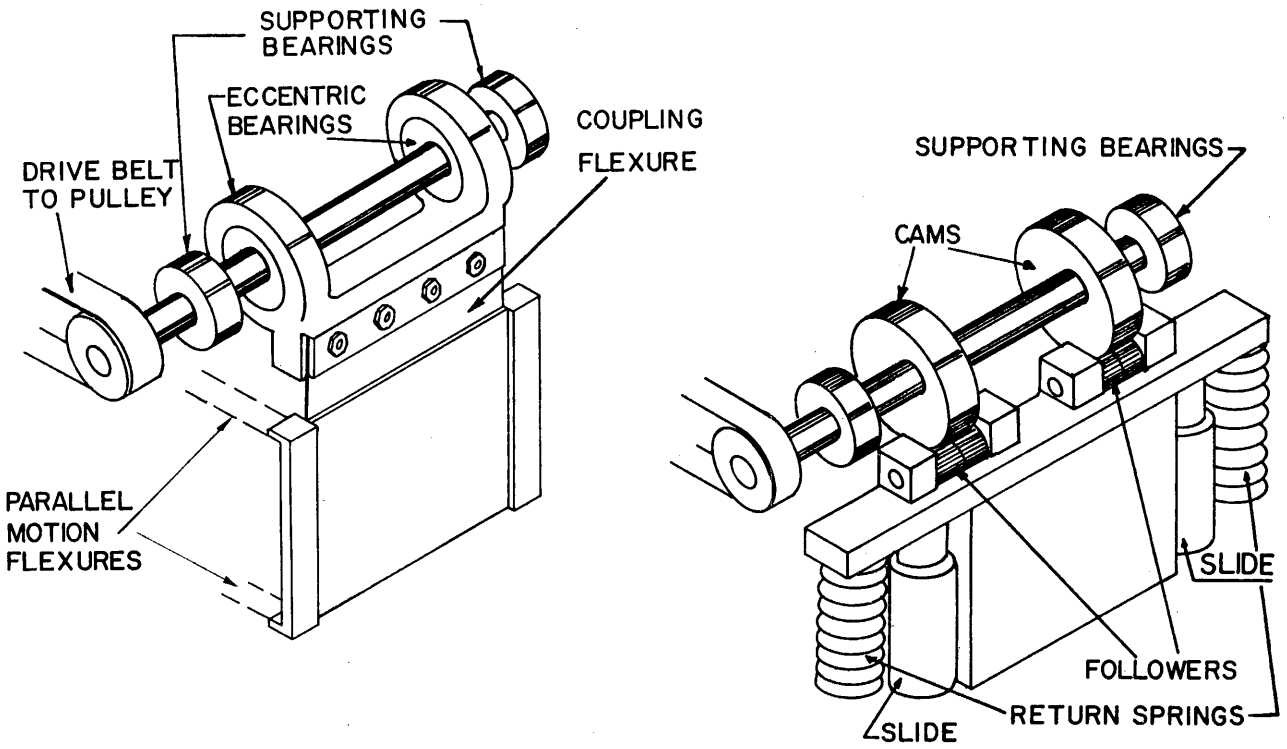


Figure 4. Eccentrics vs cams and followers. Two methods of applying controlled driving power to a reciprocating assembly.

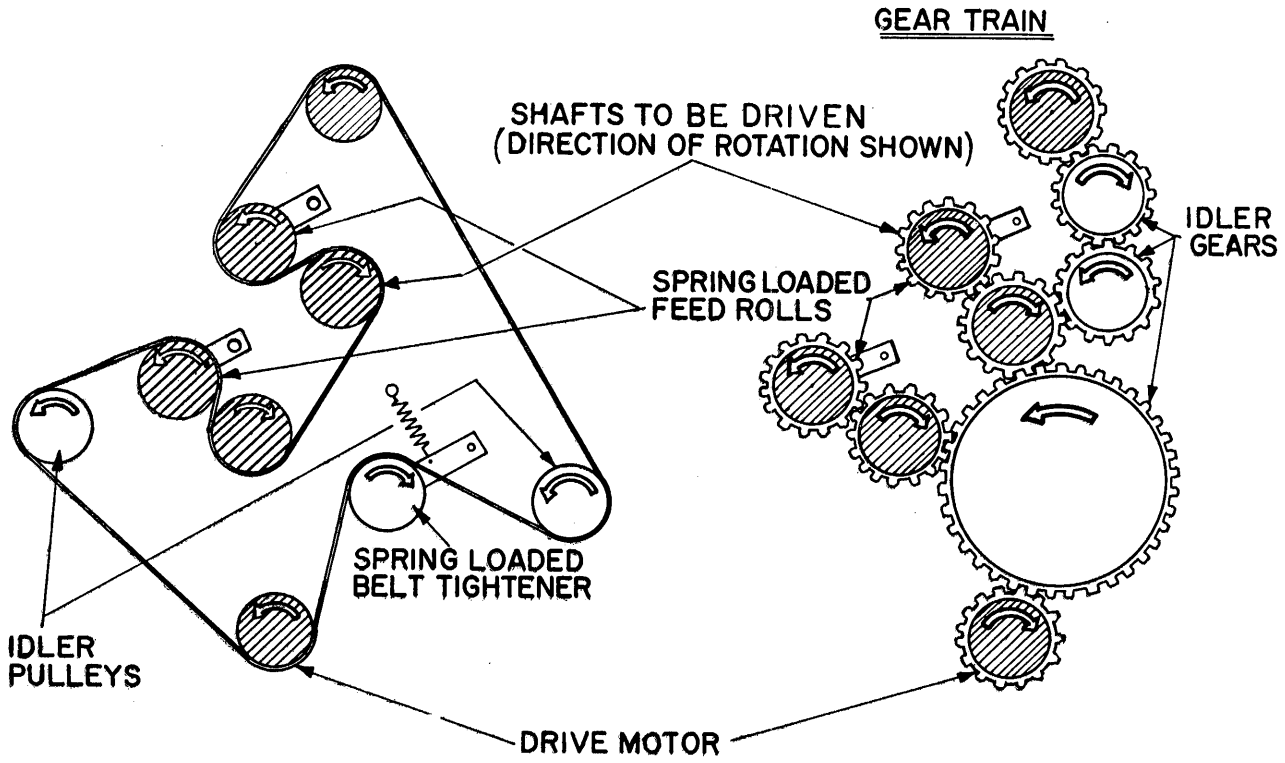


Figure 5. Flat belts and pulleys vs gear trains. Shown on the left is an actual flat belt system used to drive rolls in a card transport system. A possible gear train drive is shown on the right.

ious pulleys. It can be guided by one or more crown pulleys, and therefore, is not in rubbing contact with any element. A drive system of this type, it might be argued, has the disadvantage of not being synchronous. One answer to this is to make the entire machine asynchronous—which is no handicap. If this cannot be done, a timing belt can be employed as discussed below.

Timing Belt vs Gear Trains

Toothed belts or timing belts are a very satisfactory substitute for gear trains when it is necessary to rotate two or more shafts in synchronism. While the timing belt may not be as precise as certain high-quality gear trains, it appears that they are completely adequate in our applications. The timing belt has many of the same advantages over gear trains as does the flat belt. For example, it does not require lubrication, it is not sensitive to wide tolerances on shaft centers, and all tolerances can be absorbed by a single spring-loaded or adjustable idler.

Moving Coil Motors vs Clutches and Brakes

The introduction of moving coil motors has perhaps been as close to a breakthrough in the electro-mechanical component field as anything known. This is a DC motor in which the rotor consists of a very-low-inertia copper coil. The coil can be either a disk or a cylinder. This type of motor has many desirable characteristics: very high torque-to-inertia ratio, low inductance, does not saturate (air core), operates on low voltage and is very compatible with solid-state driver systems. Also, because of low voltage and inductance, it produces very little brush commutator arcing. We have found that all functions normally provided by a clutch and brake system can be easily accomplished by one or more of these motors in a servo system. The degree of servo sophistication depends entirely on the application. A motor in this application has many desirable characteristics such as no high-friction suffaces, no impact of mechanical parts, no change in operating characteristics throughout life. In addition, it provides its own power. Figure 6 shows the simple paper-feed drive system that can be used in a high-speed printer in place of the typical clutch and brake. Depending on the servo system, it can provide not only start and stop functions, but also variable speed and can operate in either direction. This is one of the prime examples of the way in which it is possible to exchange typically troublesome electro-mechanical elements with their many

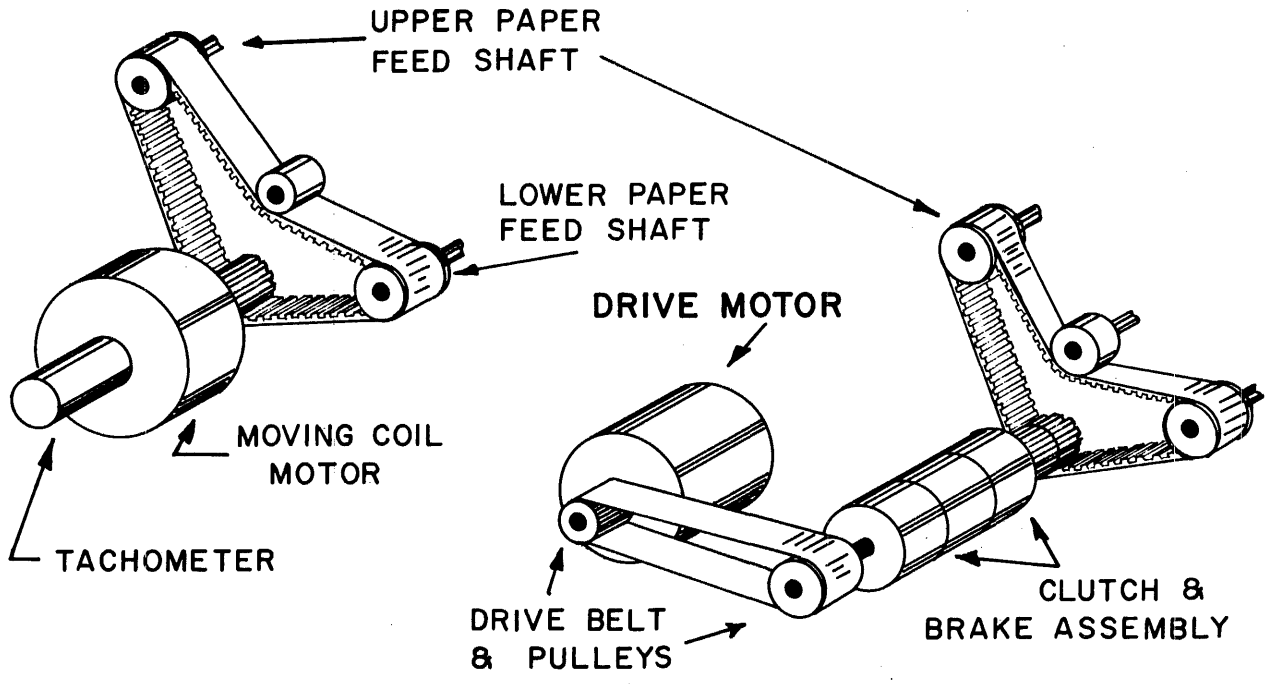


Figure 6. Moving coil motors vs clutches and brakes. Two drive systems to supply controlled drive to the paper feed section of a high-speed printer.

moving parts to a highly electronic system which has only one moving part—a simple low-inertia armature operating in simple rotary motion on sealed ball bearings.

Moving Coil Motors vs Indexing Mechanisms

The same moving coil motor discussed in the preceding paragraph can also be used to great advantage as a substitute for very complex indexing mechanisms which require very long development times because of the many problems of metallurgy, surface finish, and lubrication. Even if the complex mechanism problems are reasonably well solved, providing a device with a trouble-free life, it would still leave something to be desired. A moving coil motor can frequently provide the same indexing function with the additional advantage of very long trouble-free life, plus additional control features such as interruptions of the indexing function for indefinite periods and the substitution of continuous motion at various speeds, even in a bidirectional mode.

EXAMPLES OF GOOD DESIGN PRACTICES

Following are a few examples of good design practice rules which have been applied:

Subassembly vs Ship in Bottle. It is our general rule to make a subassembly of critical inter-related parts so that they can be bench assembled, bench tested, or bench repaired. This may seem like a very obvious design goal, but, unless it is stated, it sometimes becomes obvious too late.

Multi-Motor vs Complex Drives. We have found that it is very often desirable to use several drive motors in a single machine rather than develop a very complex drive to transmit power over large distances. This is perhaps somewhat analogous to the change in power distribution in a manufacturing plant since the days of a system consisting of overhead line-shafts with the power coming from a single large steam engine. Today, this job is accomplished by small individual motors added to each device requiring power. Because of the mass production of fractional horsepower motors, we have found it possible to buy motors for less cost than that of custom-made hardware to transmit power over distances or around corners. The multi-motor drive of a machine also provides the opportunity for a more flexible control during start up, shutdown, or test modes.

Avoid the use of Glue, Cement, and Tape in Assemblies. This may sound like a very odd recommendation in this age when so many successful products make extensive use of these techniques. Again, it must be stressed that this guide is recommended for the design of equipment to be manufactured, serviced, and used in a data processing environment. The successful application of modern cementing techniques is a very specialized business and can only be accomplished if everyone is fully aware of the critical quality control problems. Another way of stating the problem is that because of the limited production quantity and the multitude of parts and assembly problems that must be solved, it is very dangerous to assume that the quality control aspects of an assembly requiring cementing will get proper attention.

Avoid Trapped Belts. Since we have eliminated gear trains and have gone to the use of belt drives, our machines will characteristically have several fairly complex belt drives. We have found without exception, that if the design is started with the idea in mind that the design should avoid trapped belts, it can be accomplished. In all of our machines, any belt can be replaced without tools.

SUMMARY

Four Product Lines

The success of the program can be stated very simply. We have in quantity production four new product lines. Three are considered basic peripherals. This now permits our division to ship complete computer systems with all units manufactured in-house.

Success of Design Approach Rules

The tabulation of Fig. 7 shows the degree of success in both eliminating moving parts and in doing more things electrically and fewer things mechanically. The chart lists a number of machine elements and discusses the typical solution vs our method. It is difficult to do this comparison in a completely unbiased way. We believe that the chart shows a significant difference resulting from our design approach decisions.

Sometime after we had made the decision to use an "expensive" moving coil motor system in the paper feed area of our high-speed printer, we noted that the parts costs were being reduced and that with continuing development work, the entire sys-

tem was being simplified. Finally, we decided it would be interesting to make a cost comparison between the new moving coil approach and our old clutch and brake design. To our very pleasant surprise, the parts costs turned out to be almost identical and because of the mechanical simplicity of the moving coil motor system assembly, labor cost was actually lower.

In summary, then, the effect of the rather high design approach goals can be stated as follows: Of the four peripheral product lines presented, none of the machines contains operating gears,* cams or followers, clutches or brakes, mechanical indexing mechanisms, trapped belts, linkages, and none of the machines requires lubrication.

Attainment of Specifications Goals

Each of the four product lines fully met or exceeded the specification goals because of the more flexible design approaches. In some cases, the specifications can still be revised upward. It is also pleasant to note that, again because of the flexibility of the design approach, future additional improvement in performance can be obtained from each of the machines by a relatively small additional development effort.

Reliability Goals

The story of the success or failure of reaching the reliability goals is still being written. There is nothing to date to indicate that any of the design approach decisions were anything except extremely sound.

*At the present time, some "off-the-shelf" small, sealed gear head motors are in use. They do not require field lubrication.

CARD PUNCH ELEMENTS

ACTIVE COMPONENT	SUMMARY COMMENTS ON COMPETITIVE PUNCHES	HONEYWELL PUNCH
GEARS	ALL HAVE GEARS. ONE KNOWN MAKE HAS AS MANY AS 43.	0
CAM & FOLLOWER SYSTEMS	ALL HAVE CAM & FOLLOWER SYSTEMS. SOME AS LOW AS TWO, UP TO AS MANY AS 26.	0
CLUTCH & BRAKE SYSTEMS	RANGE FROM ONE TO TWO.	0
MECHANICAL INDEXING SYSTEMS	THE COMMON METHOD OF INDEXING A CARD FOR PUNCHING.	0
SLIDING ELEMENTS (IN ADDITION TO PUNCHES IN GUIDES)	THE COMMON SOLUTION IN PUNCH BAIL, STRIPPER BAIL, CARD FEEDING, AND OTHER AREAS.	0
PIVOTS & LINKS (IN ACTIVE OPERATION)	IN GENERAL USE THROUGH-OUT. UP TO FOUR PER INDIVIDUAL PUNCH INTERPOSER SYSTEM.	0
BELTS	RANGE FROM NONE TO A DOZEN WITH SOME TRAPPED.	4
FLEXURES	RANGE FRON NONE TO A FEW.	10
SERVO SYSTEMS	NONE KNOWN	2
LUBRICATION AREAS	RANGE FROM AN OIL BATH CRANK-CASE TO 30 AREAS SOME OF WHICH ARE IN SETS OF 80.	0
TRANSISTORS	RANGE 300-500	800

Figure 7. Comparison of card elements, showing effect of design approach.

ACKNOWLEDGMENTS

This paper has reported on what is believed to be an important advance in the design approach of peripheral equipment. The author wishes to thank the many individuals and groups for their contributions in making the effort a success.

A SERIAL READER-PUNCH WITH NOVEL CONCEPTS

David W. Bernard, Frank A. Digilio, Frank V. Thiemann, and Ronald F. Borelli
Honeywell EDP, Terminal Equipment Group
Waltham, Massachusetts

INTRODUCTION

A card punch traditionally has been looked upon as a machine which is composed of many complex and ingenious mechanisms because of the complex task that it must perform. Complex mechanisms, however, have an ingenious way of making noise, being unreliable and difficult to service.

An attempt has been made to break away from the historical approach, by taking a new look at the real requirements and producing a design using novel electromechanical and electronic components which avoid many of these difficulties. Some of the design considerations are presented here. While the total engineering program capitalized to a large extent on a concurrent card reader development, this reader-punch development included all engineering phases from feasibility study through production models and reliability studies. The success of the program can be attributed in large measure to the use of "off-the-shelf" mechanical components with known life and reliability, and the use of solid state electronics and new motive devices and actuators to perform functions formerly satisfied only by mechanical devices.

THE SERIAL READER-PUNCH

The Honeywell 214 Serial Reader-Punch reads at 400 cards per minute or reads and punches at 100 to 400 cards per minute depending upon the number of columns punched. See Fig. 1.

Figure 1. The Honeywell 214 Reader-Punch

The basic card transport, input hopper, read station, and stacker of this machine are modeled after and use parts common to the Honeywell high-speed card reader which was under development at the same time.

A card is picked from the input hopper, and fed broadside into a wait station. From here it is fed serially through a read station, then through the punch to a cornering station where it is kicked

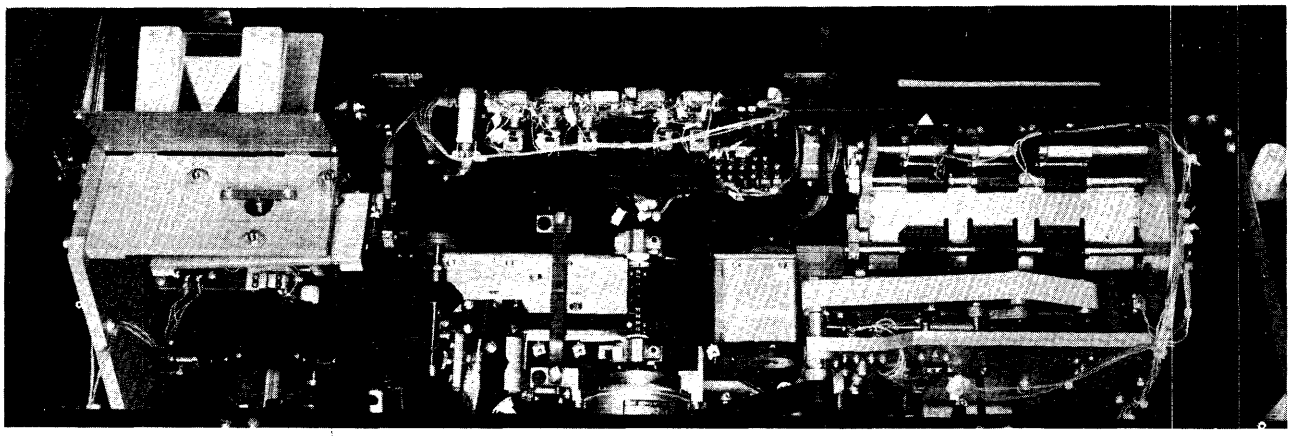


Figure 2. Transport with punch head removed.

"on-the-fly" into rollers for broadside power stacking. Figure 2 shows the transport with the punch head removed. Card motion is left to right.

Input Hopper

The input hopper has a capacity for 1200 cards and is tilted slightly away from the operator completely exposing the front side for easy card loading. Cards are selected or picked on demand (asynchronously) by a flexure-mounted picker knife under the control of a high performance servo system in lieu of the conventional clutch. The picker knife system is shown in Fig. 3. Use of flexures for articulating the knives, pivoting the arm, and connecting to the drive crank should be noted. Sealed bearings are used at the crank pin and in the moving coil servo motor. Motor acceleration and deceleration are controlled by an SCR reversing bridge, via a tachometer generator and differential amplifier. The rest position of the picker knives is determined by photoelectric detection of the motor shaft position.

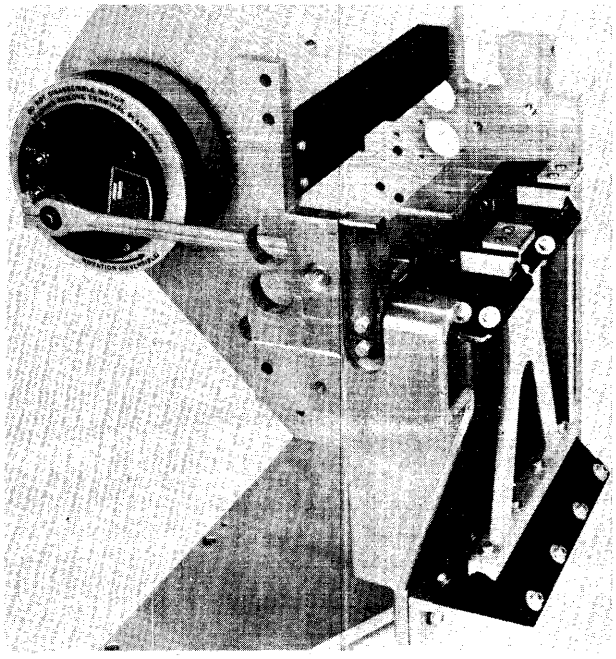


Figure 3. Picker knife and drive system.

Cards enter feed rolls and are driven into the wait station. These continuously running feed rolls and all others associated with the transport are powered by nonsynchronous induction motors through noncaptive flat belts thus eliminating the need for gear trains and lubricants as well as the close tolerances associated with fixed-center shafts. From the wait station, card motion in a serial or column by column manner is initiated by a solenoid actuated pinch roll operating against a continuously running capstan.

Read Station

Additional feed rolls continue the card through the read station at 44 inches per second. These are belt driven and are part of the read station subassembly. The read station consists of 12 solar cells for reading the data and additional cells for strobe generation from the trailing edge of the card. A single projection lamp provides nearly uniform illumination over the entire surface of the card.

As the trailing edge of the card uncovers apertures over the strobe cells, a staircase waveform is generated corresponding to the column locations. Since the trailing edge is continuously referenced,

strobe pulses are unaffected by minor changes in card velocity resulting from slippage or eccentricity of drive rolls and tolerances are noncumulative. To allow for scored cards, that is cards perforated for later separation, a continuous strip cell for all 80 columns is not used, but rather is divided into 8 segments, each with 10 apertures, arranged to become active sequentially under the control of an associated gate or guard cell which is wide enough to be unaffected by the score in the card. A wide variety of trailing edge cuts are accommodated by a mechanical shutter that switches active apertures between card rows 6 and 7, and 7 and 8.

Stacking

From the punch station which is described below, the card is transported via a flat belt to a second cornering station where it is sensed by photocells for regular or offset stacking. The card is kicked "on-the-fly" into broadside stacker rolls. Figure 4 shows a card leaving the punch station, and another entering the broadside stacker rolls.

The kicker is an armature type solenoid with a self-damping multiple layer flexure plate attached. The design keeps the operating air gaps short while providing 5/8-in throw at the card. A complete excursion occurs in 30 msec. Card edge damage is negligible because the accelerating force is applied gradually to the entire length of the card.

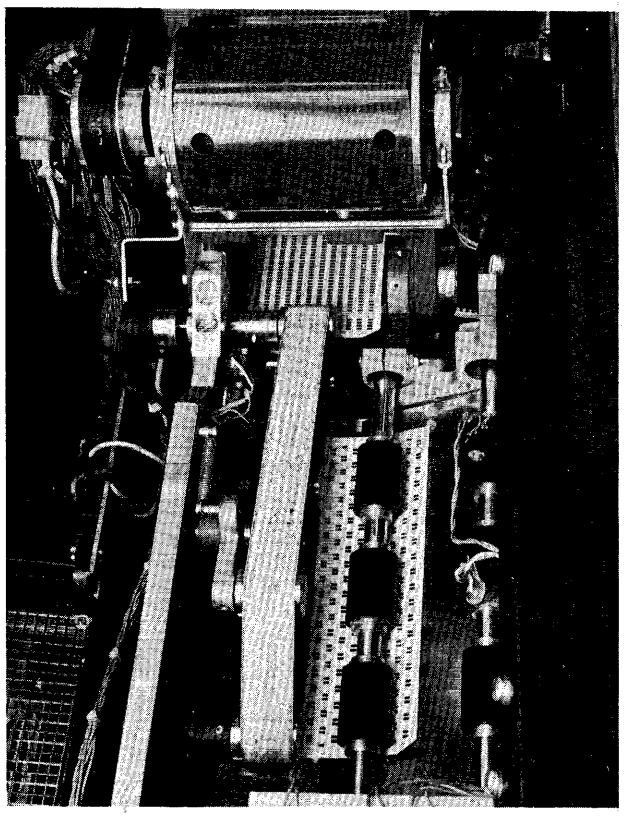


Figure 4. Punch and stacker area.

Since the transport is asynchronous, the card itself is constantly tracked by photocells, and logic conditions are established to determine the next operational sequence or error conditions. For instance, failure to stack, in either the normal or reject modes allows the card to travel to the end of the transport where it signals an error condition and the transport shuts down before a jam occurs.

Punch and Drive

The punches are driven by an eccentric shaft continuously rotating at 4800 rpm. The shaft is flexure coupled to a flexure mounted bail imparting nearly linear motion to the punch interposer carried by the bail. See Fig. 5.

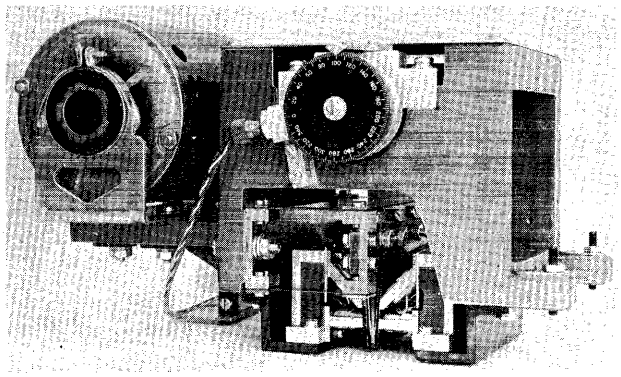


Figure 5. Punch head.

This eliminates cams, cam followers, and rotary pivots. It is estimated that if a cam and cam follower were used in this system, the follower would rotate at speeds in excess of 25,000 rpm, or if a sliding follower system were used a lubricant would be necessary. Every effort has been made to eliminate the need for lubricants throughout this device, including the area subject to most sliding motion and wear and the collection of abrasive paper dust —the punch and die itself.

A two-column punch has been designed in order to keep the speed of the eccentric shaft at half of what it would be for a single column punch with the same throughput and little additional complexity elsewhere. Flexure spring pivots are used throughout to eliminate rotary pivots which are susceptible to fretting corrosion when experiencing small oscillatory motions (Fig. 5). A properly de-

signed flexure spring for this application has a fatigue life of billions of cycles. This is borne out by representative samples of these flexures tested for over 1½ billion cycles without failure, exceeding by far the expected useful life of the machine, under average operating conditions.

Stripping Mechanism

After punching, selected punches must be withdrawn and returned to a rest position. The device which does this is known as a stripper. A precise timing relationship must exist between the punching and stripping actions. While this can be accomplished by the use of gears and cams, this design has avoided these components by making the stripper an integral part of the eccentrically driven punch mechanism. On the downstroke, the eccentric drives only the selected punches through the card. On the upstroke, the common stripper simultaneously withdraws all previously selected punches. Thus proper timing is achieved automatically and phasing adjustments are not required. (Fig. 5).

Punch Selection Mechanism

The heart of the punch selection mechanism is a simple flexure spring which serves as both the armature of the selecting solenoid and the mechanical punch interposer. There is one for each of the 24 punches. This interposer is mounted on the stripper bail between two preformed beams (Fig. 5). In the nonpunching position, it rests against the outside preform where it cannot contact the top of the punch. In the selected or punching position, it moves to the inside preform where, on the next downstroke, it makes contact with the top of the punch (Fig. 6). As punching load is applied, the thin interposer is reinforced by the stiff inside preform. The flexure thus acts like a rigid column and transmits the required punching force without buckling. Since both preforms and the flexure have a common point of attachment, there can be no relative sliding motion between them and therefore no wear. The deflected beam preforms are used here to avoid the manufacturing problems inherent in contoured machined parts. Since the interposer is carried by the reciprocating assembly and selection is done magnetically, the relative motion which normally exists between the reciprocating and fixed members and results in mechanical wear has also been eliminated. The interposers are held in a normally nonpunching position by a permanent magnet. Because of their initial preload, the interposers

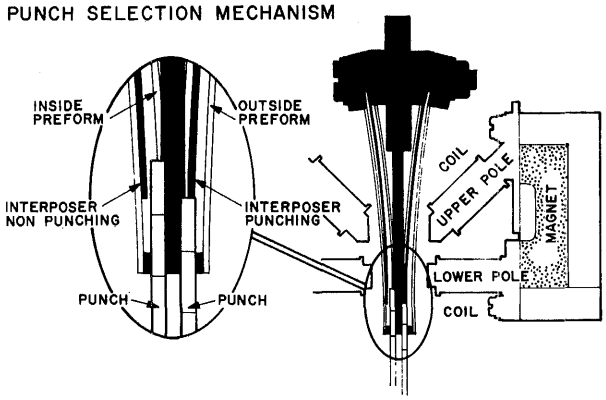


Figure 6. Punch selection mechanism.

move to the punching position, when the field of the permanent magnet is reduced by an electromagnet. This punch selection system has only one moving part, the interposer, which does not slide, pivot, or impact. The change of the interposer from the nonpunch to the punch condition occurs at the top of the stroke when a clearance is established between the punch and the interposer. There is no impact between the interposer and the punch because the action occurs very near the top of the stroke where the velocity is essentially zero.

Punch Head Accessibility

The punch head mechanism can be opened by an operator to clear card jams in a matter of seconds (Fig. 7). This feature is provided by splitting the punch head at the junction of the die and punch guide. In this case, the increased manufacturing cost of splitting the head is justified on the basis of reduced maintenance cost and increased uptime and throughput.

Punch Check

In a punch of this type a means is generally provided for verifying that correct punching has occurred. Short of reading the card immediately after punching, which is an unwarranted expense, a check on the motion of the punch or its actuator can be incorporated. Several methods of checking this punch were considered and rejected. Mechanical contacts were rejected as being unreliable and prone to failure because of wear. An echo check generally involves sensing the position of the actuator armature at a specific time during the cycle. This was rejected as being too far removed from the actual punching and difficult to implement for this particu-

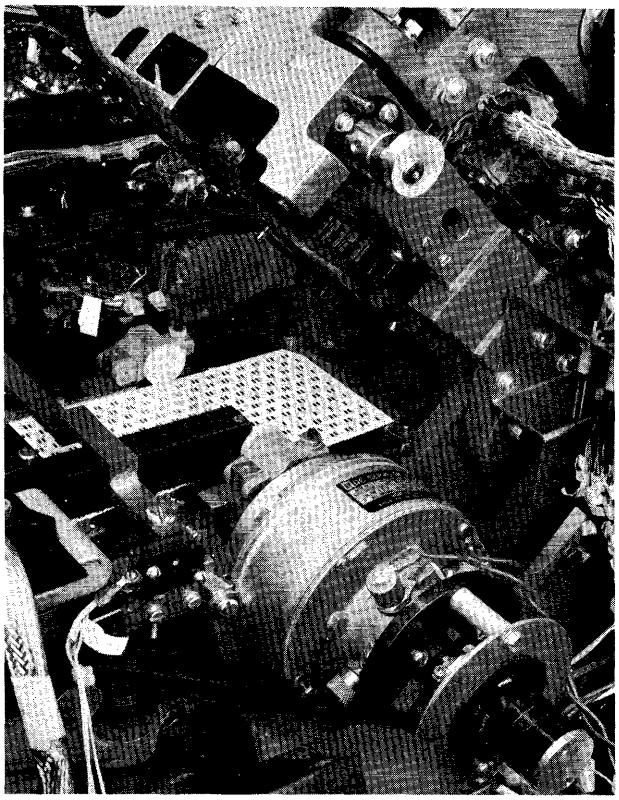


Figure 7. Punch station, punch head open.

lar interposer. Reluctance pickups would detect the motion of the selected punches, but signals would be small because of the low velocity, and peak output would occur at midstroke before the punch pierces the card.

The method selected is a fail-safe position detector which senses the punch at its maximum excursion fully through the card. With this device the position of the punch is actually sensed and checked against input data when the tip of the punch has gone through the card and into the die. Attached to the shaft of each punch is a tiny permanent magnet with a field strength at its surface of about 500 gauss. Mounted adjacent to each punch is a ferrite memory core which can be saturated by an external field (Fig. 8). When the punch is in the up position, its magnet has negligible effect on the core. When a punch is selected, the permanent magnet moves closer to its associated memory core and saturates it. Each core has single turn primary and secondary windings. The primaries are excited by a common alternating carrier current. The secondaries are connected to peak voltage detectors. In the no-punch position, the core functions as a transformer,

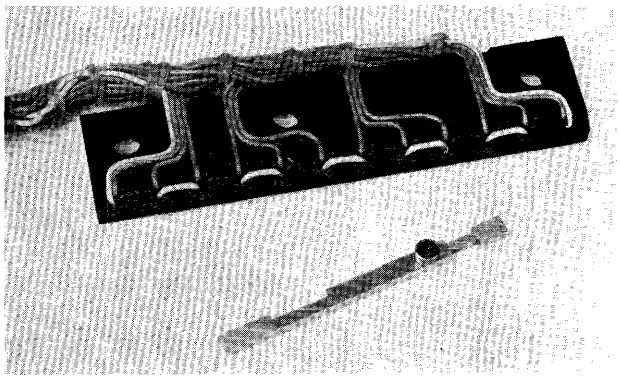


Figure 8. Punch check cores and punch with magnet attached.

coupling the carrier signal from the primary to the secondary. At the bottom of the punch stroke, the permanent magnet saturates the core decreasing the primary signal coupled to the secondary winding. The resulting null in secondary voltage causes the detector circuit to generate a logic signal indicating that the punch has pierced the card.

While the detectors operate continuously, their outputs are sampled only when the punches are in the card. The punch check logic is such that operation of unselected punches, or failure of selected punches creates an error signal. This is an extremely simple punch check system and appears to be the safest alternative to a post-punch reading of the card.

INDEXING

Indexing mechanisms fall into four general groups:

1. Clutched pinch rolls
2. Gated stop units
3. Oscillating pushers
4. Geneva and similar type mechanical indexes

The inherent advantages and disadvantages of these mechanisms can be summed up as follows.

1. *Clutched pinch roll:*
   a) Wearing clutch face, latches and sprocket teeth.
   b) Good control because the tabulating card is always gripped and under control of the rollers.
2. *Gated stop unit:*
   a) Many moving and pivoting parts.
   b) Card edge damage.
   c) Tabulating card can be ejected.

3. *Oscillating pusher:*
   a) Card edge damage.
   b) Lacks positive card control.
4. *Geneva and similar type mechanisms:*
   a) Wear.
   b) Backlash.

Because of the many disadvantages of these and similar devices, a simple servo system was designed using a high-performance moving-coil motor to drive the card pinched between a pair of index and idler rollers. The card can be indexed for punching well within the tolerances specified for the card, or skipped at high speed between columns. The card also can be transported through the punch (for read-only modes) or ejected after punching is complete.

The coincident development of this moving-coil motor was a significant factor in the successful design of the indexing system which can step the card 0.174 in 80 times/sec and skip at 100 in/sec.

The motor used has a torque per ampere constant of 6 oz in/amp and a moment of inertia of 50×10^{-5} oz in sec^2.

Since the load inertia consists only of the card, the steel index and aluminum idler rolls, their connecting shaft and tachometer, an exact inertia match with the motor is possible (Fig. 7).

The elimination of armature iron in the motor gives the low inertia, inductance and high torque-to-inertia and high torque per ampere ratios that provide an ideal marriage between low-power solid state control circuitry and the indexing requirements.

Servo System

The servo system used in the reader-punch to index a card through the punch head is a precision velocity servo as shown in Fig. 9. A velocity servo is used in lieu of a position servo because it performs the indexing task with accuracies well within the specified tolerances without requiring the use of expensive feedback elements such as resolvers, synchros, etc.

Under the control of this velocity servo, a card is moved a distance of 0.174 in (two columns) in 5 msec a maximum of 39 times per card by means of a set of driving rollers, directly coupled to the motor shaft without any intervening belts, gears, etc. Zero backlash and faster response are attainable by this direct drive system.

All timing relationships and synchronization within the punch head are derived from the con-

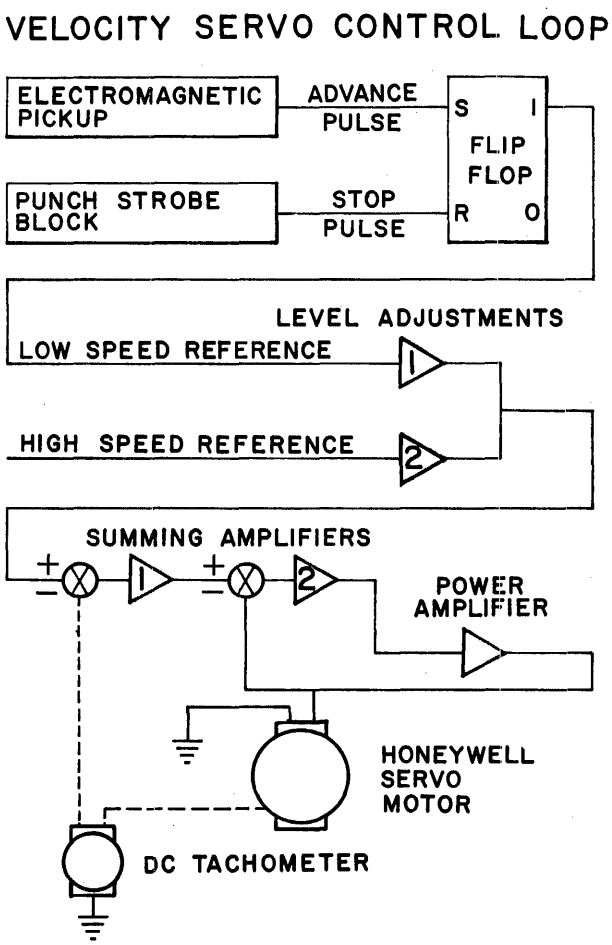


Figure 9. Velocity servo control loop.

tinuous running eccentric drive shaft by an electromagnetic pickup. This pickup generates an advance timing pulse (ATP), the signal to begin advancing from one punch position to the next as well as the punch timing pulse (PTP), used to synchronize selection of the interposers and the punch check timing pulse (ETP). The punch strobe pulse which initiates the stopping of the card is generated from a photo electric sensing block having 40 apertures, one for each of the 40 stop positions. As a card passes under the strobe block, the trailing or registering edge of the card uncovers an aperture, generating the brake pulses to position the card for punching the next two columns. Because of continuous strobing of the trailing edge of the card, each increment and its associated positional tolerance is independent of the previous increment and therefore any error is noncumulative over the length of the card. A card enters the punch station at a speed of 100 in/sec, is picked up by indexing rolls and its speed reduced to 44 in/sec. When the trail-

ing edge of the card is sensed by the first photocell, a brake pulse is generated and the card is registered. Subsequent synchronization between punching and indexing is controlled by the electromagnetic pickup on the continuously running punch eccentric. The timing relationships can be seen in Fig. 10.

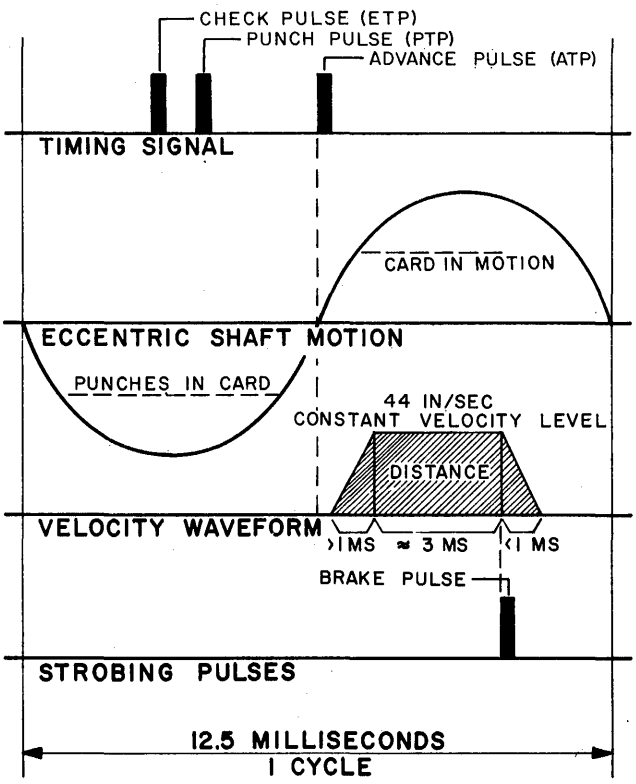


Figure 10. Punch timing diagram.

Punch Strobe

As in the read station, registration of the card for punching is controlled by photocell detection of the card's trailing edge. Strip cells are arranged behind a mask with 40 apertures, and illuminated by the single projection lamp. No lenses are used. As a card approaches the first punching position, all 40 apertures are covered by the card. As the trailing edge of the card uncovers the apertures, a staircase current waveform is generated by the photo cell. Each step of current is amplified and differentiated. The output of the differentiator triggers a multivibrator whose output signals the servo system to stop the card. The card waits until the first column pair has been punched. The servo is then restarted and

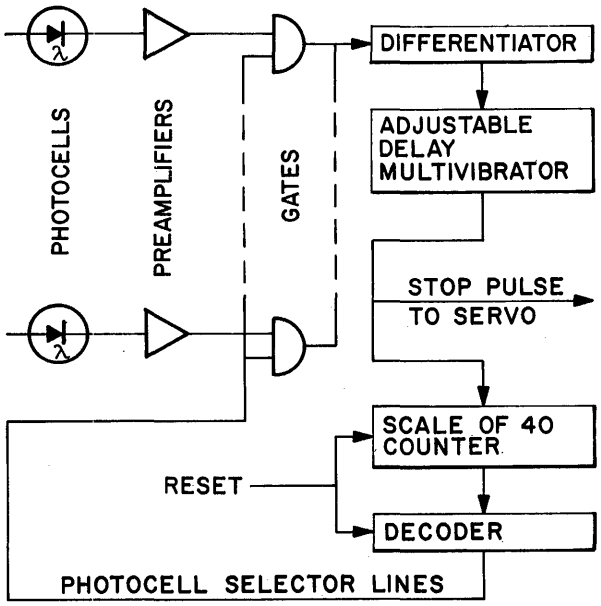


Figure 11. Punch strobe logic.

the card advanced until its trailing edge uncovers the next aperture, and so on (Fig. 11).

Only one cell is connected to the differentiator at any given time. The stop pulses are counted by a scale of 40 counter. Output gates from various stages of the counter select the cell which is connected to the differentiator. In effect, the counter tracks the cards down the row of apertures. Every time the card exposes the last aperture of a cell, the counter gates off the exposed cell and gates on the next cell. Using only one photocell at a time allows the punch to handle cards which have been scored or perforated for tear-off stubs.

An adjustable delay multivibrator between the photo cell circuits and the indexing servo allows the card to travel a short controllable distance beyond the point where the trailing edge is sensed. This electrical vernier provides adjustment of the location of the card relative to the die and eliminates the need for a fine mechanical adjustment such as a lead screw, with its problems of backlash and possible loss of adjustment with vibration.

Servo Control

The way in which the velocity servo controls the card from position to position within the punch head can be followed by referring to the block diagram, Fig. 9, and the series of timing pulses, Fig. 10.

An advance timing pulse triggers a voltage refer-

ence level providing the input to the summing amplifiers. The output of the summing amplifier is fed to the power amplifier which drives the motor. The motor begins accelerating the load to the velocity established by the voltage reference, driving the tachometer which in turn feeds back a negative voltage to the summing amplifier. This voltage is proportional to the rotational speed of the motor. The motor and load accelerate until the feedback voltage of the tachometer is equal to the input voltage reference. At this point, acceleration ceases and velocity remains constant at 44 in/sec. The acceleration from zero to 44 in/sec is accomplished in less than 1 msec. The constant velocity level is maintained for about 3 msec until a strobe-generated stop pulse drops the voltage reference to zero. Consequently, the summing point is negative since the only signal present is the feedback voltage from the tachometer. The power amplifier is now conditioned to decelerate the motor and load to a stop.

Deceleration time is less than 1 msec. The card is now registered within the tolerance required for punching. The total time for indexing and punching is 12.5 msec (80 cps) with 5 msec for indexing and the remainder for punching. In the block diagram, a second summing amplifier and voltage feedback loop are shown. The purpose of this loop is to achieve the proper gain and reduce the servo "dead zone" for optimum response. The constant velocity level is of extreme importance in this servo system because positional accuracy depends upon the motor and load braking from the same velocity for each increment. The specified tolerance for punch hole registration relative to the trailing edge is ± 0.007 in.

The two-column punch design permits this velocity level to be considerably lower than that which would be required of a single-column punch with the same throughput. An additional advantage is found in the reduced wear on component parts.

In the voltage reference level area of the block diagram (Fig. 9) a high- and low-speed level distinction is indicated. In the above explanation, reference to either input voltage or constant velocity relates only to low speed, 44 in/sec. The high speed is used for ejection when punching is complete and in a read-only mode, for transporting cards at 400 cards per minute without indexing. It is also used in the high-speed skip feature described below.

High-Speed Skip Feature

This particular feature is unique in that it permits high speed skipping between columns while punching. Skipping is performed automatically without requiring any special program routine and is accomplished by two logical functions—searching for blanks (SFB) and blanks found (BF). The control logic sends out two pulses, one for each column to be punched, requesting data and simultaneously checking for blank columns. As long as blanks are found in both columns the card moves at high speed. When blanks are no longer signaled the indexing system immediately drops from the high speed to the low speed and resumes normal indexing. Registration after a high-speed skip is maintained with the same degree of accuracy as a single increment because the low speed of 44 in/sec is resumed two columns prior to braking.

CONCLUSION

This paper has described a card reader-punch designed around mechanical components of proven reliability. The use of electrical functions to replace troublesome mechanical operations has resulted in a much simpler mechanism than found in conventional machines of this type. Additional advantages are increased operator convenience and reduced maintenance requirements.

ACKNOWLEDGMENTS

The authors wish to acknowledge the contribution of the development team whose talents have made this effort successful, in particular Messrs A. B. Ragozzino, R. Carman, C. H. Wang, P. Nelson and J. Rae, and E. G. Hazle for his help in the preparation of materials for the present paper.

THE IBM 2560 MULTI-FUNCTION CARD MACHINE

Chester E. Spurrier
IBM SDD Development Laboratory
Rochester, Minnesota

INTRODUCTION

The IBM 2560 Multi-Function Card Machine (MFCM) provides the System/360 Model 20 with a unique and versatile input/output capability. It combines the facilities of a card reader, card punch, collator, interpreter, and card document printer, all under control of the Model 20 processor. (See Fig. 1.) Two card hoppers, an optical read station that reads both primary and secondary cards, a common punch station, an optional printing station, and five selective radial stackers provide the Model 20 system with a card handling capacity never before possible with one pass of the cards.

HISTORY

Prior to the development of the MFCM, approaches to peripheral card handling equipment generally assumed a separate machine for each of the functions of reading, printing, punching, collating, and selecting cards. Combinations of some of these five functions had been successfully achieved, for example, in the collator-reproducer of Remington Rand, the IBM 101 statistical sorter, the IBM 1402 and 1622 Reader Punches with their multiple hoppers and stackers, and the Sperry Rand 1001 with its reading-sorting-collating capability. However, not until the development of the MFCM did any machine combine all five functions into one unit and thus bring a "system" philosophy to card handling.

GENERAL DESCRIPTION

The MFCM reads 500 cards per minute and punches at a rate of 160 columns per second. The card throughput during punching is a function of the number of columns punched or spaced, beginning with column 1. It varies from a minimum of 91 cards per minute for 80 columns punched to a maximum of approximately 340 cards per minute for 1 column punched. With the optional print feature installed, the machine can print 138 characters per second per line. Throughput when printing is also a function of the number of columns printed and varies from a minimum of 99 cards per minute when printing 64 characters to a maximum of 397 cards per minute when printing 1 character. Since the Model 20 operates in a time-sharing mode, the machine prints and punches simultaneously and thus only the operation requiring the greatest amount of time limits the throughput of the machine.

All the logic circuitry for controlling the MFCM is contained in the processor. The only electronic circuits located inside the MFCM are the magnet drivers, solar cell amplifiers, and pulse shapers needed for proper operation. All timing pulses needed by the processor, except those required for reading, are created by 13 magnetic sensing coils

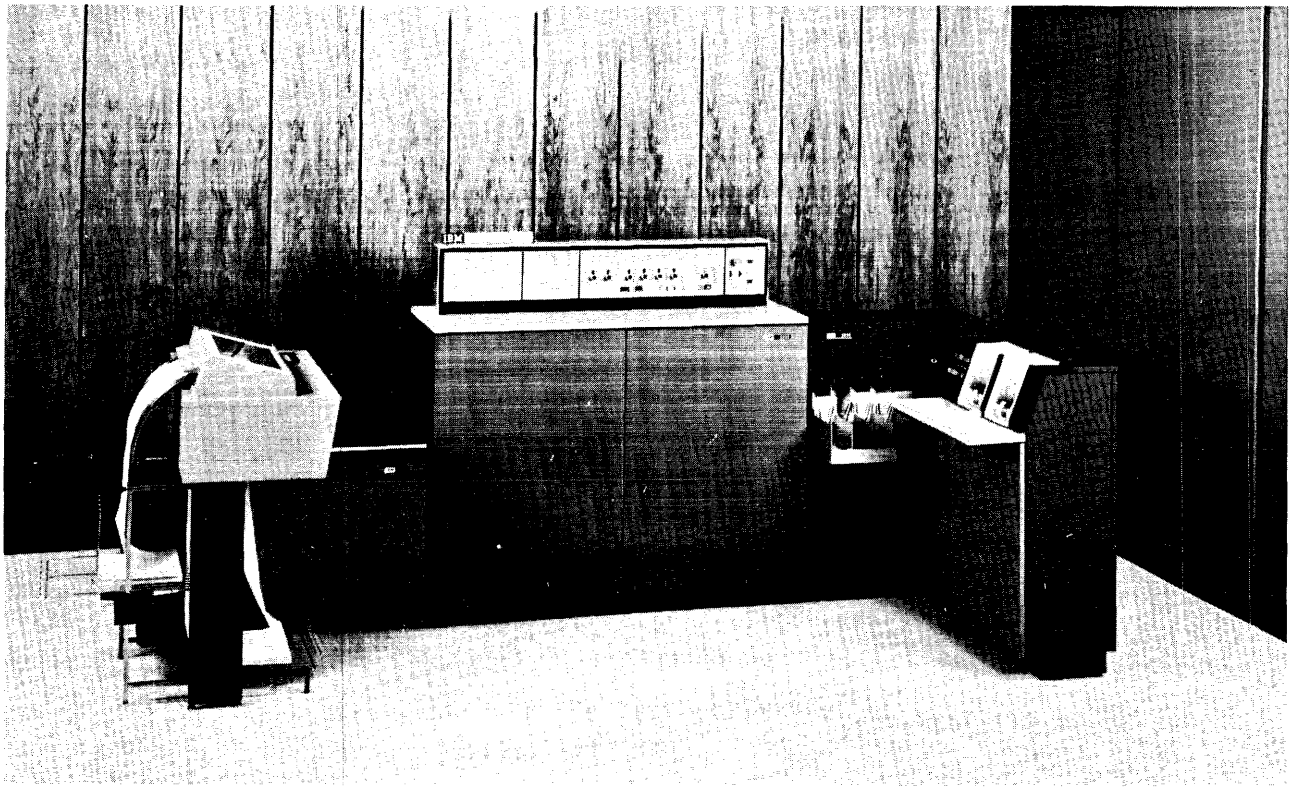


Figure 1. IBM System-360 Model 20 showing the IBM 2203 Printer, the IBM 2020 Processor, and the IBM 2560 Multi-Function Card Machine.

which detect the motion of magnets embedded in five separate timing discs. Card movement and location are monitored by nine solar cell sensors, rather than the more conventional mechanical card levers.

The lower half of the MFCM contains two gates of electronics, power supplies, and the chip box. The upper half contains the card handling mechanism (see Fig. 2). An unusual feature of the machine is the so-called "backbone" or vertically mounted plate on which are mounted most machine components. This backbone feature greatly improves accessibility of the card path and components over the more conventional side frame or "boxed-in" approach. The two card feed hoppers are mounted on the front or operator side of the backbone. The stacker assembly, which consists of a magnetic selector unit and five radial stackers, is also located on the operator side. All other components are mounted on the rear side of the backbone, including the entire serial card path, the read, punch, and print units, and the drive motor. The motor drives all units through toothed belts and pulleys.

CARD FEEDING

Card motion during reading or ejecting in the MFCM is controlled by the card feed clutch. From each parallel hopper the cards are fed into a cornering station by picker knives and continuously running feed rolls, as shown in Fig. 3. From the cornering station they move serially through the read, punch, and print stations before cornering again to move in a parallel path into one of the five stackers. The movement from the hopper through these operational stations is under complete processor control and is carried out by either cam operated or magnetically operated feed roll selection devices and the feed clutch. Cards are designated as primary or secondary according to the hopper from which they are fed. During a typical operation, one of each type of card is always in position to enter the read station, one of each type of card is always in position to enter the punch station and a single card is in position in the print station. As the card in the print station passes through the station to go on to the stacker, either

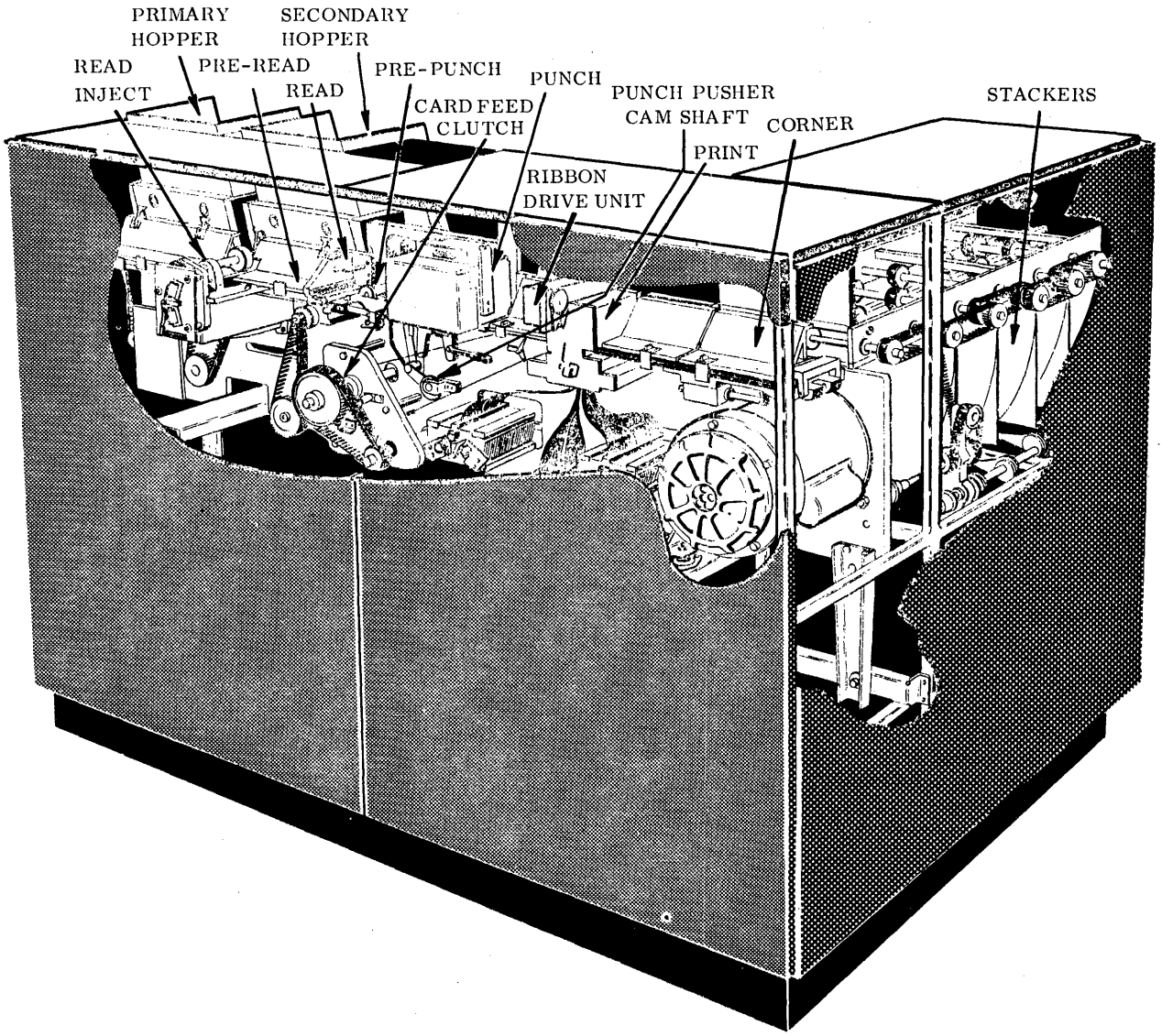


Figure 2. Rear view of the MFCM showing location of major components.

the primary or secondary card at the pre-punch station is fed through the punch unit to the print station and the corresponding primary or secondary card at the pre-read station is fed through the read unit to the pre-punch station. At the same time a card is being fed from the corresponding hopper to the pre-read station. All of this card movement occurs simultaneously during one clutch cycle, and thus all card stations are occupied on every cycle. Under processor control, cards from one hopper can be held indefinitely at the pre-read and pre-punch stations while cards from the other hopper are processed through the machine.

READING SYSTEM

A magnetically operated pressure roll and the continuously running read inject roll grip the card and feed it into the read station from its pre-read position. The card is accelerated up to the reading velocity of 146 inches per second, which matches the peripheral velocity of the continuously running

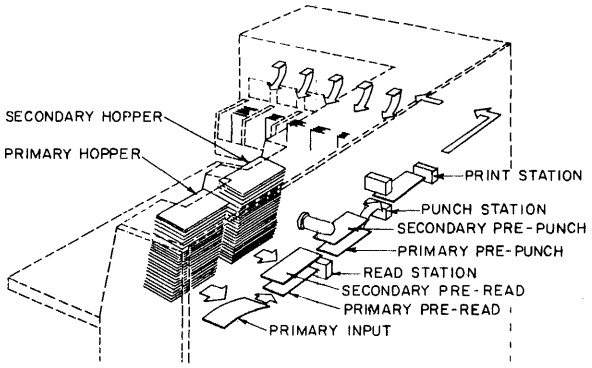


Figure 3. Schematic of the MFCM card path.

read feed roll. As the leading edge of the card is gripped by the read feed roll, the trailing edge leaves the inject roll so that the card is under the control of the read feed roll for the entire read operation.

The holes are sensed by 12 silicon solar cells located below the card path. Light is provided by 12 lens tip lamps located above the card path. The lamps are individually adjustable for position and intensity. Both the lamps and the cells are mounted inside sealed assemblies behind glass plates to prevent card dust accumulation from affecting light transmission.

During a read operation, the CPU must be supplied with a series of pulses synchronized with card motion and timed so as to indicate the optimum time for sampling the output of the read circuits for each of the 80 card columns. The device which accomplishes this in the MFCM is the magnetic read emitter. A schematic of the emitter is shown in Fig. 4. The rotor or drum of the emitter is mounted on the read feed roll shaft. The periphery of the drum is plated with a thin, hard coating of cobalt. The stationary housing of the emitter contains 60 heads or probes equally spaced around the periphery of the drum. The housing also contains a write coil, a read coil, and an erase coil.

As the leading edge of the card enters the read station, the trailing (or column 80) edge of the card uncovers a solar cell. The cell is located so that it becomes uncovered as column "zero" is directly over the read cells. The uncovering of the trailing edge cell causes the CPU to send a signal to the emitter write driver. This results in 60 magnetic bits being recorded on the drum due to the flux generated by the single write coil passing through the 60 heads. The read feed roll on the drum shaft feeds a card one column in 1/60 of a revolution so

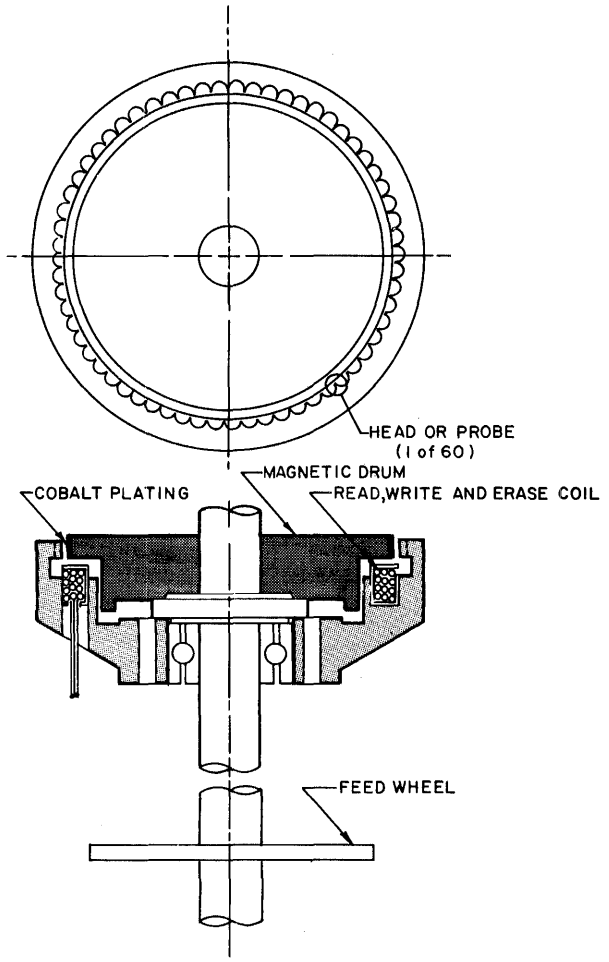


Figure 4. Schematic of the Magnetic Read Emitter.

that as each column is passing under the read station, the 60 magnetic bits on the drum have rotated one position and are being sensed simultaneously by the 60 heads. The cumulative flux change of the 60 heads is sensed by the common read coil. Any air gap variation between the drum and the heads caused by eccentricity is of little consequence since the output signal of the emitter is the sum of the 60 signals. After the card has left the read station, the erase coil is energized and the 60 bits are erased so that the emitter may be resynchronized with the next card as the card is being read.

PUNCHING

To register a card for punching, the punch pusher clutch and the desired path selection solenoid is energized. The primary or secondary card selected is pushed into column 1 registration and the incre-

mental pressure rolls are closed. Command from the processor controls the number of columns punched and incremented.

The chief difficulty in designing a high-speed serial punch is in maintaining accurate punching registration over 80 columns of movement. An error of only 0.001 inch in column to column spacing would result in the punching being about one full column out of registration after 80 spaces. The basic problem is the high card acceleration necessary to achieve desired throughput. The MFCM solves this problem in two ways: 1) by keeping the acceleration as low as possible, and 2) by using feed rolls with a very high coefficient of friction to minimize card slippage. The punch unit is designed to allow approximately three-fourths of the 6.25-millisecond punch cycle for card movement, thus minimizing the acceleration requirements. The unit utilizes three cams: one to drive the punch through the card, one for positively restoring the punch, and one for moving the interposer-armature assembly, as shown in Fig. 5. By optimizing each of these motions, the time the punch is in the card has been reduced to approximately one-fourth of the punch cycle.

The punch mechanism is controlled by a magnet unit utilizing the so-called "no-work" principle. The magnets are only required to prevent the armature from moving rather than causing them to move; thus, the origin of the term "no-work" magnets. Several advantages are realized by using this style of magnet unit. Since reluctance of the magnetic circuit is low, the coil and core size may be reduced, resulting in a much more compact unit. Also, magnet driving requirements are lowered and heat dissipation is reduced.

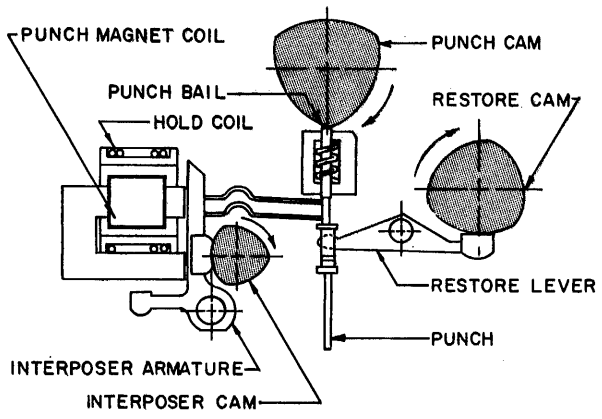


Figure 5. Cross section of the punch mechanism.

A constantly energized single hold coil surrounding all 12 punch magnets prevents the armature from following the interposer cam. When a punch is to be operated, the appropriate punch magnet coil is energized. This cancels the hold flux, which releases the armature at the high point of the interposer cam, and the armature spring causes the armature to follow the cam. This moves the interposer between the punch bail and the punch, causing the punch to be driven through the card. The restore lever pulls the punch out of the card.

CARD DOCUMENT PRINTER

Basic requirements for the MFCM included a card document printer that would provide an output of 100 fully printed cards per minute, with up to six lines per card. Existing printing mechanisms were capable of such output, but at an unreasonable cost, so an entirely new print unit was developed. The desired output is achieved by using a wire matrix print head for each print line and printing in the serial mode. These wire matrix print heads may be positioned manually by the operator in any of the 25 printing row locations from the top to the bottom of the card to accommodate any card format.

Each of the printed lines may contain up to 64 characters spaced at 10 characters per inch. A wire matrix five wires wide by seven wires high forms the characters. The wires are driven against an inked ribbon, card, and platen to print the character on the card.

One matrix of 35 wires, hereafter called a print head, prints a complete line on a card. To print six lines on a card, six print heads are required. Print heads are installed in groups of two to the maximum of six. An incremental drive moves the card from position to position, stopping the card in each position. Each print head prints a character on the card in a 7.23-millisecond cycle, which achieves the rated speed of 138 characters per second per line, or 828 characters per second for six print heads.

A card ejected from the punch station is stopped in print position 1 registration by a magnet-operated print gate. During the first print cycle the card is gripped by incrementing rolls. An emitter on the incremental drive triggers each print cycle, which consists of setting up the characters for each print head, printing all of the characters in the position, and moving the card to the next column. Upon

completion of printing, the card is ejected from the print station during the following feed cycle.

A ribbon is continuously driven past all the print heads at an average speed of 7 inches per second. A spring-loaded reversing mechanism assists the ribbon-drive motor in achieving rapid ribbon reversal.

Print Mechanism

Figure 6 is a cross section of the print unit. Each print armature is fastened to a print wire. A print unit contains 70 wire and armature assemblies or two heads. A tube, which is securely fastened at

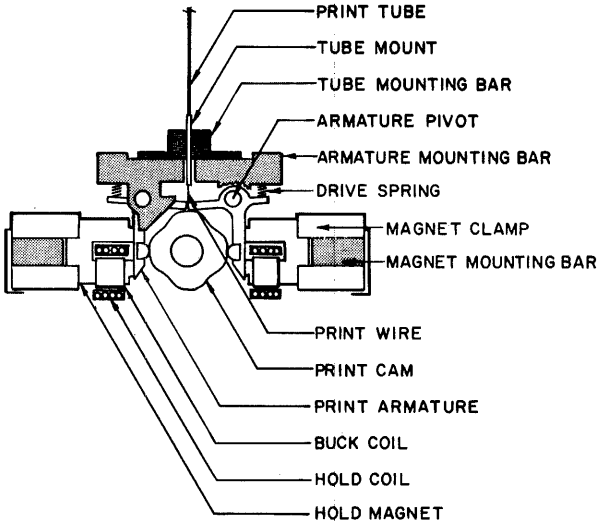


Figure 6. Cross section of the print unit.

the unit and at the print head, guides each wire. A compression spring drives each armature and wire against the ribbon, card, and platen to perform the printing. The print armatures are controlled by no-work magnet units similar to those used in the punch unit. Characters are formed by energizing the required buck coils. The selected armatures drop onto the print cam which controls the time of impact of the wires, and restores the armatures to the magnet after printing. When not printing, the armatures are held away from the print cam by the hold magnets.

Character Selection

The printing circuitry in the MFCM consists of an array of 35 magnet drivers, each of which is connected, through isolation diodes, to the six buck coils associated with the same matrix position on all six heads. The common side of the 35 coils for each head are connected to the power supply through a separate silicon-controlled rectifier. This circuitry allows the 210 magnets required for six print heads to be controlled by only 35 drivers, since the six heads are set up sequentially rather than simultaneously.

Two things make sequential operation possible: 1) a memory dwell built into the print cam, and 2) a motionless period for the card which is long enough to allow the three print cams to be timed for sequential operation.

Figure 7 shows the sequential timing and the cam profile for the three print unit cams. At high dwell on the print cam, the armatures have been restored from the previous cycle and are ready for setting up in the next print cycle. Setup begins when a pulse arrives from an emitter on the incremental drive, timed with high dwell on the print cam for print unit one. The selected buck coils for print head one are energized for 350 microseconds, during which time their armatures are released. The armatures drop to the memory dwell which is an intermediate dwell on the print cam just following the high dwell. After the coils are de-energized and the holding flux increases, the armatures are not reattracted back to the magnets due to the air gap created by the memory dwell. After the 350-microsecond buck pulse for head one, there is a 100-microsecond delay before head two is set up to allow the silicon-controlled rectifier for head one to turn off. Head two is then set up in a similar manner. The second print unit, which controls head three and four, is timed so that the setup for its two heads comes just after the print circuitry has completed the setting up of heads one and two. The third print unit is similarly timed with respect to the second. As a result of this method of timing,

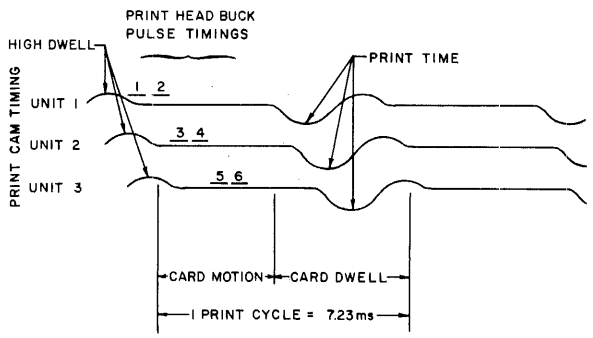


Figure 7. Electrical and mechanical timing diagram of the print unit.

the impact from heads one and two occurs just after the card has come to rest in the dwell period, heads three and four print in the center of the dwell period, and heads five and six print just before the card starts to move.

CONCLUSIONS

The challenge of designing a machine that could, in a single pass, perform most of the functions required in a card-oriented processing system, has been met by the IBM 2560 Multi-Function Card Machine.

This machine, as a part of System/360 Model 20, provides a dramatic reduction in the number of processing steps and the card processing time required by conventional punched card equipment. File maintenance or file updating is now possible with one machine. The unique concept of the MFCM has been likened to a "poor man's magnetic tape system." In this concept the two card hoppers simulate two magnetic tape input drives, the five card stackers simulate magnetic tape output drives, and the coordinated read-punch-print units simulate the read-write-copy function of magnetic tape.

The wide acceptance and interest in the MFCM, since its announcement, is an indication of the basic soundness of the multi-function concept in card handling.

A NEW DEVELOPMENT IN THE TRANSMISSION, STORAGE AND CONVERSION OF DIGITAL DATA

R. P. Burr, John J. Rheinhold
Photocircuits Corporation, Glen Cove, New York
and
Roy K. Andres
RCA Communications, Inc., New York, New York

INTRODUCTION

There is a present trend in data processing systems toward the decentralization of computer systems as exemplified by time-sharing and related techniques. A consequence of this development is a requirement for digital communication systems capable of operating reliably at comparatively high speeds. One of the oldest known forms of such communication is the printing telegraph which operates at the lower end of the speed spectrum at rates ranging from 50 to 100 baud. There is some evidence to suggest that for routine digital communications purposes, particularly in the time-sharing area, the economic maximum for a few years into the future will lie in the vicinity of 1000 to 2000 baud and that the volume of traffic at these rates will rapidly increase. Present practice is generally to transmit data at such rates in "real time" only over communication paths which are not expected to fail during the transmission period. In the future, however, it is a virtual certainty that low-cost buffering devices will be required having the speed, versatility and storage capacity to handle system requirements at rates up to 4800 baud or more.

The use of magnetic tape in preference to paper tape offers a preferred solution to the problem of high-speed buffering, provided that a technique can be devised which permits the use of magnetic tape so that it is in all operational respects equivalent to paper tape systems. Such a functional equivalence can be obtained if it is possible to: 1) record on the tape incrementally over a speed range extending to at least 300 characters per second on a character-by-character or bit-by-bit basis and 2) read the information recorded on the tape in an incremental manner through the same speed range.

A device having these characteristics would be functionally equivalent to a paper tape system capable of operating at punching and reading rates up to 300 characters per second but would of course be physically smaller and mechanically simpler because of the improved packing densities available on magnetic tape.

To additionally exploit the full advantages of magnetic tape in such an environment the device should also be capable or recording and reproducing digital information "on the fly" in a manner somewhat analogous to a conventional digital magnetic tape transport so that very high acceptance and transmission rates could be achieved if necessary.

A capstan drive system for magnetic tape which is capable of satisfying these operating requirements can be achieved by the proper combination of a fast

response, smooth torque DC motor operating in combination with a wideband velocity servo amplifier and logic control circuits. An assembly of this type is known as a "random incrementer" and can provide a high degree of flexibility in the control of tape motion when either recording or reading digital information.

In the following discussion the operation principles of the random incrementer will be briefly reviewed while two particular applications of the device in the communications environment will be described in sufficient detail to illustrate the versatility afforded by the technique.

THE RANDOM INCREMENTER

A simplified block diagram of the basic components in a random incrementer system are shown in Fig. 1. For a detailed description of the system operation the reader is referred to the paper by Burr, Rheinhold and Andres.[1] As pointed out in this reference the characteristics of the drive motor

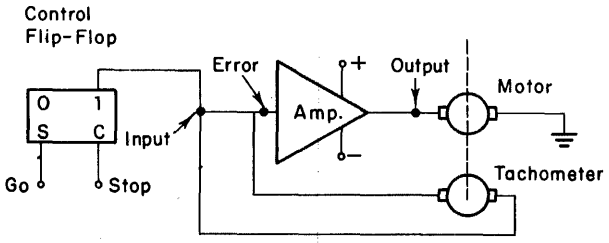


Figure 1.

utilized for the application are critical to the proper functioning of the drive system. Specifically, the motor must be a device having a small mechanical inertia, a negligible electrical time constant, essentially no preferred positions and the ability to deliver substantial pulse torques. A printed circuit motor is a component which is ideally compatible with these requirements and provides excellent performance in the system. (See Fig. 2).

To review briefly, the essential features of the incrementer system operation are shown in the waveform diagrams of Figs. 3 and 4, and depend upon the fact that that portion of the circuit to the right of the point marked "input" on Fig. 1 is basically a velocity servomechanism of the type generally known as a regulator. Such a system has the property that when a fixed DC voltage is applied to the "input" terminal the action of the amplifier and the tachometer is such as to accelerate the motor to a speed such that the difference or error between the input voltage and the tachometer voltage is just sufficient to provide the necessary driving voltage via the amplifier to the motor. Systems of this type are widely used in the industry for speed control applications of all sorts and are almost invariably operated in a "nonsaturated" mode, which is to say that the rate of change of the input signal command is restrained to a value which can at all times be satisfied by the available acceleration capability of the system.

In the present instance, however, the random incrementer action is obtained by operating the servo

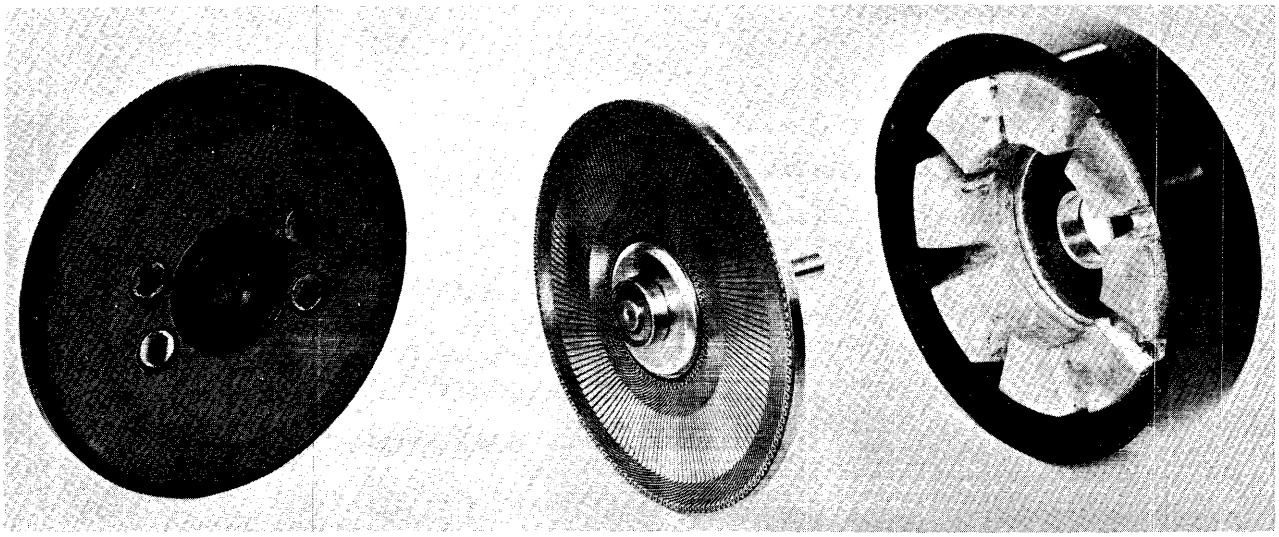


Figure 2.

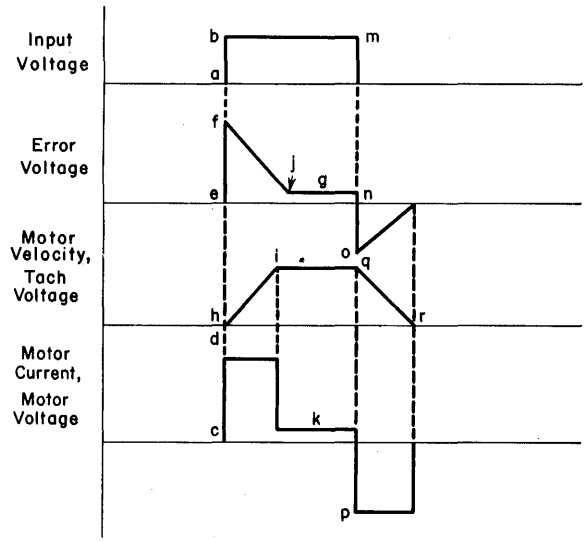


Figure 3.

in a saturated condition for acceleration or deceleration of the capstan. As will be seen from the waveforms of Fig. 3, the system is put in motion by the application of a voltage step from the control flip-flop to the servo input terminal. The magnitude of the final velocity is proportional to the magnitude of the voltage step. The servo system is designed so that it remains in saturation until the specified velocity is reached. Therefore, maximum possible acceleration is obtained from the capstan motor during the starting transient interval. Correspondingly, when the input command is removed by clearing the flip-flop, the action of the servo is such to apply the maximum possible acceleration to the motor during the stopping transient until zero velocity is reached.

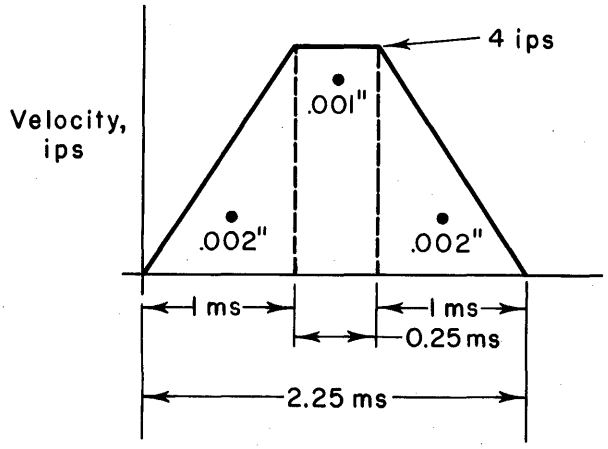


Figure 4.

The shape of a typical velocity increment is shown in Fig. 4. The distances in this diagram are referred to the periphery of an optimized tape drive capstan[2] mounted on the motor shaft and therefore to the magnetic tape itself. For purposes of the present discussion the point of greatest importance is that the transient accelerations available in this mode of operation are sufficiently high so that the capstan (and tape) attain significant velocities in very small displacements and in comparatively short times. In Fig. 4, for example, the tape achieves a stable velocity of 4 inches per second in a distance of 0.002". The total motion time is 2.25 milliseconds and the total motion displacement is 0.005".

INCREMENTAL READ BACK

In Fig. 4, the beginning of the stopping transient or stopping displacement of the incremental drive occurs at the instant that the control flip-flop in Fig. 1 is cleared. It will be noted from Fig. 4 that the stopping displacement for this particular case is 0.002". As a result the incremental drive may be utilized to read back a record on magnetic tape having a packing density such that the distance between characters is greater than or equal to 0.004". The apparatus arrangement for accomplishing this is shown in Fig. 5. For an understanding of the details of the operation it is immaterial as to whether a single or multiple track recording is used on the tape: the only requirement is that for each and every character or frame there be a flux transition of some type which can be interpreted as a "read."

It will be noted from Fig. 5 that the clearing terminal of the flip-flop is connected through suitable amplification to the magnetic head output so that whenever a read is obtained from the tape the control flip-flop will be cleared. In order to read the tape one character at a time it is therefore only necessary to apply "advance-and-read" pulses to the go or set terminal of the control flip-flop. The effect of so doing will be to put the motor in motion, i.e., initiate the starting transient. The motor velocity will then rise to the limiting value and the tape will move 0.002" in this process. If the *character spacing is 0.005"*, the read or stop signal will be received at the right-hand upper corner of the waveform in Fig. 4 at a distance of 0.003" from the starting point. The motor will then decelerate to a stop allowing another 0.002" of tape to pass over the head so that the final stopping point will be 0.002" after the previous character and 0.003" before the next character.

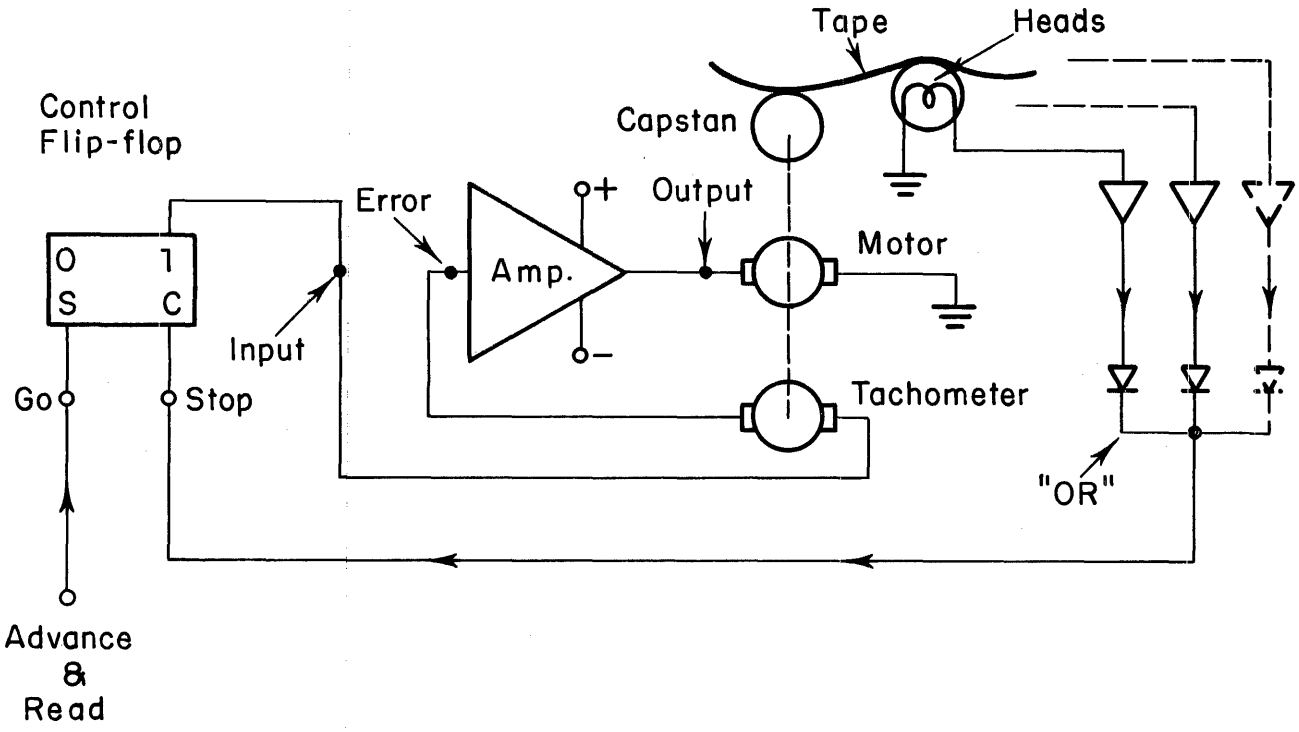


Figure 5.

The maximum rate at which information can be recovered from the tape while retaining the ability to stop before the next character, which is a limiting condition on the incremental reading,[1] would be 800 characters per second in the foregoing situation since the steady state velocity is 4 inches per second and the packing density is 200 characters per inch. The maximum permissible packing density in this case is that which results in a distance of 2 × 0.002″ or 0.004″ between characters or 250 characters per inch. The maximum information rate would then be 4 × 250 or 1000 characters per second.

It is evident that higher operating rates are possible, subject to a relaxation of the restriction that the stop distance must be less than half the distance between characters.

INCREMENTAL RECORDING

Two possible arrangements of the random incrementer for recording are shown in Figs. 6 and 7, the difference between the two schemes being principally in the accuracy with which the recorded bit spacing is generated.

In Fig. 6 the incoming information is recorded on the tape (shown in this instance for parallel format) when a pulse is received from the optical tachometer or code wheel attached to the shaft of the motor. Recording is therefore made "on the fly" in exact space synchronism with the output from the optical pick-off so that excellent accuracy is obtainable. The accuracy of bit spacing achieved by this method will be essentially equivalent to the accuracy of the divisions on the optical tachometer degraded only by any tendency for the tape to slip off the capstan or by stretching between the capstan and the recording head. Experience indicates that at the low velocities involved these latter effects are negligible.

In Fig. 7 the spacing of the recorded data is basically determined by a single shot pulse generator which energizes the incrementer for a fixed time period. The displacement accuracy of the record is therefore affected to some extent by tension and friction "noise" in the system. However, it is most important to note that if the resulting recording is to be read by another incrementer system, the presence of displacement inaccuracies is unimportant so long as the minimum bit spacing does not fall below a figure equal to the sum of the starting and stopping displacements on the reading incrementer.

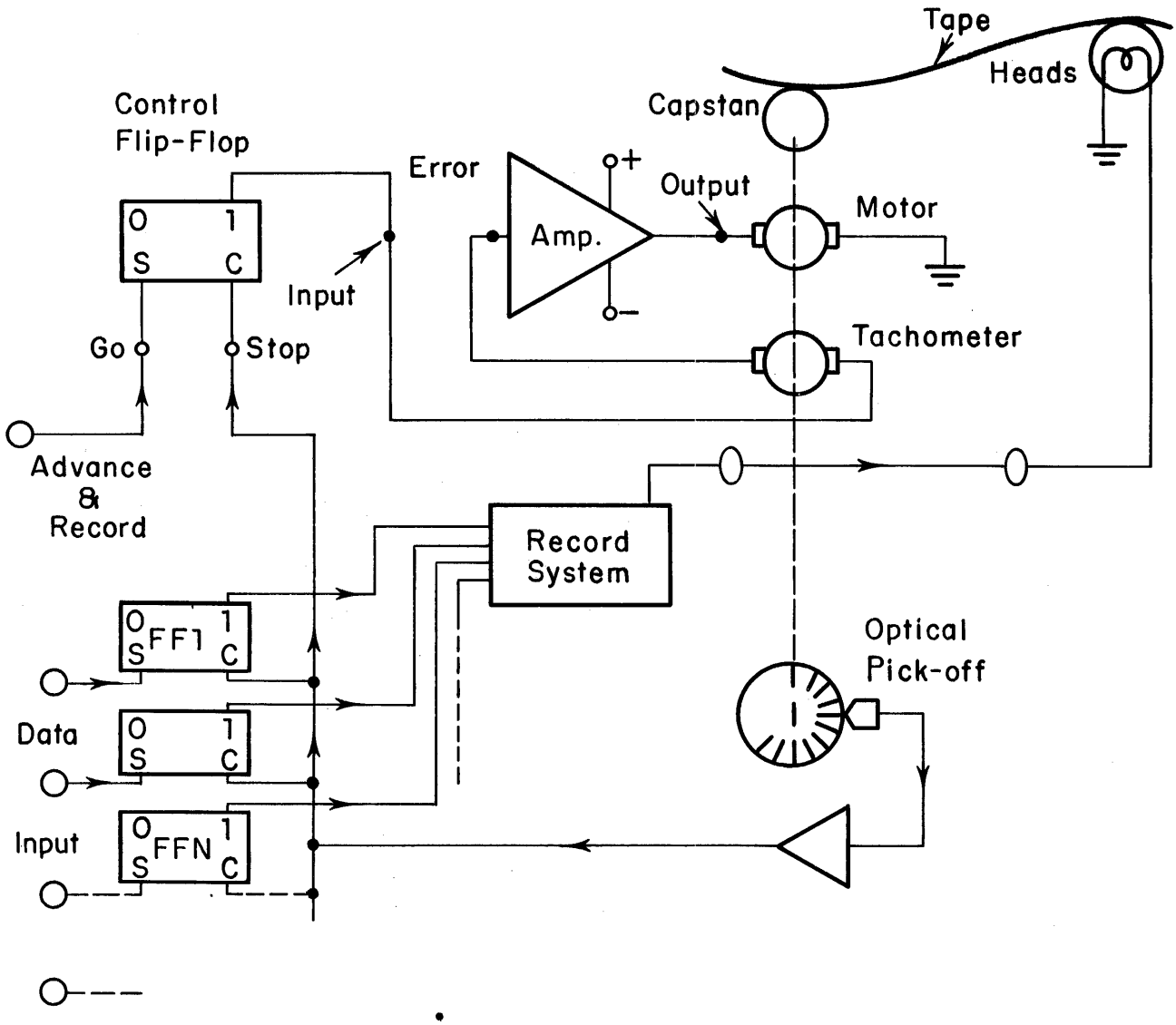


Figure 6.

APPLICATIONS OF THE INCREMENTER SYSTEM

A more general picture of the flexibility of the random incrementer tape transport can be obtained from the following discussion of two specific application areas which range in complexity from the simple to the complex. The first of these is a simple and inexpensive serial format data collection device which is well suited to the slow-speed recording and "batch" transmission of information. The second application is a considerably more versatile "parallel format" dual-capstan machine having a wide speed range and directed specifically towards buffering applications involving data rates up to approximately 10,000 baud.

SERIAL FORMAT DATA COLLECTOR

The serial format data collection device is designed to accept a serial data stream of the type emitted by a printing telegraph machine or serial paper tape reader, to record this information serially by bit and by character on one quarter inch wide magnetic tape, and to read back the data either in the same format and speed as received or at rates up to 2400 baud. The higher speed output can be

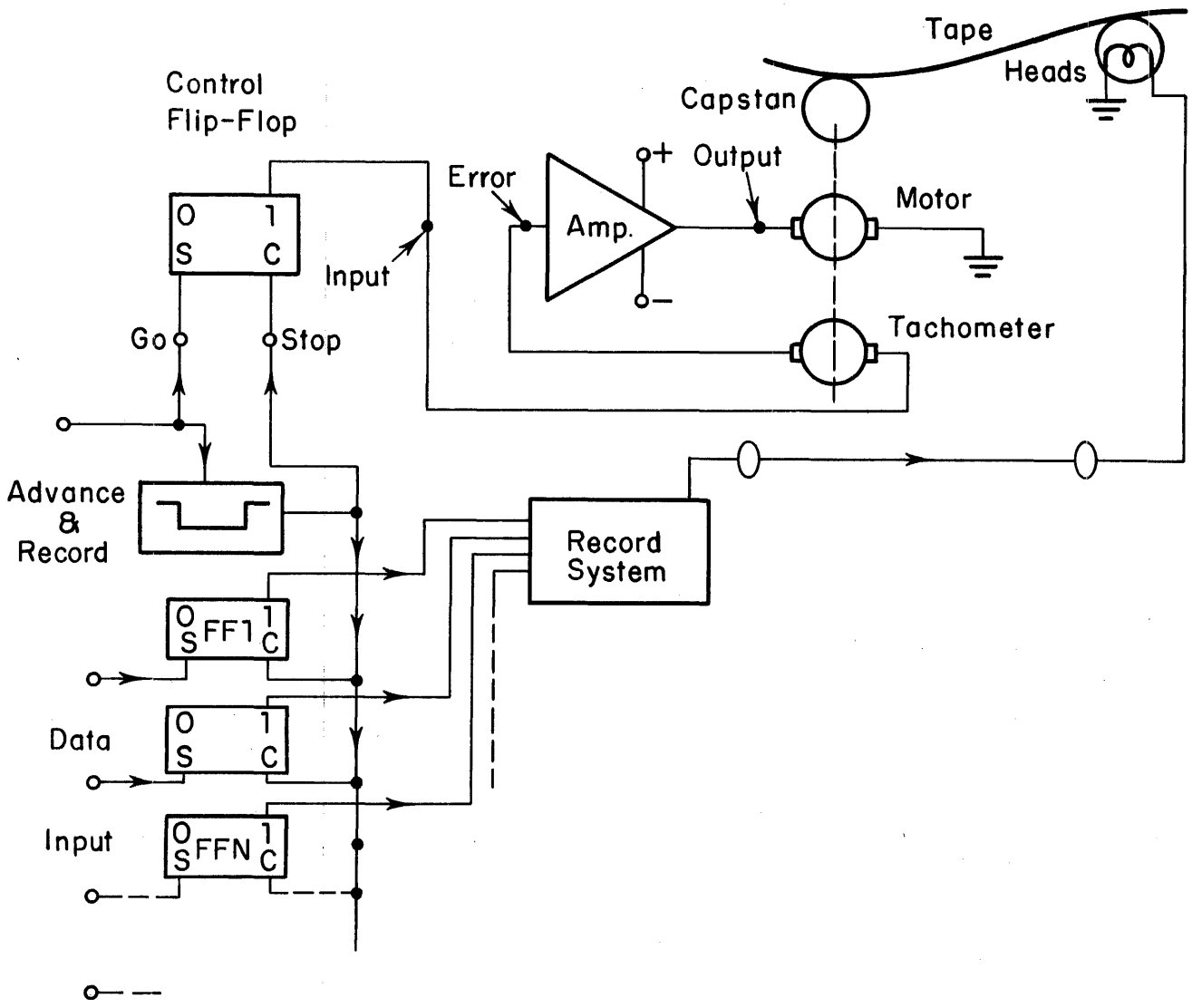


Figure 7.

either free running or synchronous with an external clock.

The format used on the tape is shown in Fig. 8. Although there are several alternative possibilities this particular case is of special interest because the "unrecorded" tape, shown in Fig. 8a, carries a "prerecorded" clock pulse on one of the two available channels. A sketch of a complete recording is shown in Fig. 8b. The "clock" is on the upper track and the data is on the lower. It will be noted that the data bits are recorded in space register with the timing bits. The magnetic head used in the system is a low-cost "two-track stereo" record/reproduce head of the type used in home entertainment tape recorders.

A block diagram of the electronic system used for recording is given in Fig. 9. It will be seen that the function of the prerecorded clock bits is equivalent to that of the optical pick-off in Fig. 6. When recording, the serial data stream enters a conventional synchronizing interface so that the character "start" or "sync" pulse is recognized and a local timing oscillator is phased with the signal. Each successive local oscillator pulse samples each data pulse following the start pulse, places the data in a one-bit storage flip-flop, and simultaneously starts the incrementing tape drive. When the corresponding "stop" signal is received from the recorded clock track, the data is recorded with an RB pulse through the data head winding, the one-bit storage flip-flop

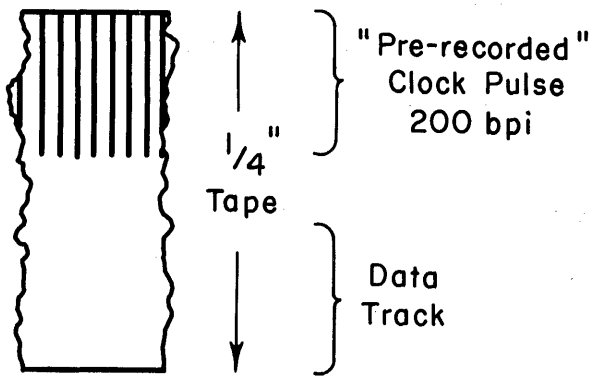


Fig. 8A

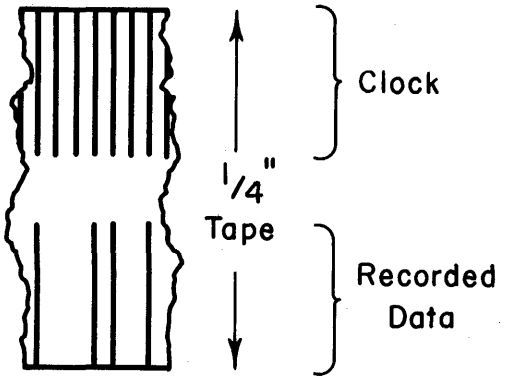


Fig. 8B

Figure 8.

is cleared and the system waits for the next bit of the character to be delivered. This process is repeated through the entire character, so that the tape is *incremented* step-by-step for each bit in sequence. As a result, for example, seven motions are made for the Baudot code and eleven for the ASCII code. When the character is completed, the system rests until the next start pulse appears on the line. The maximum rate at which this simple system will accept serial data for recording in this manner is theoretically 800 bits per second and practically 600 bits per second for an incrementer using a type PM-368 printed motor. This is, of course, far in excess of conventional printing telegraph rates. The local timing oscillator and counter chain must conform to the particular signal bit-rate and format.

The recorded packing density is 200 bpi. A standard home-entertainment EIA cartridge of 300 foot capacity will therefore accomodate approximately 60,000 characters in the serial format.

Readout can be accomplished with varying levels of sophistication depending upon the desired result.

In the simplest case the incrementer motor is run continuously by a constant DC command. The electronics for this are shown in Fig. 10. The operating speed of the tape is adjusted by setting the level of the "go" command from the motion control flip-flop to the desired level. The recorded signals are recovered from the head, amplified and limited in a threshold decision circuit to obtain "clean" pulses. The clock track and data track signals are then applied to a coincidence logic and flip-flop

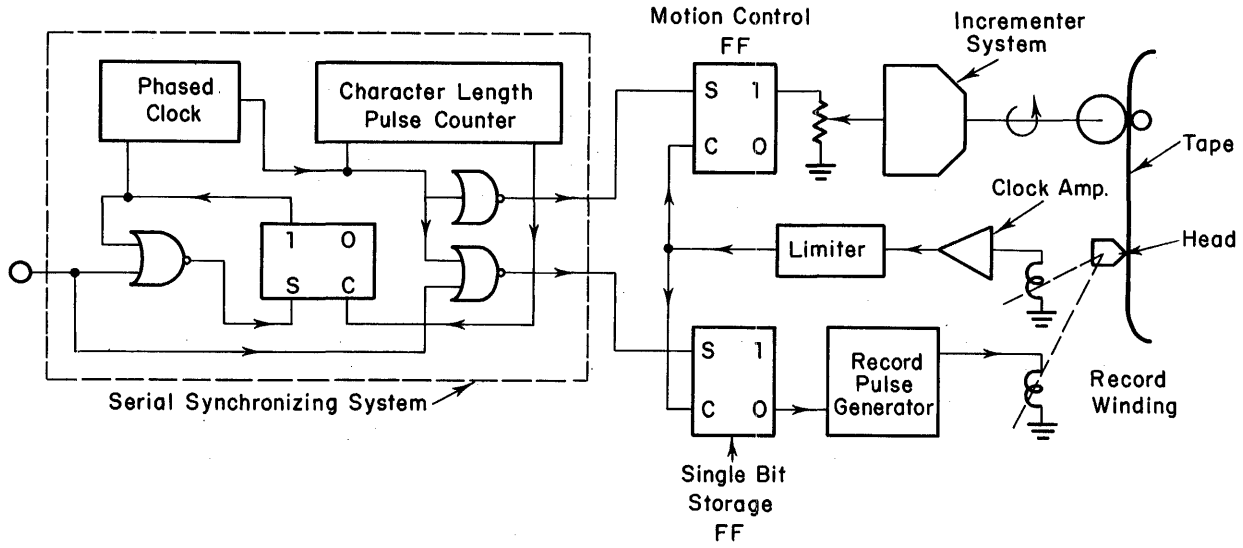


Figure 9.

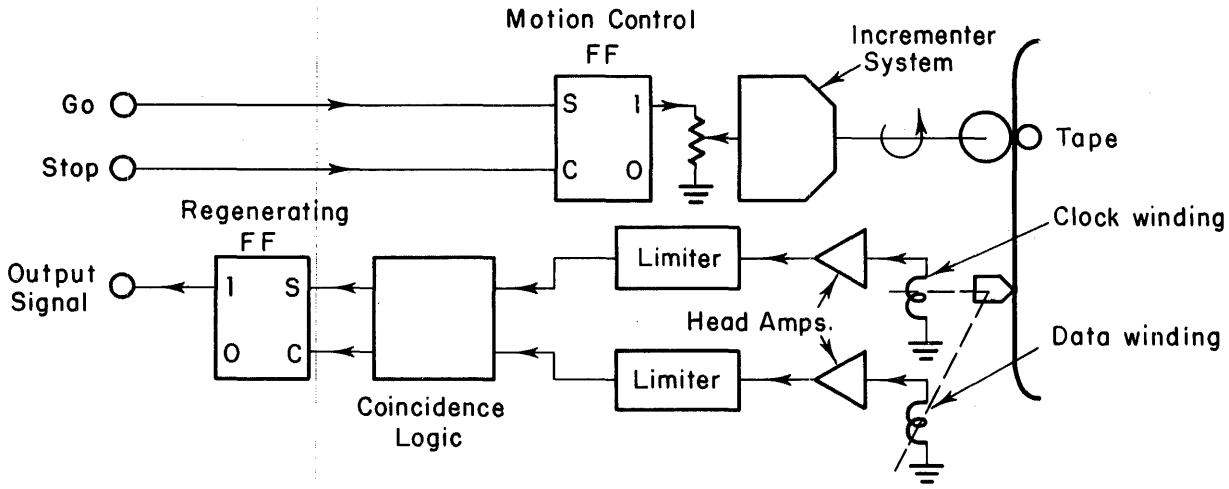


Figure 10.

circuit which regenerates the form of the original data stream.

The signal transitions (i.e., bit rate) resulting from this type of read-back are "noisy" in time since there is no speed stabilization on the tape motion other than that provided by the servo. Speed (and therefore frequency) bumps on the order of $\pm 5\%$ can be expected in the short term. Long-term accuracy of $\pm 10\%$ is readily achieved.

This read-back method is not suitable at rates of less than 800 baud and is limited in the upward direction by mechanical tape path considerations to about 3000 baud.

An improved but slightly more complex read-out method is obtained by using the system as an incrementer (Fig. 11). In this arrangement, the clock track output is used for a "stop" signal, as previously described, while the "read" command pulse is obtained from either the internal local osciallator or an external reference pulse. In Fig. 11 it is shown as originating from an external source. The read command pulse and the data pulse are applied to a logic system including a single bit storage flip-flop and a regenerating flip-flop to recreate a replica of the original input signal.

In this case, the output signal is not noisy in time

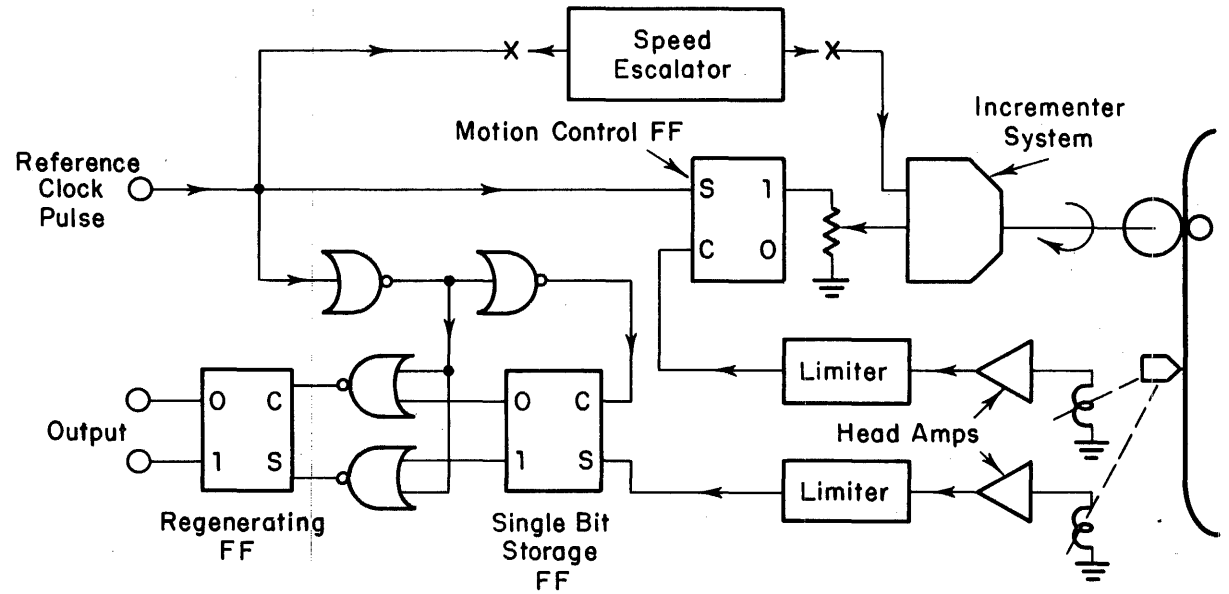


Figure 11.

but has, instead, transitions which occur exactly in synchronism with the reference pulse train. The upper frequency limit for this operation without the further improvement described below is 600 baud.

The upper frequency limit for synchronous operation may be extended by a further improvement. This is obtained by inserting the block marked "Speed Escalator" in Fig. 11 between the points marked "X", and is subject to the restriction that at rates above 600 baud the system will *not* stop the tape before the next *bit*. That is, a synchronizing time of one to six *bits* (at 200 bpi) is required on starting and a similar number of bits will be lost while stopping, the synchronizing interval measured in bits being proportional to the square of the operating speed in excess of 800 baud. For higher speeds therefore, the system begins to operate as a data "block" reader.

The effect of the speed escalation circuit is to provide an additional command to the incrementer which increases the limiting speed as the reference (read command) pulse is raised. The incrementing rate therefore tracks with the command, holding the entire system in synchronism with the reference clock up to a maximum speed set by mechanical limitations in the tape path. This limit lies at about 3000 baud for a simple mechanical system. At this speed, about 6-bit intervals (0.03″) are required to synchronize.

In this case also the output signal is not noisy and remains in exact synchronism with the reference clock throughout the speed range. From a servo viewpoint, the system operates as a random incrementer timed from the tape signals at low speeds and gradually executes a transition to a so-called "phase-locked" servo locked to the tape as the speed is raised to high speeds.

In summary, therefore, the random incrementer drive applied to a single capstan serial tape transport provides a system which will record synchronously at rates up to about 3000 baud (when speed escalation is used) and which will read synchronously or free-running over the same range. It should perhaps be specifically noted that the drive is reversible simply by inverting the command signal to the incrementer electronics so that rerecording and rereading operations can be obtained through the use of additional electronics.

PARALLEL FORMAT DUAL-CAPSTAN BUFFERING SYSTEM

This more sophisticated apparatus uses two incrementer systems in a dual-capstan drive with ½″ magnetic tape as a buffering system for store and forward applications in a digital communications path operating at rates up to 4800 baud.

The device includes five basic functions. These are the input serial-to-parallel converters, the parallel data recording system and tape drive, the machine control logic system, the parallel data reading system with its tape drive, and the output parallel-to-serial converters. These elements are shown in the block diagram of Fig. 12.

The input and output blocks constitute the interfaces between the storage system and the outside world. Since the parallel-to-serial conversions which occupy these blocks depend upon time relationships in the external signals, they both contain clocks or timing generators which must be compatible with the particular type of information which the buffer is supposed to handle. A change of element speed or bit rate on either side of the buffer requires a corresponding adjustment in the clock rate on that side, as is the case with the previous serial machine.

It is, of course, possible to omit the serial-to-parallel conversion blocks. In this case, the middle three components function simply as a parallel-to-parallel storage device. The heart of the buffer is, in fact, contained in these three blocks and a description of the apparatus is most easily understood by assuming initially that we are dealing with a parallel device and looking at these functions first.

The mechanical arrangement of the magnetic tape path as shown is intended to indicate that the recording and reading stations are separated by a distance which is as small as possible. When there is no information stored on the tape in the machine, the tape is drawn up into a "tight" condition between the magnetic heads. The existence of this tight condition is sensed by a switch in the tape path and modifies certain of the machine functions, as will be explained later.

When a character arrives at the input terminals of the recording circuitry, it is immediately recorded and transported to the output where it appears in a register. If the character is not immediately removed from the machine, additional characters arrive at the machine input and will continue to be recorded upon the tape which will therefore begin to form a loop containing stored information between the recording and reading stations. The size of this loop can grow until it contains approximately 250,000 characters. At any time during this process, characters may be removed from the output, which will, of course, tend to decrease the size

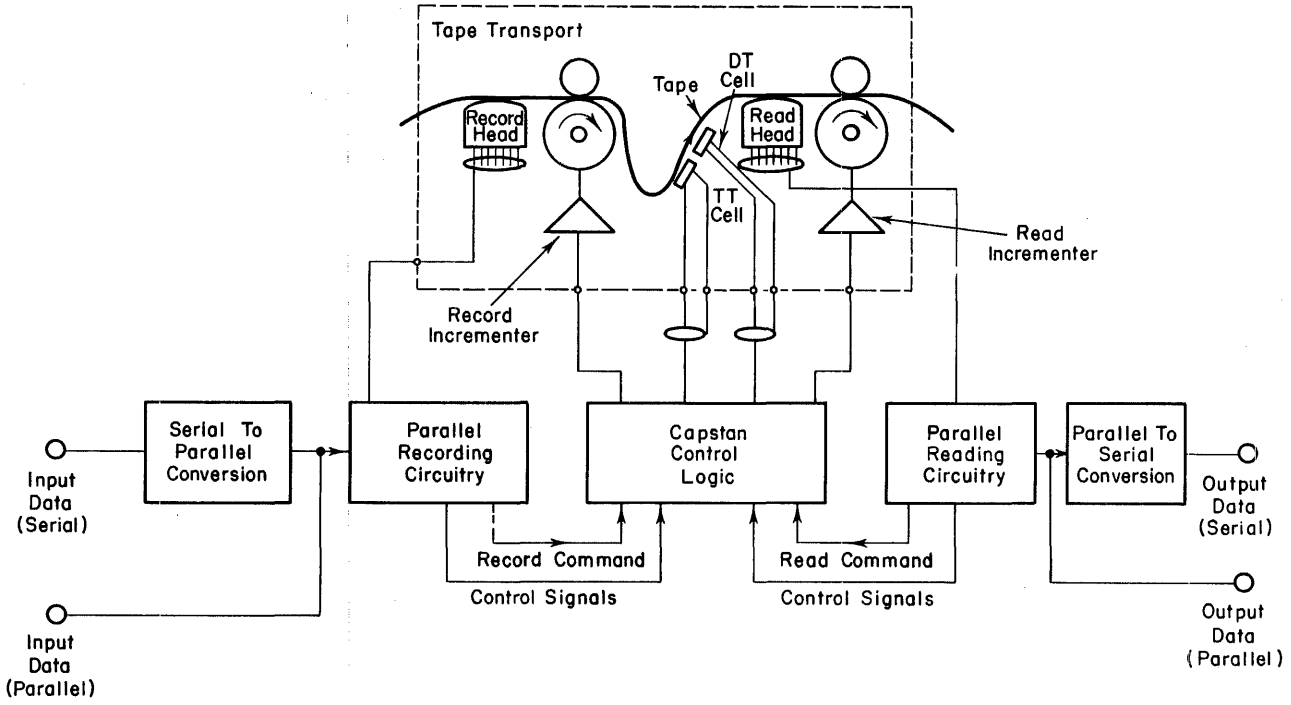


Figure 12.

of the loop. Finally, when characters are no longer being applied to the input, the output can continue to run until the tape has resumed its tight condition and has been cleared of all data.

It is important to note that each capstan is individually controlled and the device may therefore be used as a speed changer. As an example, the input character rate could be 2,000 baud while the output could be provided at slower speeds such as 100 baud. The speed conversion could be maintained until the storage loop is filled to capacity. Reverse speed conversion is also possible, that is, slow input speed, high output speed. In this particular case, it would be necessary to inhibit the readout until sufficient information is built up in the loop between the two magnetic heads. Low to high speed conversion could then be maintained until all the information in the loop had been transmitted. It is obvious that once this point is reached, the output cannot be any faster than the input data rate.

The "tight tape" condition modifies the machine functions. When no input data is being recorded but transmit or read data is being commanded, the record capstan remains stationary and only the read capstan moves on external command until such time as the stored information in the loop of magnetic tape is depleted or until the tape becomes tight between the record and read head. When this final condition exists the tight tape switch actuates and commands the record capstan to move at the same velocity as the read capstan even though no information is being put into the record station. As the read commands continue, a point will eventually be reached where no character is recorded on the tape. In this case, a timing circuit is provided to transport a length of tape over the read head equal to the length of tape between the record and read heads. Should no additional characters be recorded each time a read command is given, both the record and read capstan will operate at their control velocity for this period of time insuring that no information is trapped between the two heads.

The device also provides for bidirectional tape motion in both the record and read capstans. Via proper external command, the read capstan may be reversed and stopped at any specific point on the tape. This feature is utilized for retransmission of information in the event that the receiving terminal did not properly receive the transmitted information. The record capstan provides the same type of control so that incorrect data received and recorded on the tape may be erased and rerecorded with the proper information. Other interfaces may be provided to allow this system to interrogate the received

information prior to recording. This serves the purpose of deleting certain information which is not required to be stored, providing the capability of code conversion, and allowing for additional data to be inserted for coding of the information block to follow.

CONCLUSION

The random incrementing technique can be incorporated in many equipment configurations other than those described in this paper. The flexibility and simplicity of the basic capstan drive allows the design of peripheral devices that combine the functions of a number of presently used machines into a single unit.

The new technique also offers the higher speeds and greater storage capacity of magnetic tape in new areas and can eliminate data conversion operations in many systems.

Equipment incorporating the "random incrementer" has been manufactured for RCA Communications, Inc. and is in use at the present time. The rapidly increasing amount of digital communication would indicate many additional applications and requirements for the unique capabilities of this new technique.

REFERENCES

1. R. P. Burr, J. J. Rheinhold and R. K. Andres, "A New Technique for Application of Magnetic Tape to Digital Communications," to be presented at IEEE International Convention and Exhibition, New York, Mar. 22, 1966.

2. "Operating Principles and Performance of a Developmental Random Incrementer for Reading and Writing Magnetic Tape," Engineering Memorandum No. 29, Photocircuits Corp., Glen Cove, N.Y. (Nov. 8, 1962).

IBM 2321 DATA CELL DRIVE

Alan F. Shugart and Yang-Hu Tong
IBM Corporation, Systems Development Division
San Jose, California

INTRODUCTION

Ten years ago IBM announced the 305 and 650 RAMAC Data Processing Systems—systems that heralded the on-line processing concept, wherein business transactions could be economically processed as they occurred. This was made possible largely through the use of the 350/355 Disk Storage Files, whose 5 million stored characters were directly accessible. Since that time, significant advancements have been made in direct-access memory development. These include the double density 350/355, the 1405, the 1301 and the 2302. Pioneering the removability and interchangeability features of direct-access storage media was the 1311 Disk Storage Drive. A further advancement is the 2321 Data Cell Drive.

During the conceptual stage of the 2321 Data Cell Drive, two basic decisions were made:

1. To use magnetic recording techniques for data storage because of simplicity in the recording and reproducing processes, and
2. To use thin strips as storage media because of the high volumetric efficiency in packaging the media within a machine frame.

Physically, the components are arranged in a mechanical section and an electrical section (Fig. 1). Each section is a separate self-contained frame, facilitating manufacturing, shipment, and installation. The L-shaped configuration was devised to offer optimum servicing accessibility to the cell array.

Functionally, from a circular array of 10 cells with 20 subcells each, a cell drive positions a selected subcell of 10 strips beneath an access station. At this station a selected strip is withdrawn from the subcell. Gliding on a film of air which acts as a hydrodynamic bearing, the selected strip is rotated past a magnetic head for data transfer. Upon completion, the strip is returned to its original location in the subcell.

Magnetic recording, strip transport, and logic organization are described in the next three sections. Following these is a section briefly discussing some other important design considerations. These include anticlastic curvature, squeeze film, hydrodynamic lubrication film, computer simulation, minimum wear of machine elements, and contamination control.

MAGNETIC RECORDING

Storage Medium

The storage medium is a 2¼ × 13 × 0.005-inch saturated polyester strip having an iron oxide coating for magnetic recording on one side and an antistatic coating of carbon on the other (Fig. 2). A set of 10 strips is contained in a subcell. Each strip has a pair of coding tabs for identifying its position in the subcell, and a single latching slot for picking up the strip. Chamfered sides and a "swallow" tail are introduced to control strip dynamics.

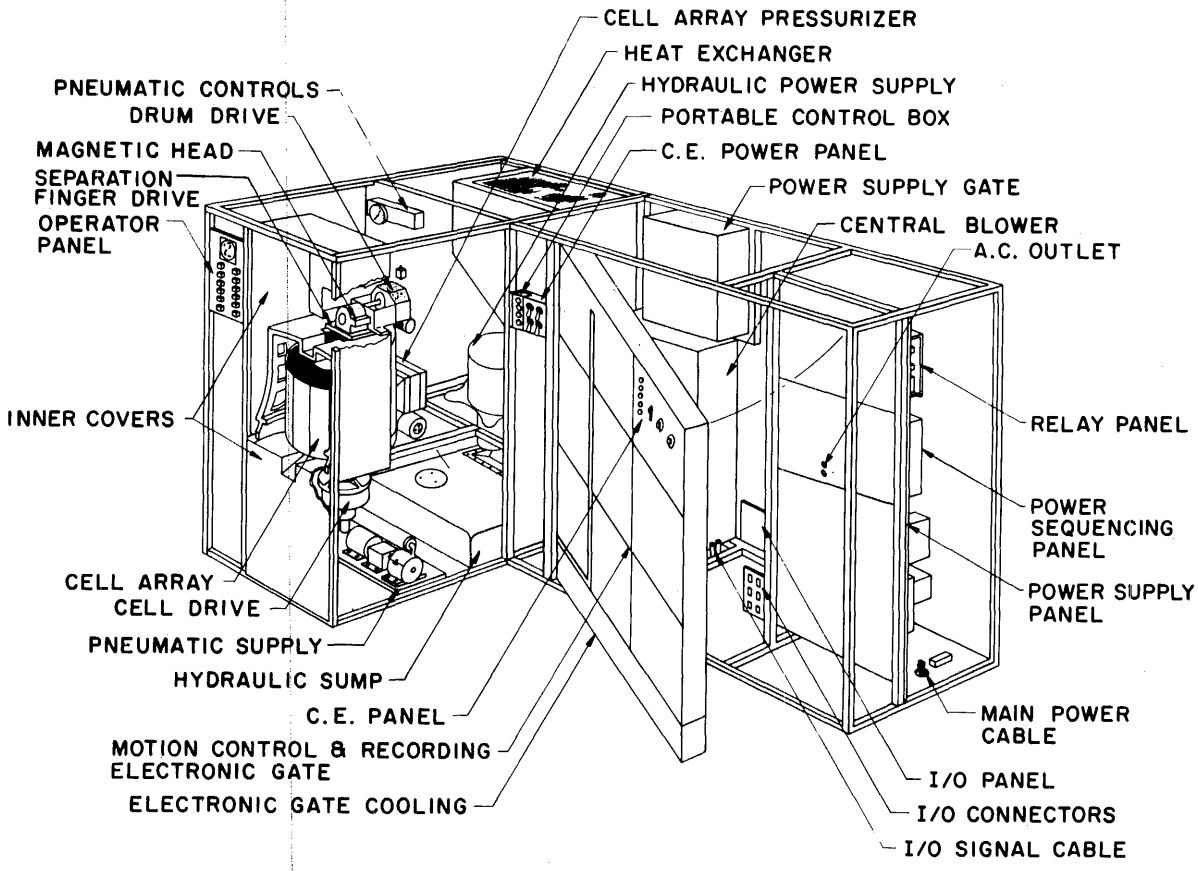


Figure 1. Component layout.

Twenty subcells are housed in a cell. Ten cells are attached around the periphery of a spindle, forming a cell array. Each cell is physically removable and interchangeable. For machine operation involving less than a full complement of 10 cells, ballast cells are used to dynamically balance the rotating array.

One hundred addressable recording tracks are available on each strip. Dual-frequency recording at a nominal density of 3500 flux reversals per inch provides each of the 100 tracks with 17,500 bits of storage capacity. Logical extensions show the following incremental storage capabilities:

		Bits
Track		17,500
Head position	(20 tracks)	350,000
Strip	(100 tracks)	1,750,000
Subcell	(10 strips)	17,500,000
Cell	(20 subcells)	350,000,000
Full array	(10 cells)	3,500,000,000

Recording is accomplished in a serial-serial fashion. At a nominal strip velocity of 250 inches per second, the data transfer rate is 437,500 bits per second.

Magnetic Head

The magnetic head has 40 laminated elements, 20 write and 20 read (Fig. 3). Each element (read or write) is physically aligned with its adjacent element on a 0.090-inch center-to-center spacing. The head can be moved to 1 of 5 discrete positions, thus providing for 100 tracks per strip.

The effects of thermal and hygroscopic expansion of the flexible medium, the element position tolerances, and strip registration repeatability are compensated by making read elements much narrower than the write elements, that is, 0.007 inch vs 0.018 inch. Furthermore, since the updating of any record requires the identification of its address, the reproducing gap precedes the recording gap by 0.150 inch.

Strip-to-head spacing of approximately 75 micro-

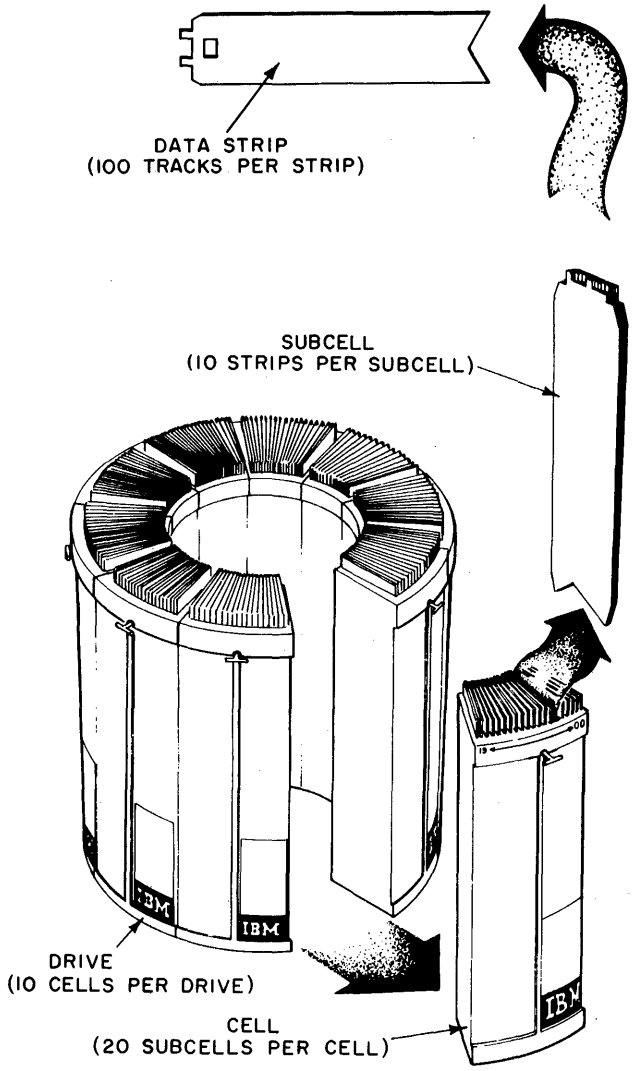


Figure 2. Data cell array.

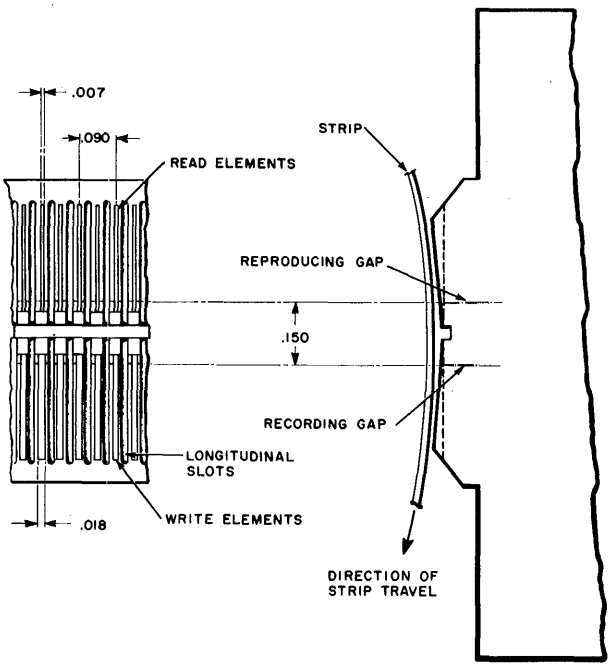


Figure 3. Magnetic head.

inches is controlled by a self-generated hydrodynamic lubrication film and longitudinal slots on the head surface. Circuitwise, a read element is equivalent to 1 millihenry in parallel with 30 picofarads on a differential basis. This represents a self-resonant frequency of approximately 1 megacycle and an impedance of about 7 kilohms at 437.5 kilocycles.

Recording Electronics

The recording circuits perform the following functions: selection, write, read, safety, and control.

Since only one of the write and read elements is operative at a time, its selection is accomplished by an XYZ diode matrix of dimensions 2 × 10 × 2, respectively. XY selects one of 20 elements, and Z controls the mode of operation (either write or read).

In recording, the data transmitted from the control unit is converted from return-to-zero into non-return-to-zero dual-frequency code by two write drivers, each supplying unidirectional current into one or the other half of the write winding (Fig. 4). In readback, processing of the signal is begun by the head preamplifier, located with both write and read matrices in a special package mounted on the head assembly. To minimize the effects of common-mode

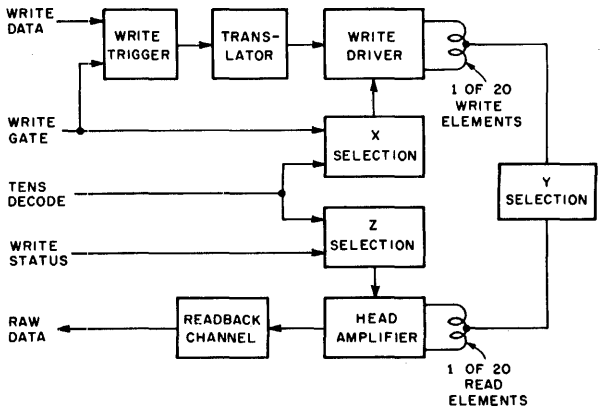


Figure 4. Block diagram of write/read circuits.

noise, amplification starts differentially and remains so throughout the entire readback channel.

Because of the dual-frequency encoding and the gap-filling techniques used in the recording process, the amplifier is never required to read through an absence of signal. Hence, the overall philosophy was to have the readback channel maintain no threshold level, so that the ability to recover data depends exclusively on the signal-to-noise ratio.

To insure data integrity, safety circuits are used to monitor critical points of the recording electronics and to set the 2321 inoperative immediately, should an unsafe condition develop.

Synchronization between data handling and strip position with respect to the head is accomplished by two transducers mounted on the drum drive and associated amplifiers. These transducers produce one pulse for each revolution of the drum and define the beginning and end of the usable strip area.

Data Handling

To compensate for variations due to the oscillator frequency and/or the medium velocity, a variable-frequency oscillator (VFO) is used in the readback process to produce a signal whose frequency and phase are controlled by the reproduced raw data. A synchronous detection is thus possible. Although the VFO is physically located in the control unit, its relationship with, and importance to, the 2321 warrant a brief discussion. The block diagram of Fig. 5 shows the significant components of the VFO:

1. The error detector, whose output is a linear function of the phase difference between data and VFO output over a periodic range.
2. The filter, which has a low-pass characteristic to attenuate fast changes in the phase error due to noise in the input signal or to instantaneous peak shift.
3. The variable oscillator, whose output frequency varies in proportion to the error signal. When the error signal is zero, the output frequency is the design center frequency of the oscillator.

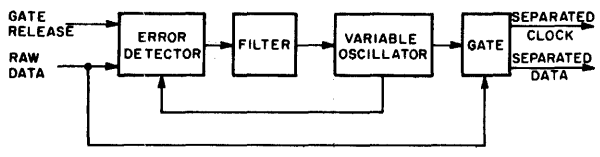


Figure 5. Block diagram of variable-frequency oscillator.

STRIP TRANSPORT

Cell Drive

The basic function of a cell drive is to bidirectionally rotate the circular array of 10 cells to 1 of 200 discrete subcell positions through the shortest angular distance. For high speed and positioning accuracy involving a moved inertia of 2.4 in-lb-sec^2, an electrohydraulic servomechanism was selected. A schematic of the drive and its control is shown in Fig. 6.

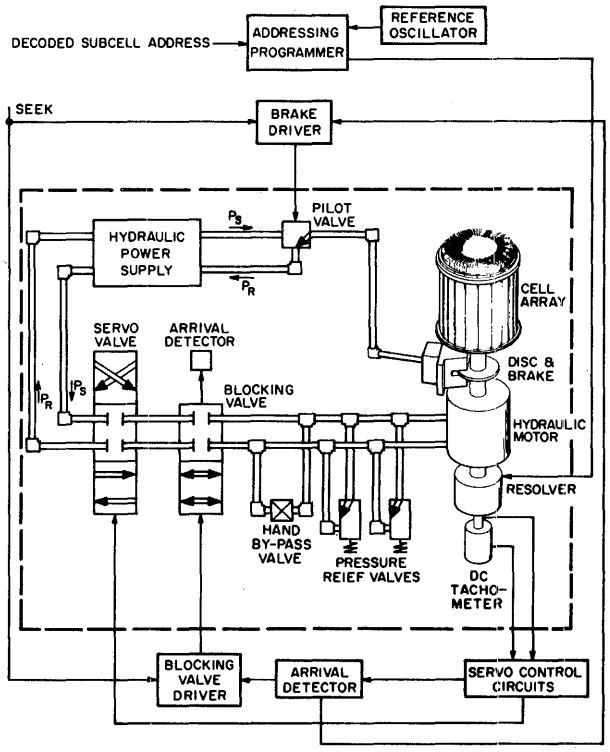


Figure 6. Cell drive and servo controls.

Cell positioning is initiated by a seek instruction. When all safety interlock conditions are satisfied, the decoded subcell address selects the appropriate reed relays. With the use of a reference oscillator and programmable transformer, the proper voltage level is applied to the stator windings of a resolver. Induced voltage in the rotor windings, generally known as error voltage, is amplified, demodulated, and summed with a velocity feedback from the DC tachometer in the servo control circuit. The controlled current is used as an input to the dual-gain servovalve.

During reed relay selection, a brake is released,

and the blocking valve is moved, permitting a flow of oil through the servovalve to the hydraulic motor which converts flow to rotary motion of the array.

As the array rotates, the rotor windings sense a decreasing error voltage which, in turn, decreases the input current to the servovalve. The resultant decrease in oil flow to the hydraulic motor decelerates the array until no further error signal is present or a null point is reached. As the null is reached, the arrival detector signals the blocking valve to be moved and the brake to be applied, locking the array at the addressed position.

Finger Drive

Concurrent with cell motion, two pairs of strip separation fingers are moved to 1 of 10 positions for the selection of 1 of 10 strips in a subcell. A separation finger drive (Fig. 7) is then initiated. The subcell springs, which maintain the vertical position of the strips, are first parted by a pair of wedges. Simultaneously, the separation fingers move vertically downward, followed by a horizontal movement for strip separation. The wedges and fingers remain in the actuated position until the strip is fully restored.

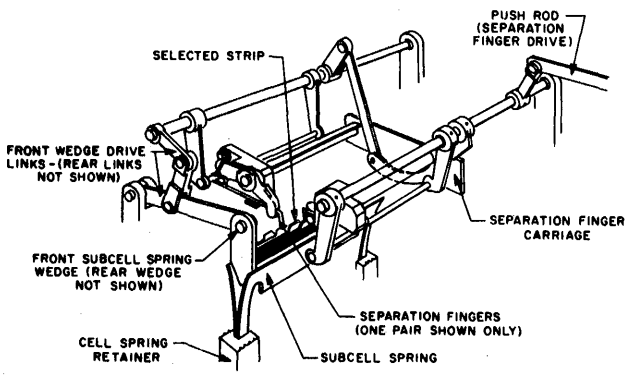


Figure 7. Finger drive.

Drum Drive

To pick and restore the selected strip, the drum drive is required to rotate both clockwise and counterclockwise. Conventional electric motors do not provide the torque-to-inertia ratio to meet the stringent access-time requirements. In addition, during strip restore, it is difficult for the motors to control the precise indexing at turn-around (defined as the shaft position when the direction of rotation changes). Lack of turn-around control further complicates the latching of the drive shaft at its home position when the restore operation is completed.

The present drive uses mechanical indexing, which allows energy storage and transfer. Two magnetic clutches are mounted on the shaft—one to maintain constant speed in the clockwise direction, and the other, constant kinetic energy in the counterclockwise direction.

The basic principle is shown in Fig. 8. A torsion spring, through the lever and the index roller, stores the kinetic energy of the drive during the restore cycle. During the pick cycle, the wound-up spring provides sufficient torque to rapidly and smoothly accelerate the drive to near synchronous speed, at which time the magnetic clutch is engaged to maintain a constant period of 50 milliseconds per revolution or a strip surface velocity of 250 inches per second. A similar arrangement controls the turn-around indexing.

Also mounted on the drive shaft is a drum. Attached to the drum by two connecting links is a pickup head which contains a latching flipper for strip pickup and a release lever for strip release.

In front of the drive is the strip housing with two openings, one for the magnetic head and the other for passage of the pickup head as well as the selected strip. Attached to the strip housing are a front guide ring and a rear mounting plate. They guide the pickup head and register the revolving strip.

The drum drive may be considered as a slider crank inversion (Fig. 9). The drum is analogous to a crank shaft, the links to a connecting rod, the pickup head to a piston, and the housing guides to a cylinder. With the drive shaft latched and the drum at rest, the pickup head is approximately $\frac{3}{8}$ inch above the strips. When the shaft is unlatched and the drum begins to rotate, the pickup head first moves downward to latch the selected strip, then upward, and subsequently locks onto the rotating drum. During restore, the strip is propelled by the pickup head in a reverse direction. The pickup head is unlocked and directed downward by housing guides. At "bottom dead center" of its downward travel, the strip, fully replaced in the subcell, is released. The pickup head then reverses its direction and continues upward to the latched or home position of the drive shaft with the drum once again at rest.

LOGIC ORGANIZATION

The logic organization of the 2321 is shown in Fig. 10. Principal areas are the interface, address-

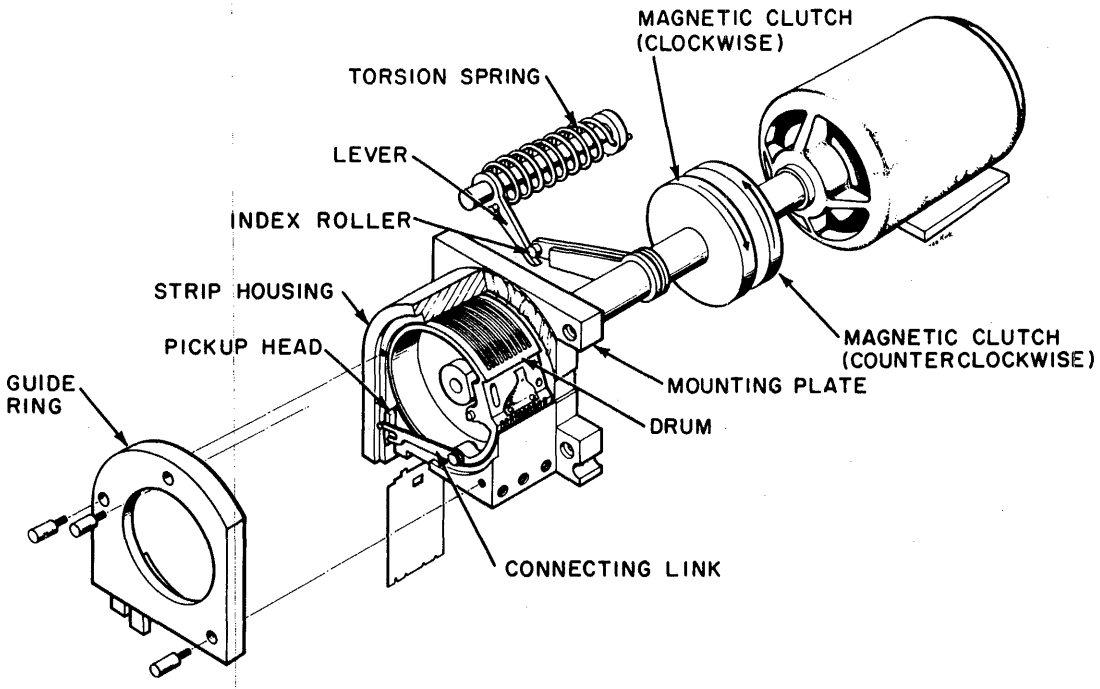


Figure 8. Drum drive layout showing principle of mechanical indexing.

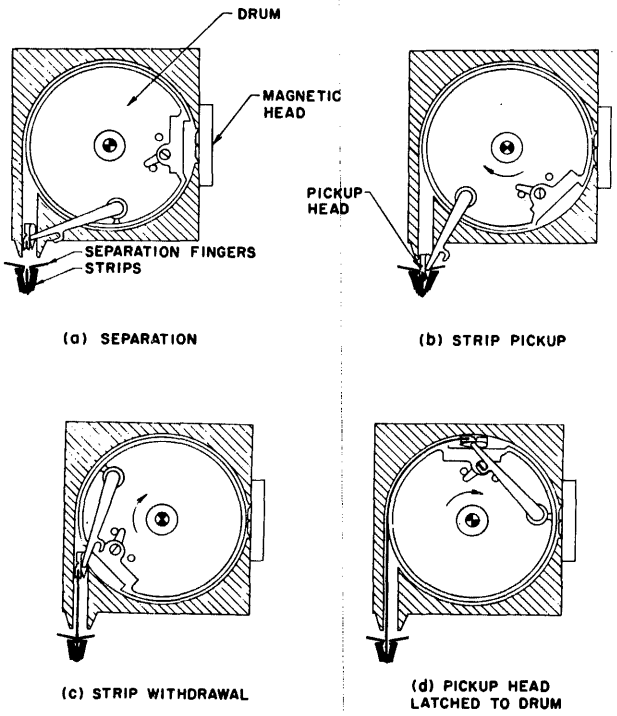


Figure 9. Strip pick cycle.

ing, motion controls, and recording electronics. Address information and control signals are transmitted to the 2321 through the interface to initiate the selection of the proper track for data transfer. Similarly, status signals are transmitted from the 2321 through the interface to indicate the drive conditions.

Addressing

The physical location of a recording track is identified by:

1. Data cell drive number
2. Data cell number
3. Subcell number
4. Strip number
5. Cylinder (head position) number
6. Track number

When a record is selected, each item of the new address listed above is compared against that of the previous address. From this comparison, the necessary electronic and electromechanical action required to place the selected record in the data transfer position is determined. For example, assume that the new record is in the same data cell drive, but the remaining portions of the address differ. The new address thus causes the following to occur:

1. The cell array is rotated to the selected subcell.
2. The magnetic head is positioned.
3. The finger drive and the drum drive are activated to select and transport the selected strip past the magnetic head.

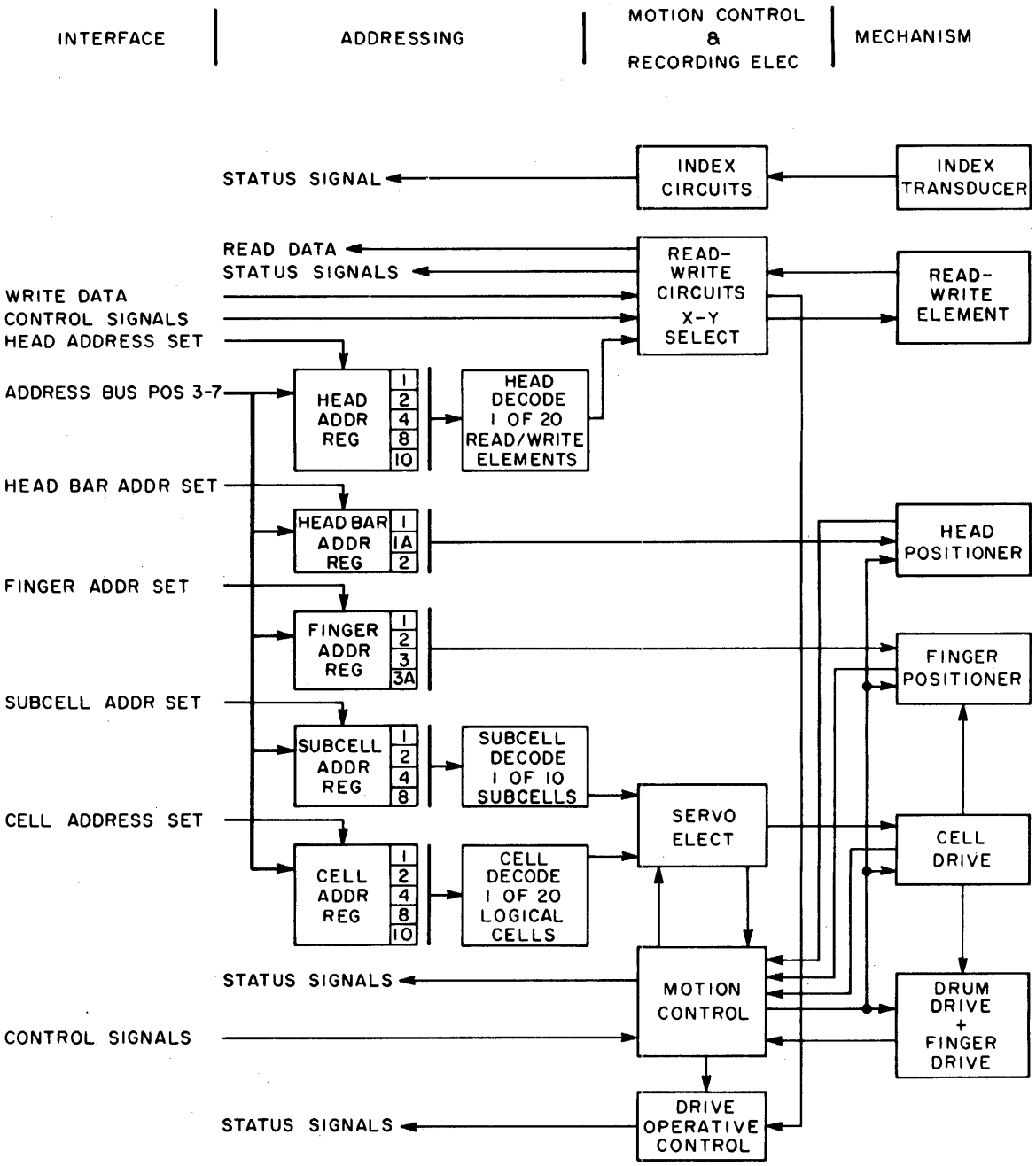


Figure 10. Logic flow diagram.

4. The specific read/write head element is electronically selected.

When the new address is the same as the previous address in a specific area, no electromechanical action is required for that area. For example, if the only difference between a new and previous address is the selection of a track within the same cylinder position, the only action would be the electronic switching of the proper read/write head.

Typical Record Update

During the cycle of updating a record, a typical sequence of events involves a seek, read (for address search and verification), write, write-check, and restore.

A seek operation causes the addressed data track to be placed in the proper status to allow data transfer, as described under "Addressing."

A read operation involves the amplification and shaping of the reproduced signals prior to their transfer to the control unit. Successive read operations, in a simple read mode, can be performed on up to 20 tracks on 20 successive drum revolutions because of the multiple read-elements-per-head design.

A write operation entails the transfer of write data to the 2321. Track formatting is organized so that updating of the data portion of a record does not require the rewrite of either the entire record or the entire track.

A write-check operation is a read operation performed following a write operation to verify the data written.

A restore operation returns the strip presently on the drum to the original subcell. This operation is performed under one of three conditions:

1. When a strip different from that on the drum is requested.
2. When the drive is not selected within 800 milliseconds upon completing the instructed operation, be it a seek, a read, or a write.
3. When the drive is specifically instructed to restore the strip.

Formatting

All storage devices on the IBM System/360 have no mechanical formatting designed into them. Instead the format of a record is determined at the record level and by the user. Each track is identified by:

1. A home address which defines the physical location of the track, as well as the track condition.
2. A track descriptor record which allows alternate track assignment.

Following the home address and track descriptor record, data records may be formatted to any record length. The maximum record length, as in a single record per track application, is 2000 8-bit data bytes.

SOME OTHER DESIGN CONSIDERATIONS

Anticlastic Curvature

Transverse deformation of a bent strip creates serious compliance and wear problems, when the strip is rotated past the magnetic head in the cylindrical housing. The phenomenon of the curled-up edges is known as anticlastic curvature (Fig. 11a).

The cross section (Fig. 11b) assumes a wave form, whose amplitude is a maximum (y_o) at either edge and exponentially diminishes to zero at the center. The major governing parameters of the phenomenon include Poisson's ratio (μ) of the elastic material, the material thickness (t), and the bending radius (R). Simply explained, a force couple exists at the strip edges, as shown in Fig. 11c, because the outer fiber of the bent strip is under tension and the inner fiber under compression.

A thorough mathematical analysis has been made, and a practical approximation of the ideal solution of the defining equations was used in strip fabrication. This requires a carefully controlled edge chamfer of the side of the strip next to the drum (Fig. 11d).

The chamfer reduces the circumferential compressive force on the inner fiber, leaving a surplus of the tensile force on the outer fiber. This surplus allows a radially inward-directed force component to counteract the force couple, producing a nearly perfect flat cross section. Uniform spacing between the rotating strip and the stationary magnetic head is thus achieved, resulting in a constant reproduced signal and drastically reducing the wear rate.

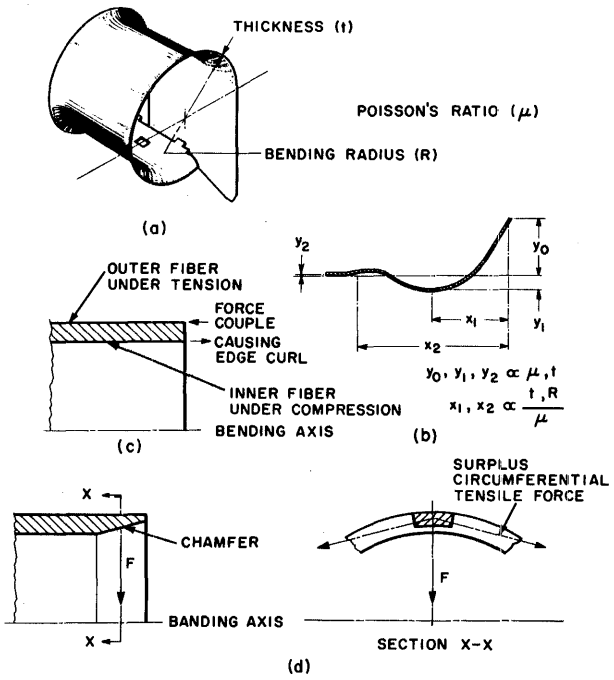


Figure 11. Anticlastic curvature.

Squeeze Film

The theory dealing with pressure of a thin flat film whose bounding surfaces are in relative lateral motion was extensively treated by Osborne Reynold some 80 years ago. However, limited work exists in the study of pressure dealing with surfaces in relative normal motion. While "squeeze film" literally applies to surfaces being brought together, the term is also conveniently borrowed here in describing the converse phenomenon when surfaces are being parted.

In transporting strips, squeeze film effect was experienced at three critical areas and times:

1. During strip separation, the swift parting of strips creates a pressure differential as shown in Fig. 12a such that:

(A) $P_1 \neq P_2$ (on either side of the selected strip)

and

(B) $P_A \gg P_1$ or P_2 (on either side of the separated packs).

In (A), long strip settling time was needed to equalize the pressures and to achieve stability before the selected strip was brought within a pickable range. As for (B), in addition to a further penalty in terms of time, a high separation force was encountered in overcoming the pressure differential.

The potential problem of access time and strip wear was resolved with the use of carefully filtered, accurately directed, and precisely balanced air flow from above the cell to nullify the squeeze film effect and to dampen strip oscillation. Serving an equally important function, this same air provides lubrication to reduce resistance, wear, and "stiction" during strip picking and restoring.

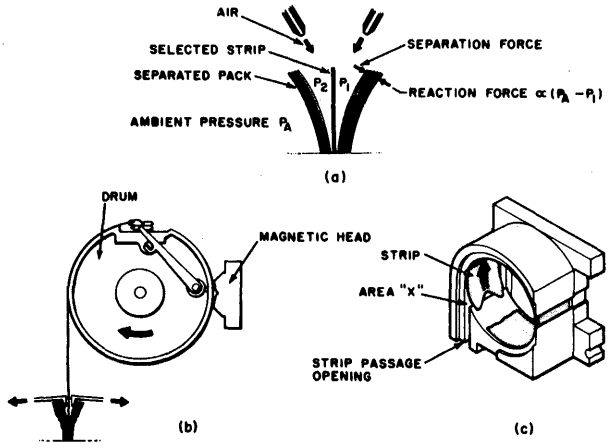


Figure 12. Squeeze film.

2. During the first pass (defined as the condition when the strip is partially in the cell as shown in Fig. 12b), the wrapping action of the strip on the drum causes the strip to adhere to the drum. In due time, after several drum rotations, centrifugal force and "unwrapping" force (similar to a bent beam) overcome the squeeze film effect and allow the strip to comply to the magnetic head. However, by introducing longitudinal grooves on the drum (as shown in Fig. 8), pressure between the strip and drum is relieved, and the squeeze film effect is removed almost instantaneously, thus permitting first-pass read/write operation.

3. During forward rotation, as the trailing end of the strip is brought sharply against the housing across the strip passage opening, the impact is cushioned by air (Fig. 12c, Area X). In this instance, the beneficial effect of the squeeze film is instead fully exploited.

Hydrodynamic Film

Compliance to the magnetic head of a flexible rotating strip constrained in a cylindrical housing is very complex in analysis, and influenced by many parameters. Two of these have been briefly mentioned under previous headings. This section deals principally with the self-generated hydrodynamic lubrication film effect.

Briefly stated, a thin layer of air, the "boundary layer," adheres to, and travels with, a moving body. When this boundary layer comes into contact with a similar layer adhering to a stationary body, a viscous shearing takes place, causing a change in momentum of the moving layer, converting the change into static pressure, and developing a load-carrying force. In the case of a rotating strip, this force balances the centrifugal force and "unwrapping" force. The film effect is achieved by creating a converging wedge-shaped spacing in two critical areas:

1. At the leading edge of the strip with a properly derived drum geometry, as shown in Fig. 13a.
2. At the area downstream from the protruding magnetic head, as shown in Fig. 13b.

Once compliance was achieved, optimization of the strip-to-head spacing became possible. This was accomplished, in turn, by controlling the effective air-bearing area with properly designed longitudinal slots on the face of the magnetic head (as shown in Fig. 3).

By shaping the unrestrained strip end in a tra-

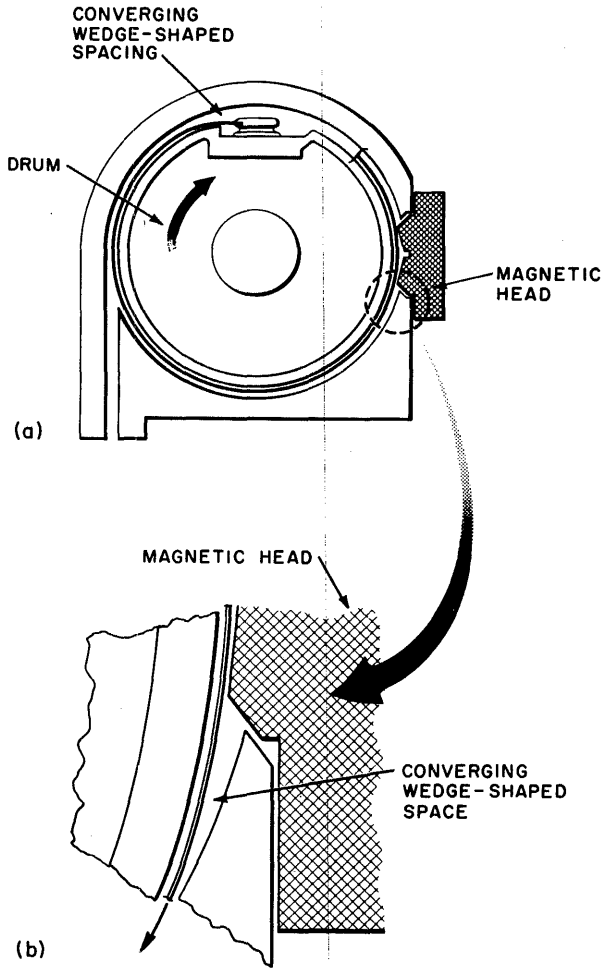


Figure 13. Hydrodynamic film.

pezoidal ("swallow-tail") design, a near circular form throughout the rotating strip is obtained. With the incorporation of other equally critical parameters (such as, surface finish, strip registration force, strip housing and magnetic head geometry, etc.), total compliance at optimum spacing was thus attained across the strip width.

Computer Simulation

Analytical techniques through computer simulation have been broadly applied throughout the 2321 Data Cell Drive Program. For instance, the powering and control of a solenoid application was optimized by equating and solving the electromechanical energy conversion process. Control of the magnetic clutches was similarly designed for optimum performance. Also, in designing the strip housing and pickup head guide, the basic fundamentals of smooth contours for static and dynamic transients were followed to avoid impact and wear. However, none of the computer simulation work is as significant as that done in analyzing the feedback control system.

Early in development, a root locus program was developed to facilitate the transient response analysis of open and closed-loop systems with limited capability in treating certain nonlinearities in the forward path. Subsequently, a digital simulation language was written with extensive coverage of nonlinear conditions, such as servovalve hysteresis and saturation, viscous and coulomb friction, electronic control, etc. Without the aid of the simulation work, the very nonlinear nature and the time-variant characteristics of the complex AC servo system would have rendered the design an extremely difficult, if not an impossible, task.

Minimum Wear of Machine Elements

A simple wear theory has been postulated and developed by IBM's Physical Technology Laboratory in Endicott. The theory is based on the maximum shear stress criterion; that is, plastic flow is

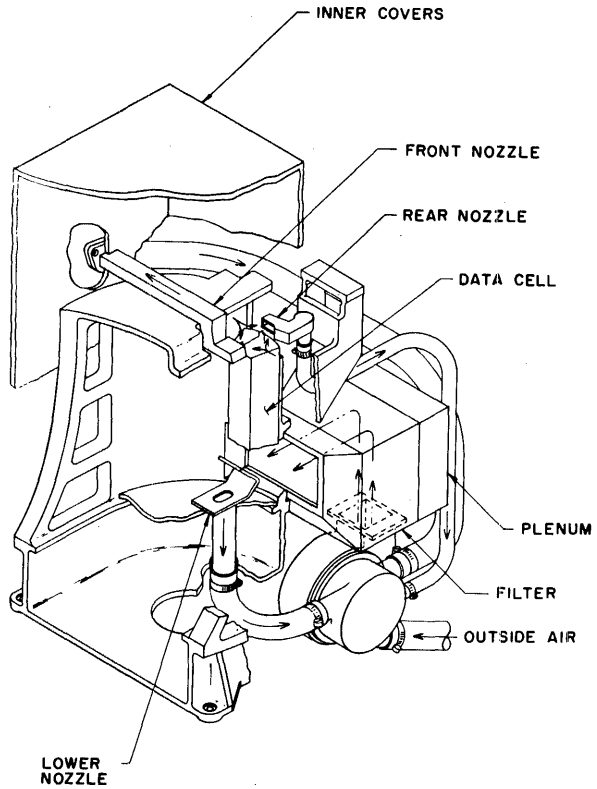


Figure 14. Contamination control.

assumed to occur when the maximum shear stress at any point exceeds the shear stress at yield. By keeping the stress in the contact area below an established level, the initiation of wear can be prevented.

The wear theory and its experimentally verified wear data have been extensively applied in determining the limiting design stresses of all critical contact surfaces in the 2321. The analysis included two general categories: first, cylinder-against-cylinder as in cams and followers, and second, cylinder-against-groove as in journal bearings.

Contamination Control

It was recognized in the early development stage that particulate contaminants in the strip transport system could adversely affect recording reliability and eventually the life of the strip. Other than the implementation of the appropriate error-recovery actions, the problem was attacked by creating a miniature "clean room," housing the area of concern. This consists of:

1. Inner covers to isolate the cell array from the surrounding mechanism, with the front entry door serving a functional purpose.
2. Pressurization of the "clean room" with filtered air.
3. Three vacuum nozzles strategically located to continuously insure the cleanliness of the critical areas.

Physical arrangement of the contamination control scheme is shown in Fig. 14. A high-pressure blower is used to draw air from the outside and to recirculate internal air through the vacuum nozzles. Inside the plenum is a replaceable filter which effectively removes micron-size as well as larger particles.

ACKNOWLEDGMENTS

The success of the 2321, embodying numerous technologies, is the result of the dedicated effort of many individuals in development, product testing, and manufacturing. To these individuals, the authors extend their sincere appreciation and acknowledgments.

HYBRID SIMULATION OF A HELICOPTER

W. J. Kenneally
Avionics Laboratory, U.S. Army Electronics Command

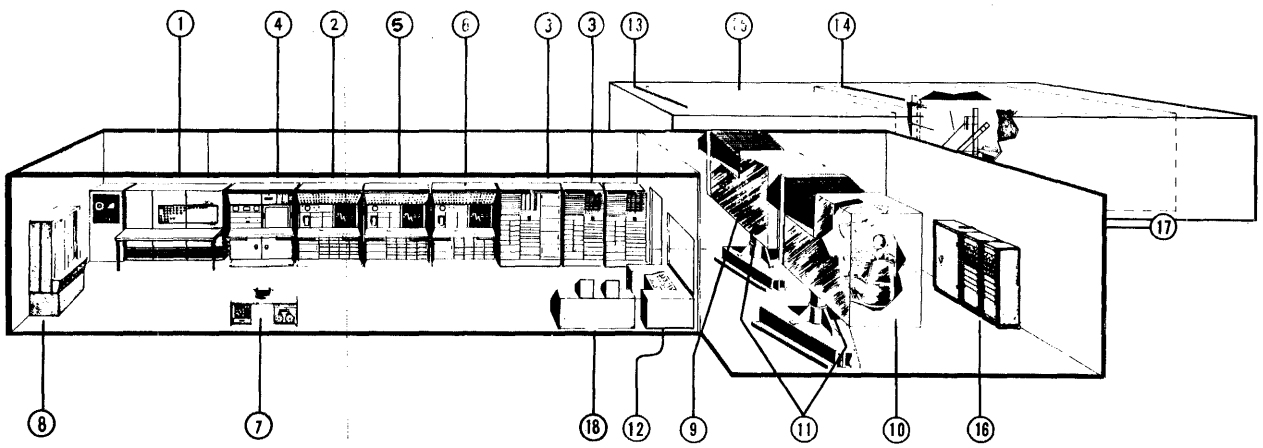
E. E. L. Mitchell, I. Hay and G. Bolton
Electronic Associates, Inc.
Princeton, New Jersey

INTRODUCTION

The recent and rapid increase in the application of the airmobile concept within the U.S. Army is well known. Accompanying the fact of the airmobile division, brigade, etc., has been the concomitant requirement for the development and improvement of the airborne electronics (avionics) to support this major innovation in conventional warfare. This increased emphasis for the development of new and more sophisticated avionic equipments and subsystems has led to the organization of an Avionics Laboratory within the U.S. Army Electronics Command. In the Avionics Laboratory, the problem of defining system performance characteristics for advanced avionics has in turn generated a requirement for analyzing the tactical mission envelope of both existing and advanced Army aircraft. One aspect of this particular task—that of evaluating avionics systems synthesized to provide particular mission capabilities has resulted in the development of a unique man-machine known as the Tactical Avionics System Simulator (Fig. 1). This simulator system integrates a real-time hybrid digital-analog computer (expanded EAI HYDAC 2400) with two operable cockpits—e.g., functional combination of crew inclosures, motion systems, synthetic instruments, control loading, and acoustic and visual simulators (Fig. 2). As illustrated, the TASS includes all of the necessary subsystem hardware to provide for the simulation of the aircraft, the avionics systems and the external environment. The basic requirement in the implementation of this expensive and sophisticated system was that the crew be able to realistically "fly" the aircraft from hover thru transition to high-speed flight while providing the avionic engineer with a "hands on" simulation capability.

The primary goal then was to develop a computer program that would provide for the realistic representation of the vehicle dynamics, which when coupled to the rather extensive "cue generators" (e.g., acoustic, motion, feel, and visual simulators) would generate an accurate and meaningful tactical environment and hence a valid base for the evaluation of various avionic equipments and systems.

The first major system to be evaluated with the aid of this system was that of the Advanced Aerial Fire Support System (AAFSS)—a next generation armed helicopter. Schedule requirements dictated that the hardware components that collectively comprise the TASS would be delivered only 75 days prior to the initiation of the AAFSS evaluation. The installation, checkout, integration, and computer programming for the evaluation were required to be accomplished in this 75-day interval. To



1 DIGITAL COMPUTER
2 HIGH SPEED ANALOG
3 NON-LINEAR EQUIPMENT
4 DIGITAL OPERATIONS SYSTEM
5 231R ANALOG COMPUTER
6 HIGH SPEED ANALOG
7 DIGITAL CONTROL DESK
8 30" X 30" X-Y PLOTTER
9 PILOT'S COCKPIT
10 GUNNER'S COCKPIT
11 COCKPIT MOTION SYSTEMS
12 CENTRAL CONTROL DESK
13 PILOT'S TERRAIN MODEL
14 GUNNER'S TERRAIN MODEL
15 SYSTEM AVIONICS (BLACK BOXES)
16 CONTROL LOADING SYSTEM
17 VIDEO SPECIAL EFFECTS GENERATOR
18 VIDEO CONTROL CONSOLES

Figure 1. Tactical Avionics System Simulator (TASS).

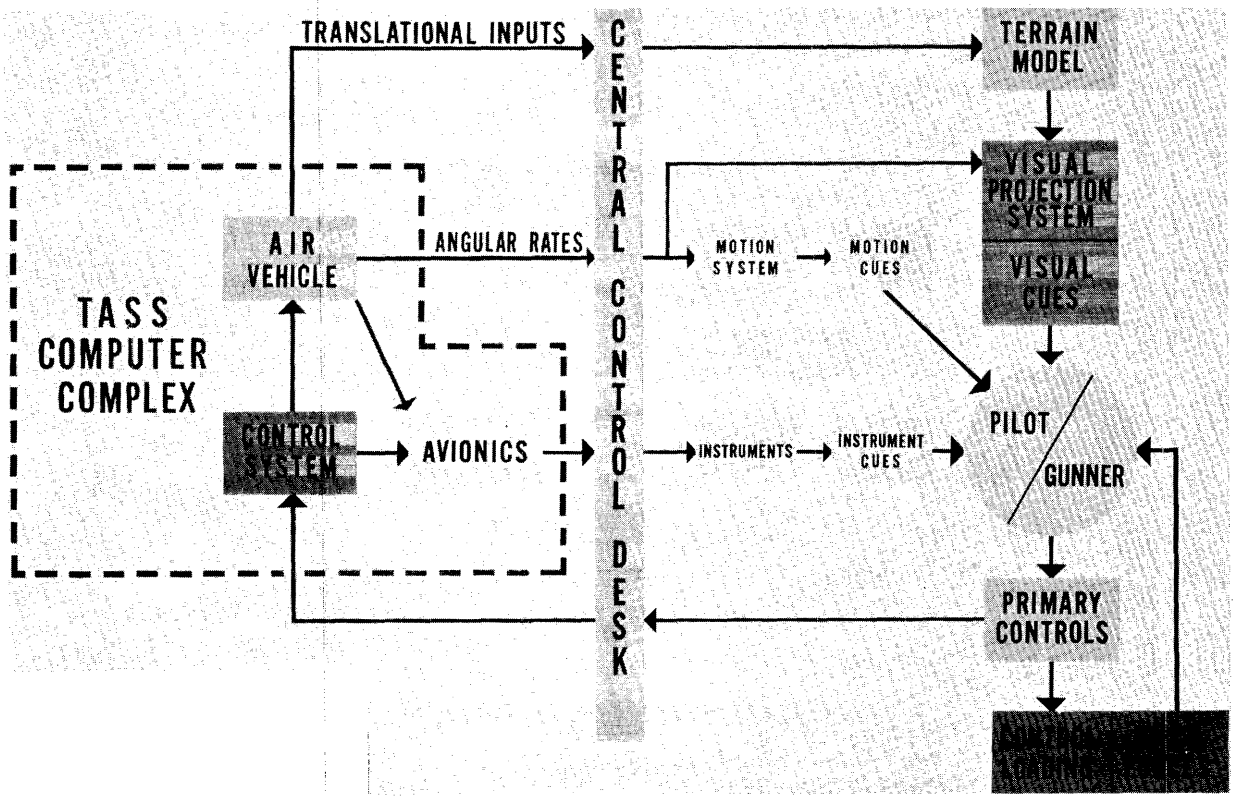


Figure 2. TASS functional block diagram.

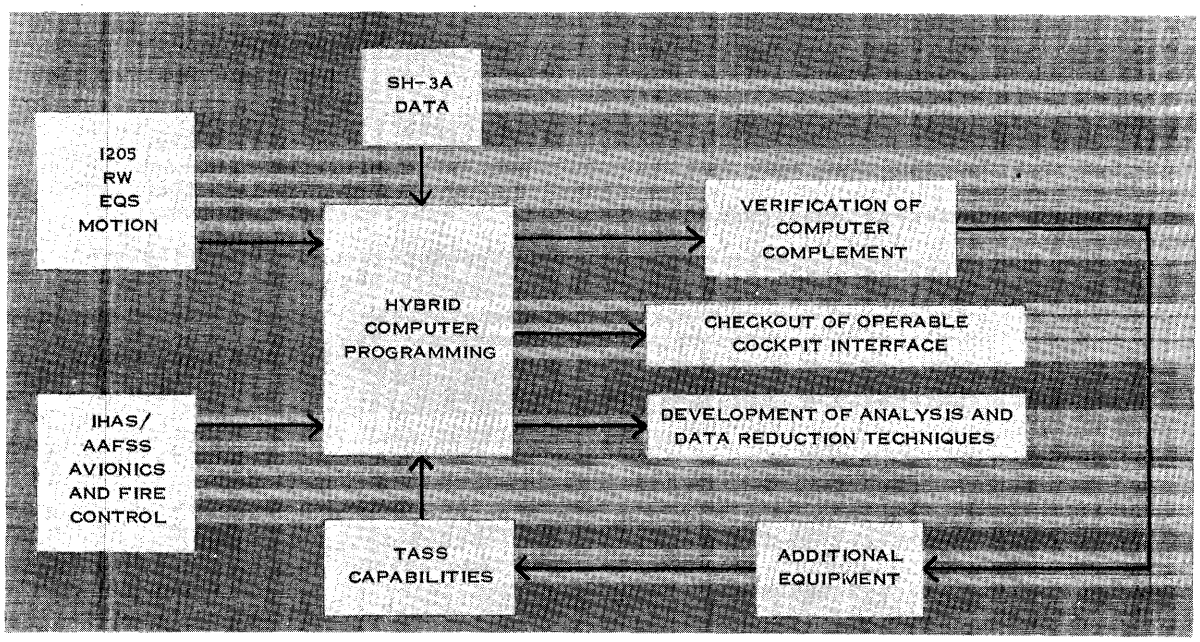


Figure 3. Development of generic model.

minimize the risk inherent in a program scheduled this tightly, a software development program (Fig. 3) was initiated at an earlier date to provide:

- *Verification of the Computer Complement.* This step was intended to identify the additional computer modules that would be required to accomplish the necessary simulation of the AAFSS vehicle.
- *Checkout of Operable Cockpit Interface.* Emphasis here was on the definition and implementation of the specific interfacing required to provide for man-machine-computer integration.
- *Development of Analysis and Data Reduction Techniques.* Again, the program scheduling required that the "Evaluation tools" be in hand at the beginning of the evaluation period. This particular effort provided for the development of the data reduction techniques and functional implementation to verify the concepts postulated.

The software program illustrated did, in fact, develop a hybrid computer program for a generic single rotor helicopter complete with outer loop avionics and fire control subsystems. If you will, the conceptual system was a "straw man" that allowed the analyst to come to grips with a realistic engineering model of a typical system.

Of greater significance than the successful accomplishment of a project goal however is the flexibility afforded by the basic concept of the computer program developed. From the viewpoint of an *avionics system designer* the aircraft (be it proposed or existing) must be considered a given condition since no design alteration can be accomplished. Design alteration of the avionics is however both possible and necessary and since the hybrid computer program developed accomplishes the calculation of the aerodynamic forces and moments on the digital side (main rotor, tail rotor, etc.) and the coordinate resolution and integration on the analog side, the avionic designer can easily implement the study of both inner and outer loops of the flight control system, the crew interaction, and the navigation, terrain following, formation flight and display subsystems while accepting the given condition of the aircraft performance as a boundary value. Viewed more generically, the digital computer stores the program that describes the aerodynamically unique data corresponding to a particular aircraft. Thus, the digital computer can be thought of as an "Aero Function Generator" and with the introduction of the unique aero data into the digital program a different aircraft configuration can be interjected into the avionic analysis loop.

The potential of this aircraft switching capability is considered to be of significant value in the design evaluation of avionic systems that apply to more than one aircraft (e.g., flight control systems) but must be tailored to each particular aircraft. Looked at in a more direct manner the simulation program provides for the "hands on" simulation of the avionics and the "hands off" simulation of the basic aircraft.

The concept expressed here has been proven since the generic hybrid computer program developed was used as the function building block in the analysis of proposed avionic designs for the Advanced Aerial Fire Support System. The specific computer program has recently been updated to accurately reflect the latest design configuration for the AAFSS and is presently being used as the real-time model for design synthesis and analysis.

HYBRID PROGRAM

Organization

The major airframe simulation was divided into computational tasks suitable for analog and digital

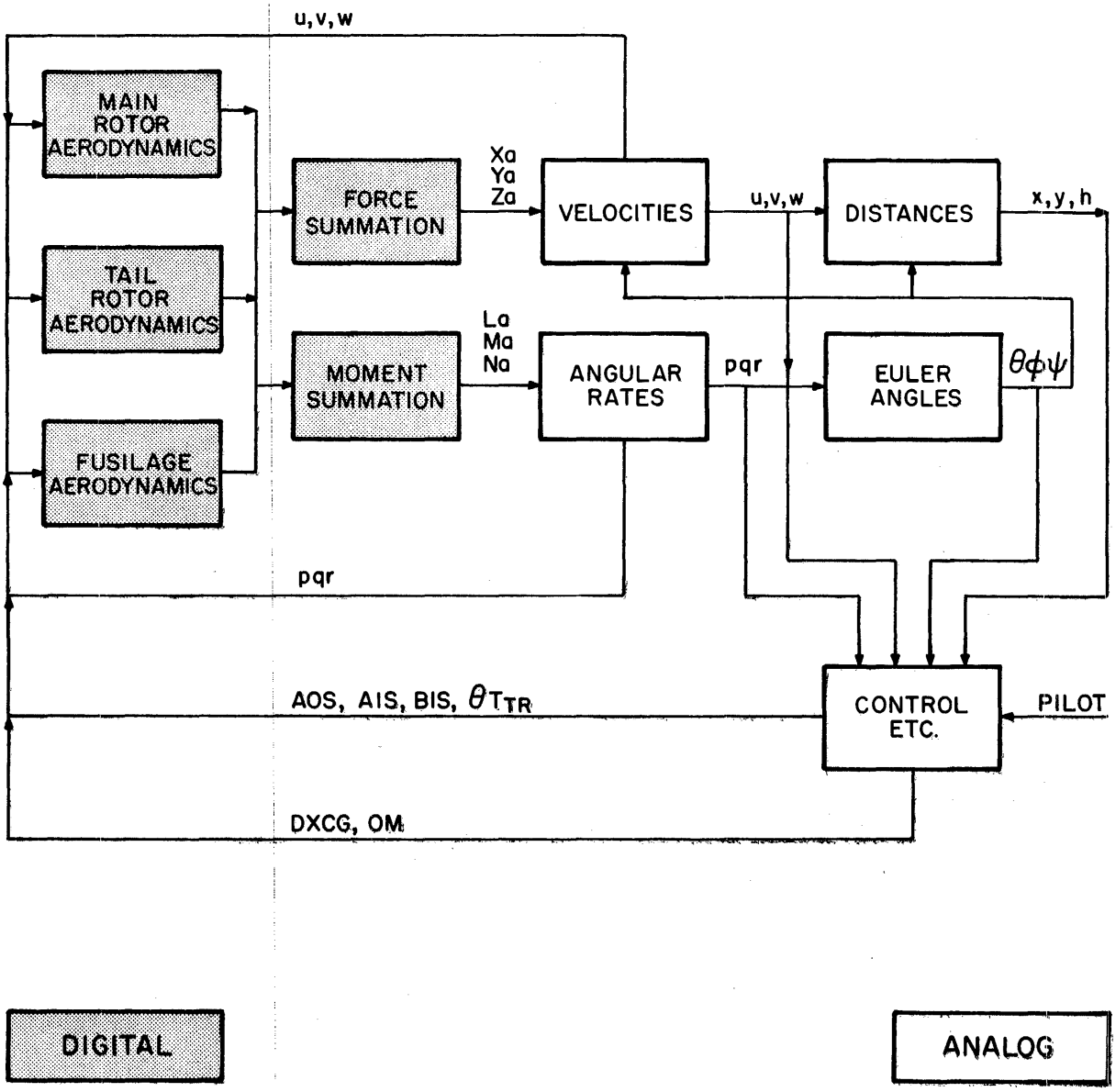


Figure 4. Division of tasks between analog and digital sections.

sections of the hybrid computer. With the requirement that the simulation be able to cover a full mission from takeoff to touchdown, a representative simulation of the airframe dynamics over all flight regimes was of great importance.[1-3]

It was found that a straightforward, conventional analog program for the complete simulation would have required too large a complement of analog equipment, and of this the major demand was from forces and moments produced by the main rotor.

Hence, a division of tasks between analog and digital sections was arrived at by assigning all the forces and moments calculations to the digital section and integrating these forces and moments by continuous, parallel analog equipment to derive the required outputs.

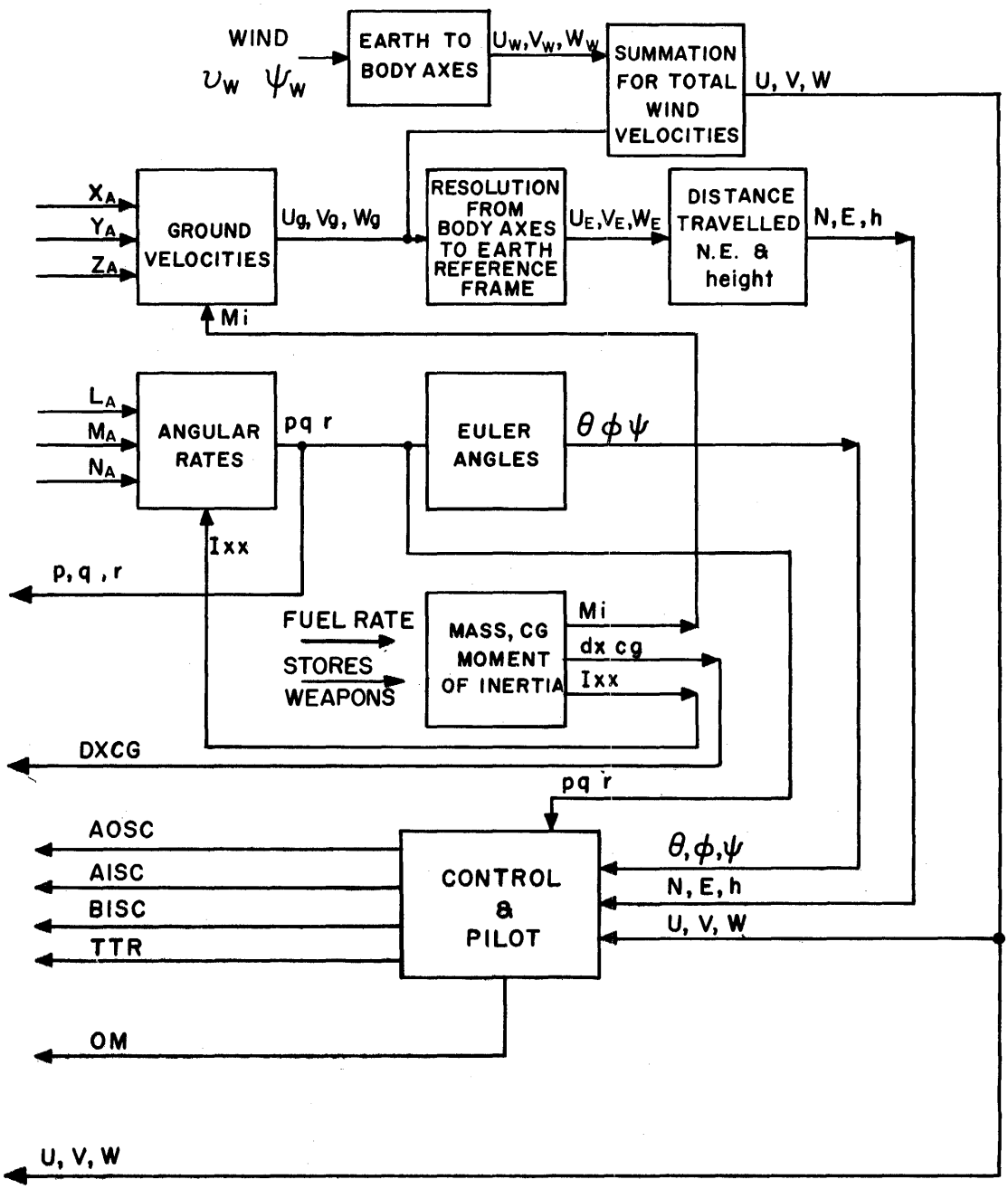


Figure 5. Analog program—block diagram.

The scheme is shown in Fig. 4. The analog requirement was approximately 100 amplifiers, which left a considerable amount of equipment for the simulation of the flight control systems.

The digital computation of the forces and moments of the main rotor, tail rotor and fuselage was accomplished in 50 msec, thus giving a sampling time of 20 samples per sec. One consequence of hybrid programming is the existence of an interface between analog and digital sections. As a result of the sampling effects introduced, there is an unavoidable phase lag that can adversely affect loop damping within the simulation.

However, using digital prediction on the computed forces and moments, equivalent damping to the real-world vehicle is obtained—up to about 2 cps natural frequency.

Analog Section

The analog program developed for the six degrees of freedom simulation followed the block diagram shown in Fig. 5. Here, the moments and forces generated digitally were integrated to give angular rates and body velocities, and these in turn were integrated to give vehicle orientation and distance traveled. The operations involved in the computation are primarily integrations and rotations of the velocity vectors from one axis system to another.

Figure 6 shows the generation of angular rates and orientation angles of the airframe. So as to allow for continuous feedback paths around the rate integrations, the hub moments were linearized as functions of the control inputs A_{1S} and B_{1S} and spin rates p and q. These feedback terms were then allowed for by subtracting equivalent terms from the hub moments computed on the digital computer, so that only differences had to be transmitted.[4]

$$\Delta L = L - (L_p P + L_q Q + L_{A1S} A_{1S} + L_{B1S} B_{1S})$$

$$\Delta M = M - (M_p P + M_q Q + M_{A1S} A_{1S} + M_{B1S} B_{1S})$$

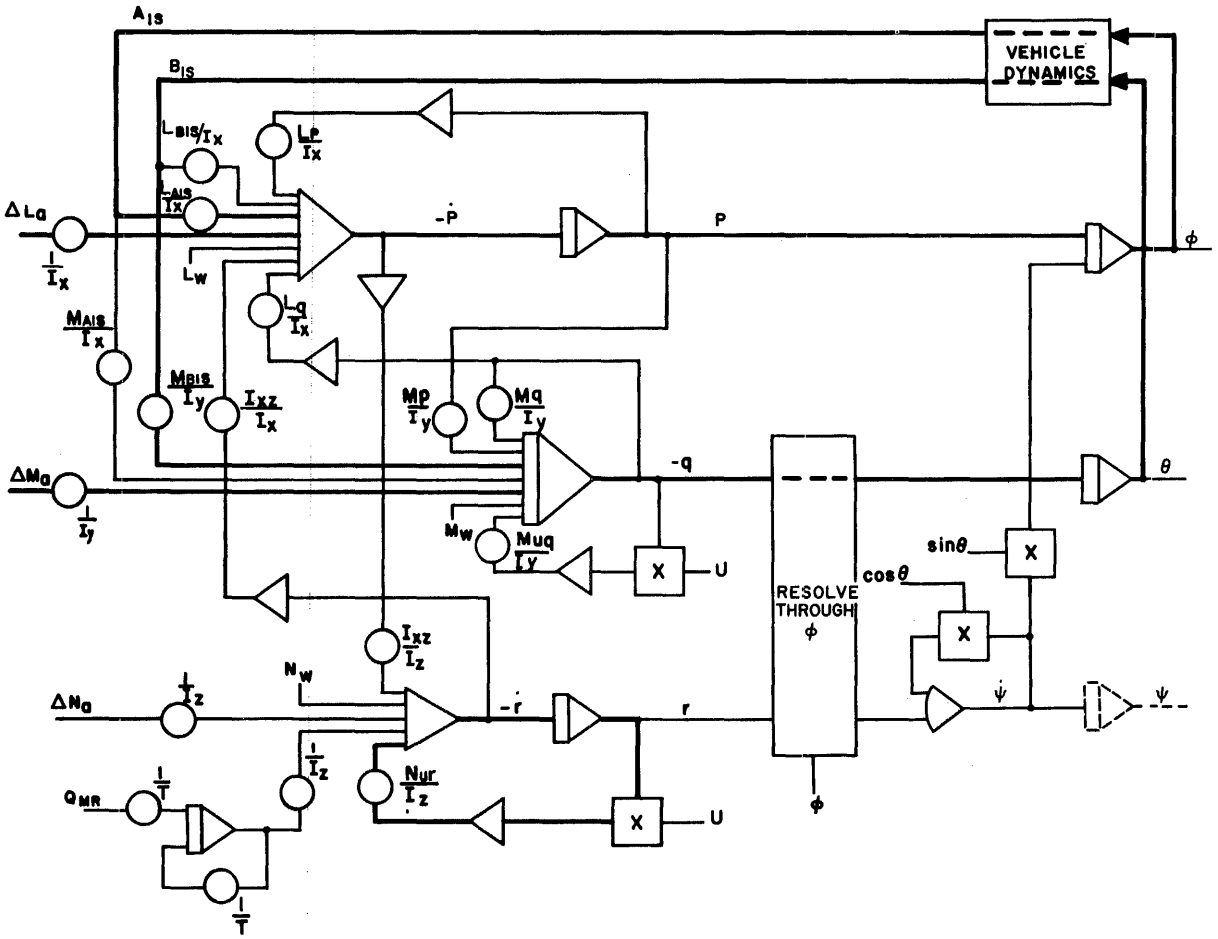


Figure 6. Analog simulation of angular rates and Euler angles.

Since the moments due to velocity were not removed, these corrections were only close to zero at hover. It should be mentioned, however, that there was not really any approximation, since the net pitching moment at the analog integration at any instant did correspond to that computer digitally, but part of it arrived a little early to help reduce effects of phase shift.

Values for the derivatives were obtained by measuring changes in the variable at various speeds and choosing compromise values to cover as wide a range as possible.

Avionics, comprising an inner loop stability augmentation system and outer loop long-term stabilization were simulated by standard analog techniques to provide the necessary control inputs and feedback.

Linkage Section

Data transfer between the analog and digital sections of the simulation was controlled by a standard program written for the HYDAC 2400. This program provided for up to 20 channels of analog to digital and 12 channels of digital to analog conversion. The transfers were 12 bits and sign in a sign magnitude convention.

The Input/Output Subroutine on the digital computer controlled all data transfer, reading a block of multiplexer channels on analog to digital input, and establishing voltages on banks of track/store amplifiers on digital to analog output.

The analog to digital conversion at 70 μsec per channel was used to input the variables u, v, w, p, q, r, Ω, A_{0S}, A_{1S}, B_{1S}, θ_{TR} and dx to the digital computer. Since scale factors are normally different between the analog and digital sections of a hybrid computer, the scaling was adjusted by multiplying by a constant during the spare time while the next channel was being converted, before storing the variable in an input buffer.

Because the number of DA converters was limited to six, voltages were transferred to two banks of track/store amplifiers to provide the equivalent of block demultiplexing. The first bank was used to output X, Y, Z, L, M, N to the analog computer, and the second outputted V_T, Q_{MR}. Spare channels in this bank were used for monitoring digital variables dynamically and were found extremely useful during debugging phase of simulation.

Digital Section

The digital section received inputs from the pilot's control actions through A_{0S}, A_{1S}, B_{1S}, θ_{TR}, from the helicopter velocities u, v, w, the spin rates p, q, r and the rotor speed Ω. The primary purpose of the digital program was to compute all forces and moments acting on and about the center of gravity of the vehicle. The total forces and moments were then

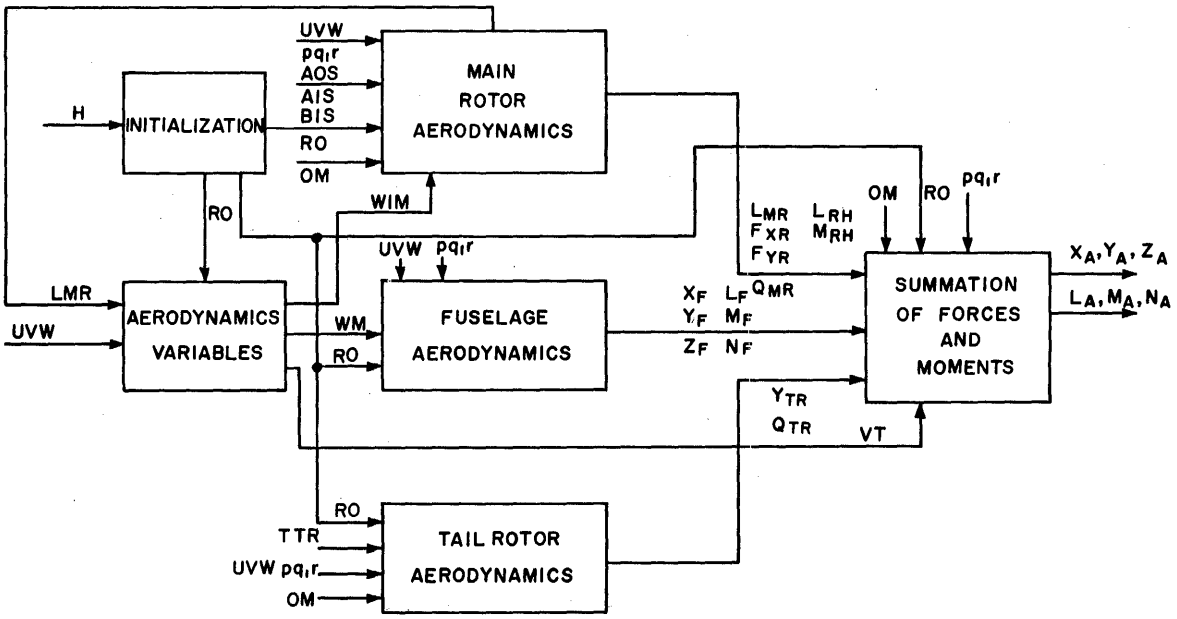


Figure 7. Digital program—block diagram.

transmitted to the analog computer 20 times a second, synchronization with real time occurring just before the input data block where a waiting loop was executed for the master timing pulse.

The digital program generated these forces and moments by a succession of subprograms which calculated the effects of fuselage, tail rotor and main rotor separately. These are shown in block form in Fig. 7.

A secondary function of the digital program was to make various program options available during program checkout and running. These permitted printing of intermediate results, programmed halts, system checkout with test data, and for the hybrid operation, inclusion of a prediction computation to compensate for the 50-msec delay introduced by the computation interval.

Implementation of the digital programming phase was concerned with coding and extensive checking. Fixed point arithmetic was used throughout the computation cycle, necessitated by the short cycle time required because of interaction with the continuous analog section. The fixed point coding of the arithmetic operations was quite straightforward; once scaled equations had been written, occasional checks for overflow and improper divide being the only precautions taken.

The main program embodied all the computation for the 12 rotor stations, since this was by far the largest section of coding. Computations for the forces and moments of the other components, i.e., wing, tail, propeller, etc., were coded as closed subroutines mainly to simplify program organization and checkout and to simplify modification as the programs developed and the vehicles become more firmly defined. Communication between main and subprograms was by means of a common storage area so that no time was required to transfer arguments and results from one region of storage to another.

Of the various programs shown (Fig. 7) by far the most important is that of the Main Rotor. Its organization was dictated by the requirement to step around the azimuth computing forces and moments on the disc at each point. The main rotor was represented by 12 stations defined by the azimuth Ψ and station y—discussed previously.

The sets of computations of forces and moments contributed by the stations Ψ, y were essentially identical, differing only in the terms involving sin Ψ and cos Ψ. Because the values of Ψ are multiples of 90°, clearly it was not necessary to evaluate all the sines and cosines; instead the program was arranged to branch to one of four paths at each of two appropriate points. For each path, it was then necessary only to add, subtract or omit corresponding terms equivalent to sin and cos of 1, -1, or 0.

REFERENCES

1. "Simulation of Helicopter and VSTOL Aircraft," vol. 1, Navtradevcen 1205-1.

2. "The Simulation of Helicopter Motion and Avionics System," PCC Report 65-6, Electronic Associates, Inc., R & C Division, Princeton, N.J.

3. "Preliminary Dynamic Report for Trainer SH-3A (HSS-2) Weapon System," Navtradevcen 1524-5.

4. M. Connelly and O. Fedoroff, "A Demonstration Hybrid Computer for Real Time Flight Simulation," AMRL-TR-65-97.

A TIME-SHARED HYBRID SIMULATION FACILITY

R. Belluardo, R. Gocht and G. Paquette
United Aircraft Corporation Research Laboratories
East Hartford, Connecticut

INTRODUCTION

The capability for combined analog-digital computation at the United Aircraft Research Laboratories was greatly expanded in January 1966 by the addition of a large-scale, time-shared, general-purpose digital computer and a pool of hybrid linkage equipment which can be distributed to any of four analog consoles. The digital computer time-sharing system provides hardware and software for the simultaneous use of the system by several users at teletype consoles.

This expansion was necessitated by the increased demand for time on an existing hybrid facility which has been in operation since December of 1962.[1,2] Typical uses for this equipment include determination of static and dynamic performance characteristics of rocket engines, jet engines, fuel controls, helicopters, and V/STOL aircraft.

GENERAL

The ability to efficiently and simultaneously handle several problems of varying size and complexity, and ease of utilization were the basic requirements which influenced the selection of components for this system. The first of these requirements was met by selecting a digital computer capable of performing both time-dependent (real time) and non-time-dependent calculations on a machine sufficiently fast and large enough to simultaneously accommodate several problems. The second of these requirements was met by selecting a digital computer system and designing a hybrid linkage system such that each user, whether hybrid or digital, would find this system as easy to program and operate as he would a single non-time-shared hybrid computer.

A Digital Equipment Corporation PDP-6 Digital Computer was selected for this purpose.[3] This computer is equipped with a core memory of 65,000 36-bit words and 16 high-speed storage register/accumulators. Cycle time for these memories is 1.75 microseconds and 400 nanoseconds, respectively. Five teletype stations, six Dectape units each capable of storing about 65,000 words and a paper tape reader constitute the input/output devices for this system. Randomly addressable core storage associated with a single processor can be expanded to 262,000 words. The system can accommodate several parallel processors. The number of teletype stations can be expanded to 64. A schematic of the complete system is shown in Fig. 1.

This computer will simultaneously accommodate several problems in real and non-real time in addition to several users involved in utility operations such as edit, assemble, compile and debug. Each user's program resides in a portion of core memory and is protected from all other users by hardware and software features incorporated in the machine. On-line communication with the system permits use of symbolic debugging techniques with resulting

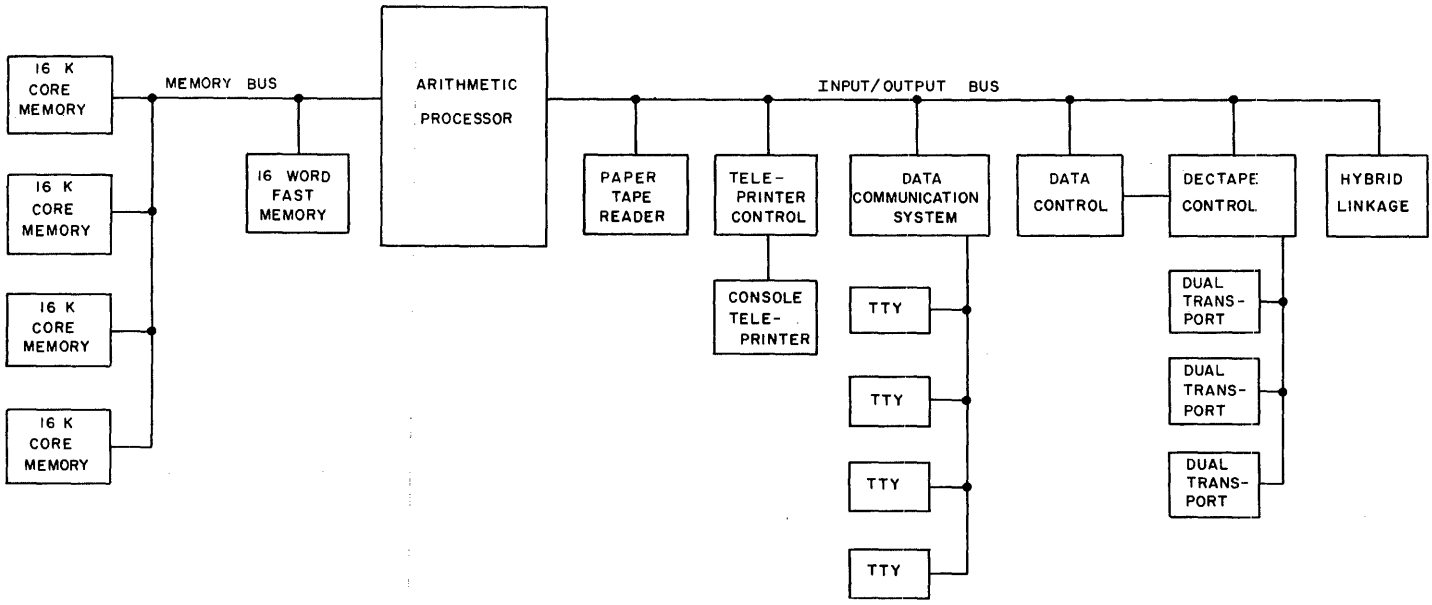


Figure 1. Schematic of digital computer.

reduction in large problem checkout time. Finally, initial and operating costs are low enough so that relatively long periods of time at one of the consoles are still economically feasible.

Basic to this digital system is a 6000-word executive routine which provides overall coordination and control of the total operating system. Additional software packages available include an editor, FORTRAN II (FORTRAN IV is currently under development), a macro-assembler, a relocating linking loader, a desk calculator and a symbolic debugging program.

The present hybrid linkage system contains a Raytheon Multiverter which combines a 48-channel multiplexer with a 14-bit analog to digital converter. Total conversion time is under 40 microseconds. The system also contains 40 Adage Model 4W13 digital to analog converters. These units feature ± 128 volts output and 14-bit accuracy.

Since the system was designed as a multi-user system, it was felt that all users should be able to address particular D/A's or A/D's beginning with address 0. This was accomplished by designing address relocation hardware for the linkage system. Basically, relocation constants are added to A/D and D/A addresses. These constants are predetermined numbers that are functions of how many D/A's and A/D's are assigned to other users. Protection hardware which prevents inadvertant addressing of converters assigned to another user was also designed into the system.

The remainder of this paper is concerned with a detailed description of the hybrid linkage system and modifications to the PDP-6 monitor system that were required to make time-shared, time-dependent computing possible.

THE HYBRID LINKAGE

The possibility of having several hybrid simulations in progress simultaneously on one digital computer creates a need for a rather special hybrid linkage. Over and above the requirements that each analog computer have the appropriate number of accurate, reliable and fast converters are the special requirements imposed by the use of one digital computer to service several analog computers. The solution to this problem was found by using a pool of linkage equipment which could be distributed easily and effectively. The analog to digital converter channels and the digital to analog converters are distributed to the four analog consoles by a patch board located within the PDP-6 hybrid linkage as shown in Fig. 2.

Special I/O Instructions and Analog Assignment

A PDP-6 user cannot normally execute any input/output instructions. Input and output are performed by the monitor on request from a user. Devices must be assigned to a user before he can expect to use them. In order to make the use of

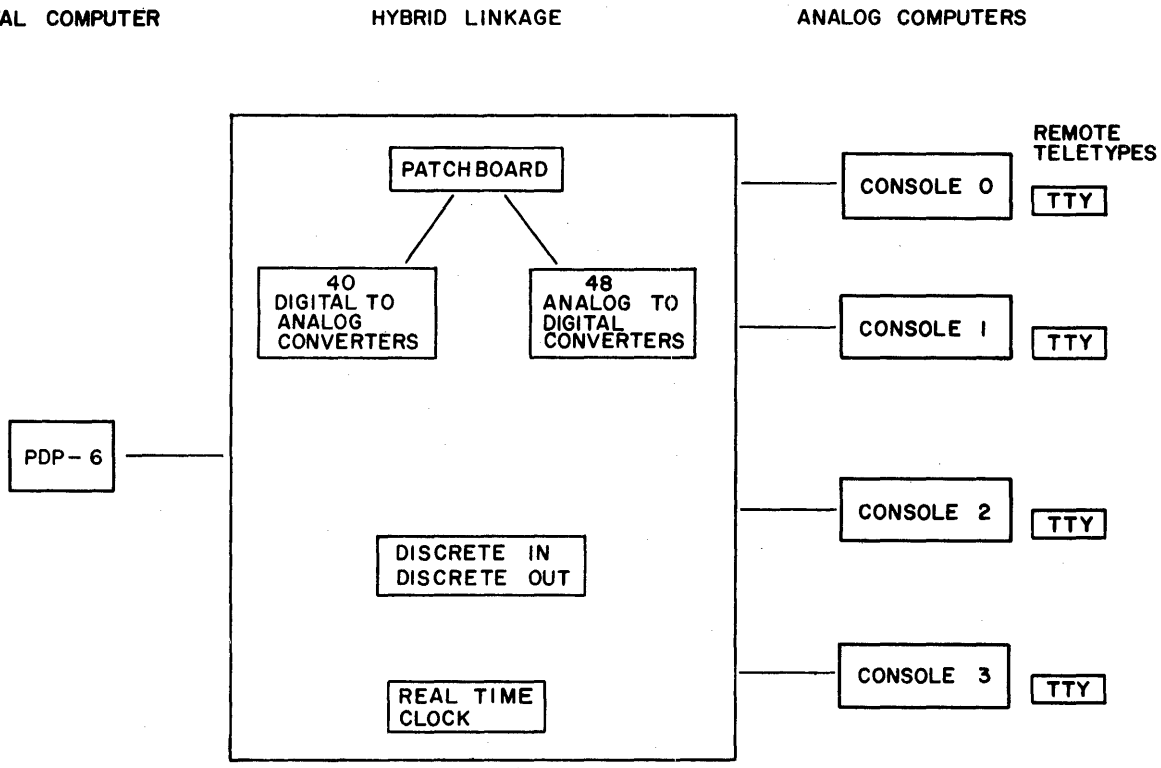


Figure 2. Hybrid linkage.

hybrid equipment as simple and as fast as possible, the in-out instructions on the PDP-6 were divided into two classes; the normal in-out instructions which are illegal for a user and the special instructions which are legal for users. Most instructions for the hybrid linkage are legal for all users.

One user cannot, however, affect another user's hybrid equipment. This protection is accomplished by both software and hardware features. Once a user has been assigned a specific analog console he can use only that portion of the linkage connected to that analog console. The monitor turns on the correct portion of the linkage whenever a job is selected for running. The monitor can actually turn on any one of five analog consoles. These are consoles NONE, 0, 1, 2 and 3. Console NONE is turned on for all users not assigned a specific analog console. A user cannot change analog console assignment except via the monitor commands.

Patch Board Wiring and Converter Addressing

Once set up the hybrid linkage allows each programmer, both digital and analog, to address analog to digital converters and digital to analog converters starting at address 0. Each analog program will use as many A/D's and D/A's as is needed.

The corresponding digital program will address the converters by identical addresses. This addressing scheme applies to all "four" hybrid computers. A particular program will not change from day to day as the converters are distributed to users in different arrangements. Of course the total number of converters is limited. This addressing scheme also protects each user.

In order to properly distribute analog to digital converters and digital to analog converters to the analog computers the hybrid linkage must be "set up." This requires a patch board to be wired and inserted in the patch bay at the hybrid linkage. The number of converters used on each console must also be entered in a toggle switch register. The linkage hardware will then automatically relocate converter addresses so that each user starts addressing with number 0.

Since this addressing is perhaps the most unusual feature of this hybrid linkage a more detailed description is in order. The outputs of all digital to analog converters appear on the patch board in the hybrid linkage. The analog computer patch panel

positions that are to be used for D/A outputs are also wired to this patch board. The D/A outputs must be patched to the analog computers in an orderly way. The D/A outputs are used starting at number 0. The required number of converters for console 0 are wired in order. The next consecutive group of D/A converter outputs are wired to analog console 1, starting with position 0 on console 1. The next group is wired to console 2 and the final group to console 3. The number of converters delegated to each analog console is then entered on the toggle switch register. The analog to digital converters are distributed in the same manner.

Figure 3 is a block diagram of the address calculating hardware. The six-bit address comes directly from the programmer's in-out instruction. The console number is selected by the time-shared monitor. The numbers N_0, N_1, N_2 and N_3 are set in the toggle switch registers and correspond to the number of components of this type assigned to the corresponding analog console.

Consider an example. Let console 2 be selected and $N_0 = 10$, $N_1 = 10$ and $N_2 = 4$. The program addresses converter number 3. The output of the subtractor signals an illegal address when the result of the calculation is zero or negative. $4 - 3$ is positive and therefore does not signal an illegal address. The addition circuit will add 3 to $N_0 + N_1$ and will therefore calculate 23. If the patch board wiring is examined at this point, converter number 23 will be found wired to patch panel position 3 on console 2.

As a second example, suppose console NONE is selected and the programmer addresses channel 0. The subtractor signals an illegal address and the converter action is inhibited. Any illegal converter reference would result in this same action.

Summary of Linkage Instructions

In this section an outline of the use of each component in the hybrid linkage is given. This is done without going into a detailed description of the instruction or data formats. These instructions are exactly the same for any of the "four" hybrid computers.

Two instructions are used with the digital to analog converter system. One instruction is used to select a converter address. The other instruction transfers data to the selected converter. Each data transfer instruction will increment the address by one. Consecutive data transfers will load consecutive converters.

Three instructions are associated with the analog to digital converting system. One loads the converter address and starts the converter, another transfers data to the computer. The third instruction is used as a skip instruction for timing purposes. This instruction will skip when conversion is complete. As the data is transferred into the computer the converter address is automatically incremented by one and the next conversion is started.

Discrete inputs can be read from each analog console into the user's digital program. Discrete outputs can be set at each analog console from the user's program. This requires two instructions; one to read and one to set. There are 12 of these discrete inputs and outputs permanently assigned to each analog console.

The hybrid linkage contains a millisecond clock that is used mainly by the real-time monitor for all timing control. Any user can, however, read the clock. This is useful in computing times for program execution within one computing time interval.

TIME-SHARING SOFTWARE

Time-dependent problem solutions using the digital computer combined with dynamic devices, as in hybrid simulation, require execution of the digital program within a repeated, fixed time interval while the problem is operating, i.e., compute mode, in analog terminology. In previous hybrid computation at United Aircraft Research Laboratories the repeated digital program cycle and the execution time were the same. In many instances, the repetition rate, or duty cycle, need not be as short as execution time. With a longer duty cycle and/or a faster computer, a portion of the duty cycle can be shared with program service activities. By minimizing the ratio of time-dependent program, execution time to an acceptable duty cycle, a number of these programs can coexist within a common duty cycle.

Time-Sharing Control

Software control is provided to accomplish time sharing by the PDP-6 Multiprogramming System.[4] This system includes a resident monitor which handles input/output service and schedules user program execution. Using the PDP-6 priority interrupt channels, the monitor sequences programs based on delayed input/output requests, clocked timing, and user's programmed or teletyped commands. As shown in Fig. 4, service is provided, in order, to input/output device requests, monitor service requests from user software or teletype, and a round-robin sequencing of user programs.

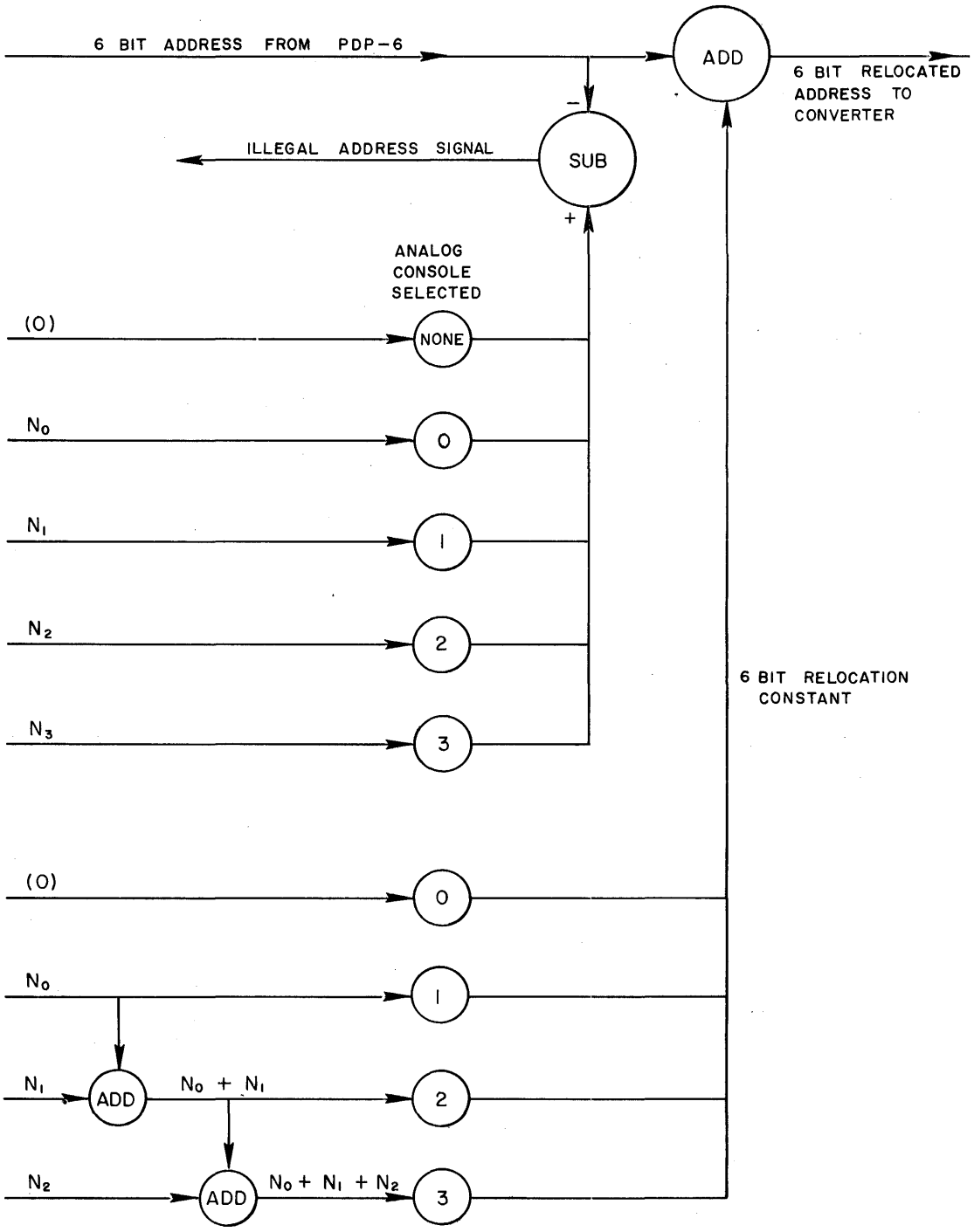


Figure 3. Converter addressing.

Programs in the round-robin include the users software as well as library routines called by the user such as editor, FORTRAN, etc.

Exact, repeated interval control is not a function of the basic monitor. Clock control is used to terminate programs in order to prevent long operation of a single user without looking at the service requirements of others. However, clock control is at low priority and is often deferred for monitor or input/output functions.

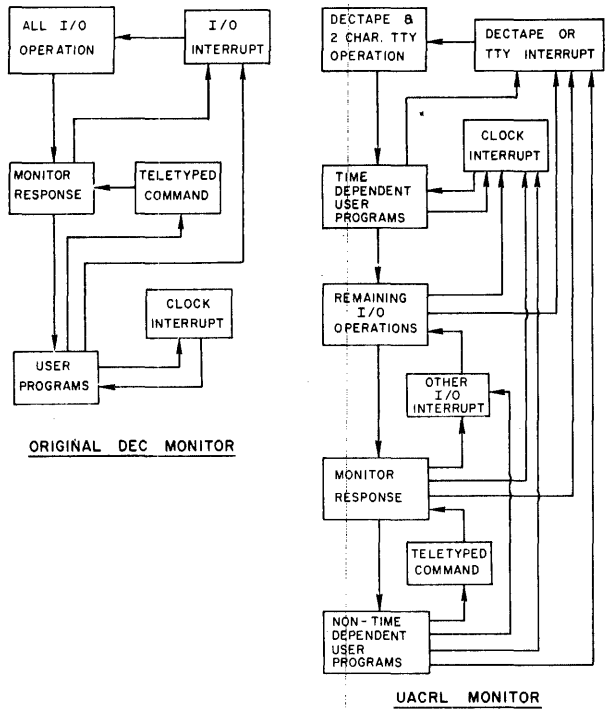


Figure 4. Monitor priority.

Exact timing of time-dependent users has been designed into the monitor using the UACRL clock at high priority. The time-dependent programs are sequenced within an accurate, repeated time interval (the duty cycle) preset to a desired value in milliseconds. Each time-dependent user receives a preassigned position and time interval within the duty cycle.

Dectape and teletype reading and writing operations can interrupt time-dependent service. Any of the routines used at this higher priority is completed well within one millisecond. Their combined use over the duty cycle should never exceed 10% of the duty cycle.

The current time-sharing at UACRL permits up to 95% of the established duty cycle for time-dependent use. Time-dependent users are commutated within the duty cycle in a queue separate from the round-robin of non-time-dependent users. The remaining 5% of the duty cycle is reserved for monitor functions and the round-robin. While the percentage for non-time-dependent use seems small, time-dependent users often operate in compute mode for reasonably short, infrequent intervals. When compute mode does not exist, most of this user's time is returned to the system for lower priority service.

In order to implement this form of time-sharing, a number of new features were added to the monitor. These functions, described below and summarized in Table 1, include control of:

1. Duty cycle
2. User time
3. Time synchronization
4. Hybrid linkage assignment

The last is provided for the assignment of proper linkage hardware to each user.

Duty Cycle

The duty cycle can be set by a teletyped command TSET x, where x is the desired duty cycle in milliseconds. This command is accepted by the monitor only when all time-dependent jobs are absent since the duty cycle affects all time-dependent users. If a user attempts to set the duty cycle when it is protected, he receives a typed return indicating the number of a job still in time-dependent status and the original duty cycle is retained.

A command TCLEAR can be typed by a resident time-dependent user to permit a duty cycle change while retaining his original memory and equipment. He must reinstate his timed operation by a TJOB command, described below, once the new duty cycle is set.

One command, TREAD, can be entered from teletype or from the user's program to access timing information. As a teletype entry, TREAD will yield a printout of the duty cycle, time-dependent job numbers and their reserved time intervals, and time left for additional reservation (balance of 0.95 duty cycle in the present system).

The use of TREAD as a programmed operator results in the transfer of the duty cycle value to a designated machine accumulator. This information is useful in calculating data affected by the time interval as a problem initializing operation.

User Time

With one time-dependent user in the system, execution of the program at the start of each duty cycle would provide accurate cycling. Program execution time varies due to data values and program branching and interruptions, but accurate time cycling of one user is possible by converting analog information at the start of the cycled program.

With more than one time-dependent user, the second and later users must be assured of similar

Table 1. Summary of Commands Added to the Monitor

Influences	Source	Command	Action	Errors	Teletype Response
Duty Cycle	Teletype	TSET x	Set duty cycle to x in milliseconds if none of the jobs are currently in time-dependent status.	x > 8191 or x = 0	x MSEC CAN'T BE SET
				Duty cycle protected by existing time-dependent job.	j IS A TD JOB
				None	Carriage return/line feed
	Teletype	TCLEAR	Release duty cycle protection for initiating job.	None	Carriage return/line feed
	Teletype	TREAD	Type value of duty cycle, reserved user intervals, and remaining time available for time-dependent use.	None	n MSEC DUTY CYCLE TIME JOB NUMBER m_1 j_1 ⋮ ⋮ m_L j_L x MSEC AVAIL
	Program	TREAD	Transfer value of repetition time to specified accumulator.	None	None
User Time	Teletype	TJOB y	Establish a reserved time interval, y, in milliseconds, if available, and set duty cycle protection for the initiating job.	y > (0.95 Duty Cycle less sum of existing reserved intervals)	z MSEC AVAIL
				None	Carriage return/line feed
	Teletype	TJOB 0	Release the execution time interval (or set zero time) for the initiating job. Duty cycle protection is set.	None	Carriage return/line feed
Time Synchronization	Program	TSYNC	Return control to the monitor for job switching and signal the execution of the succeeding instructions in the initiating job when its next execution interval begins.	Execution when job does not have nonzero reserved time interval.	TSYNC FROM NON TD JOB
				TSYNC instruction not executed before clock interrupts time-dependent job.	TD JOB TIMED OUT
				None	None
Hybrid Linkage	Teletype	ENABLE ANAx	Reserve analog console x linkage for the initiating job, if available. The physical hardware assignment exists only during this job's operation. Only one console may be assigned to a job.	Device previously enabled by another job.	ANAx ENABLED FOR JOB j
				Nonexistent console number.	NO SUCH DEVICE
				Illegal device name	DEVICE CAN'T BE ENABLED
				None	Carriage return/line feed
	Teletype	DISABLE ANAx or DISABLE	Release analog assignment for the initiating job.	None - note, wrong address is ignored and any console that was assigned is released.	Carriage return/line feed

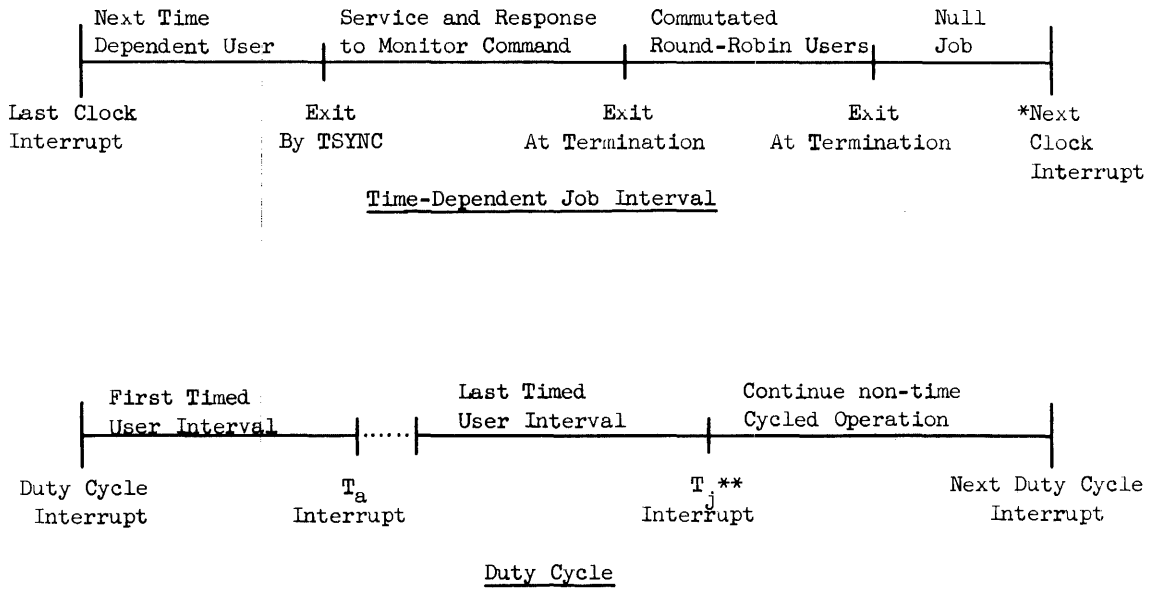
cyclic scheduling. This is accomplished by requiring all time-dependent users to establish a reserved interval within which they must complete their operation. Any time not needed by the assigned user, within his time interval, is allotted to the queues of non-time-dependent operations.

Time-dependent programs operate at high priority following each interrupt for successive reserved intervals until all job intervals are exhausted. The remainder of the duty cycle is used for non-time-dependent job functions. If all job operations are exhausted, the remainder of the duty cycle is spent in counting this dead time (the null job). The null job data is a useful indicator of computer utilization.

Figure 5 illustrates the subdivision of the duty cycle. The monitor schedules all these operations within separate queues with high priority given to the time-cycled queue. The information supplied to the clock comes from a ring table in sequence. This table contains the reserved job intervals and the balance of the duty cycle. Thus, with two time-dependent users active, the executive sequences three time values to the clock which determine the interrupts. When the executive responds to an interrupt, it sets the clock interval for the next operation. This will always occur within the first clock count after interrupt was signaled. Should the interrupts occur during execution of a time-dependent program, the job is reduced to round-robin operation and is informed of the error via teletype.

To establish an interval for the user a command TJOB y is typed into the system, where y is the interval in milliseconds to be reserved for the initiating job. A time slot is then entered at the end of the time-cycled queue for the user. The time requested should represent maximum program execution time plus a buffer time for higher priority input/output and program swapping. This buffer is recommended as 10% of execution time or one millisecond, whichever is largest.

The sum of reserved intervals must be less than



* This interrupt must occur after time-dependent user service.

** T_j must occur before 0.95 of duty cycle.

Figure 5. Time sequences. *This interrupt must occur after time-dependent user service. **T_j must occur before 0.95 of duty cycle.

0.95 of the duty cycle. Should an excessive interval be specified, an error response will be typed and the user denied entry to the time-cycled queue.

A command TJOB O can be used to release the user's reserved interval but retains protection of the duty cycle. This allows for additional sharing of cyclic operation by two or more cooperating users. Since they require time cycling only during active analog linked computing in simulation, an often small percentage of total use of the computer, users may successfully alternate time-dependent and round-robin operation.

Time Synchronization

While the TJOB command reserves an interval for the user, it does not actually enter his program into cyclic operation. A programmed operator TSYNC must be executed within the user's program to accomplish the transfer. Prior to execution of this instruction, the reserved time interval was used for non-time-dependent functions.

The TSYNC command will result in an error return if it is encountered without a reserved interval being previously set. This includes the condition of zero time reserved with TJOB 0.

When TSYNC is correctly executed, the user's operation is terminated for the remainder of the current duty cycle whether he was operating in time-cycled queue or round-robin. When this user's interval is entered during the next duty cycle, the program begins execution at the instruction following the TSYNC. Thus, the TSYNC command provides for initiating and maintaining time-cycled operation. Termination of timed operation occurs when nonhybrid input/output is attempted or the user signals an exit from program branch or teletype.

While input/output service results in termination of cyclic operation, this monitor controlled service proceeds legally at its lower priority. Automatic return to cyclic execution is possible if the user provides a TSYNC after completing the input/output programmed operator sequence (Table 1).

Hybrid Linkage Assignment

The monitor provides for user assignment and protection of hybrid linkage as provided for any other input/output device. The teletyped command, ENABLE ANAx, allows a user to reserve analog console x for his needs. Once assigned to a specific user, only that job is permitted to access the actual equipment. Only one console can be assigned to a job. A subsequent analog ENABLE, calling for a different console, will result in the release of the prior console as well as enabling the newly specified console.

Hardware assignment of the hybrid linkage takes place whenever the job is selected for operation regardless of which queue contains the job. In this way, round-robin operated debugging can include analog equipment.

The reserved equipment can be released by a corresponding DISABLE ANAx, or just DISABLE, as well as the job canceling KJOB.[4]

CURRENT STATUS

The entire system which includes the digital computer as delivered by Digital Equipment Corporation, the hybrid linkage as designed and assembled by United Aircraft, and the modifications to the standard PDP-6 monitor as implemented by United Aircraft has been operating satisfactorily as described since February 1, 1966.

REFERENCES

1. R. Belluardo, R. E. Gocht and G. A. Paquette, "The Hybrid Computation Facility at United Aircraft Corporation Research Laboratories," *Proceedings*, DECUS (Digital Equipment Computer Users Society), 1963, Maynard, Mass., pp. 261–269 (1964).

2. G. A. Paquette, "Progress of Hybrid Computation at United Aircraft Research Laboratories" *Proceedings*, 1964 Fall Joint Computer Conference.

3. "Programmed Data Processor—6 Handbook," (F-65), Digital Equipment Corporation, Maynard, Mass.

4. "PDP-6 Multiprogramming System Manual," (ACTOO), Digital Equipment Corporation, Maynard, Mass.

HYBRID SIMULATION OF A FREE PISTON ENGINE

R. E. Gagne
Mechanical Engineering Division, National Research Council
Ottawa, Ontario, Canada
and
E. J. Wright
Free Piston Development Company
Kingston, Ontario, Canada

INTRODUCTION

The free piston engine principle is one which has intrigued mechanical engineers for decades. Indeed, the original gas engine of Otto and Langden employed a piston assembly which did not contain the now conventional connecting rod and camshaft arrangement[1] and hence may be considered as an early implementation of the free piston principle. Today, however, the free piston principle is normally applied to a highly supercharged two-stroke compression ignition engine in which two opposed reciprocating pistons are used in a diesel cylinder. These pistons do not transmit mechanical energy but instead pneumatic energy is delivered in the form of high-pressure, high-temperature exhaust gas to a separate power turbine to obtain shaft power.

Such an engine has great performance potential in that it combines the superior thermal efficiency of the diesel cycle and the flexibility of a free turbine drive with a basic mechanical design simplicity in which highly stressed rotating members are absent. Further, unlike the conventional engine arrangement, complete dynamic balance of the pistons is obtained so that the engine is vibrationless, thus eliminating any need for complicated and expensive vibration isolation mountings.

Unfortunately, however, the lack of any mechanical connection to the pistons, other than a simple rack and pinion system to maintain an out-of-phase relationship between the pistons, presents a piston control problem which does not exist in a crankshaft engine. This problem is one of ensuring that the pistons do not exceed certain mechanical design limits irrespective of engine loading. It was principally to investigate various piston control schemes that this simulation of a specific engine was undertaken. The ability to implement any control action without the expense of tooling special components and without the risk of mechanical damage has shown the hybrid simulation to be a valuable design tool.

A number of attempts have been made to simulate the free piston engine in the past[2-4] from which we concluded that neither the analog nor digital computer simulations alone provided the most suitable means of handling the problem. It was felt that the ability of the analog to handle the system dynamics and to allow intimate operator contact with the problem should be teamed with the digital computer's ability to perform the thermodynamic calculations involved; and this hybrid approach seemed a most reasonable way to proceed.

In this discussion we will describe our hybrid

approach to this problem, confining ourselves to a discussion of the simulation itself rather than the results obtained from it. We use it only as a vehicle for exhibiting the power of hybrid simulations of systems of this type.

THE HYBRID COMPUTER

The hybrid computer available to us was certainly modest compared to systems available today. It consisted of an Electronic Associates 131R analog computer, a Control Data (née Bendix) G15D digital computer, coupled together through an Epsco 8 × 8 converter system. The analog equipment has been modified to allow individual mode control of some integrators; all mode relays were replaced with the faster and more dependable reed type, and a scheme for master mode controlling the equipment from the digital computer was implemented. The system is shown in Fig. 1.

The patchable logic used was a small logic trainer, the Digiac 3010, which interfaced to the individual mode control relays of the integrators through relays. The comparators used were analog amplifiers with suitable diode feedbacks.

The limited dynamic performance of the analog equipment required that the simulation be run at a much reduced time scale, which certainly detracted from the operator's 'feel' for the problem. As well, since the digital computer is also slow, the calculation times were quite long, and this calculation speed was further decreased over that possible with machine language programming by using all programs, including interface control, written in ALGO.[5] The slow computing speed was possible only by halting the simulation at both ends of the engine stroke at which time the digital calculations were done.

ENGINE DESCRIPTION

The free piston engine consists of a pair of opposed pistons which are constrained to move out of phase under the influence of gas pressures existing in chambers adjacent to them. The pistons do not transmit mechanical power as in conventional crankshaft engines, but instead serve to provide a

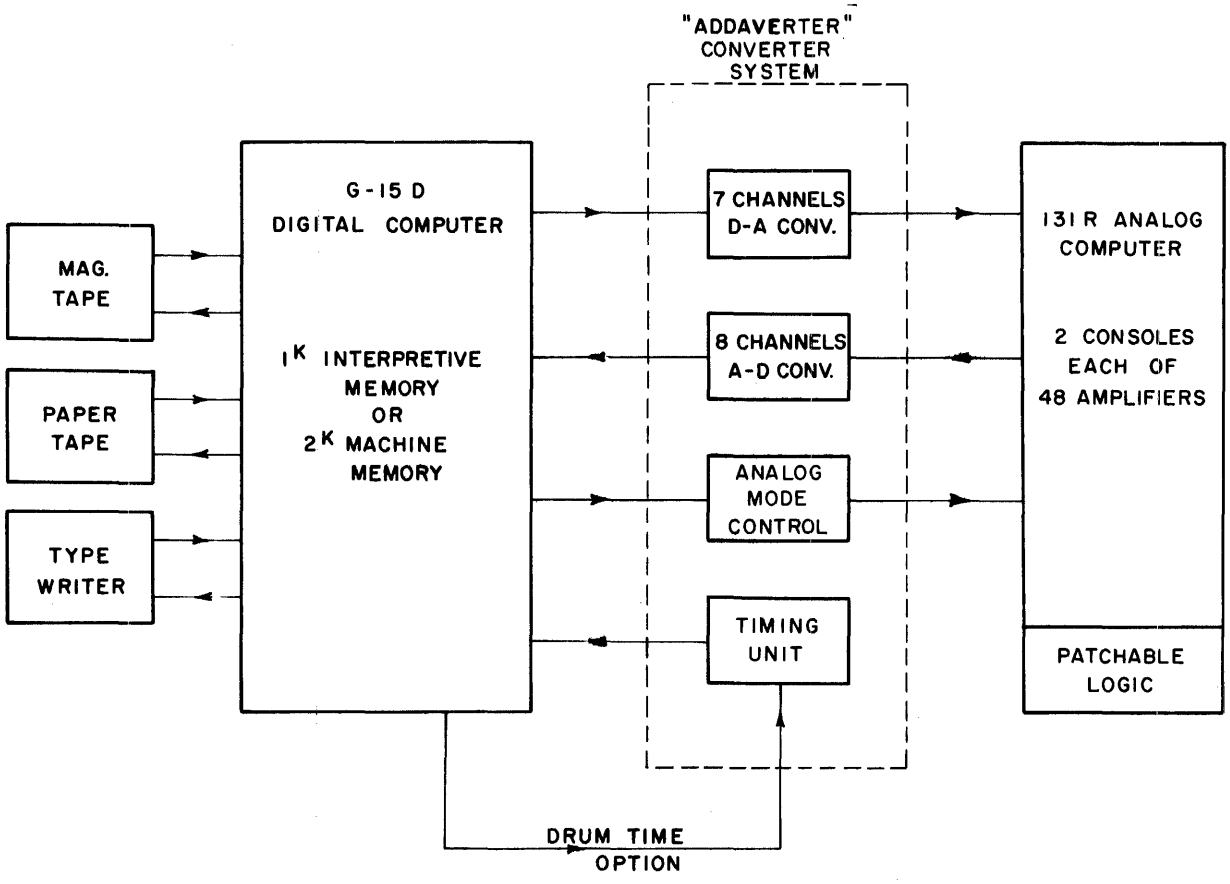


Figure 1. National Research Council's hybrid computer.

cylinder of varying volume in which the chemical energy of liquid fuel may be released in a highly supercharged and efficient diesel cycle to generate hot, high-pressure exhaust gas. This is subsequently passed through a simple gas turbine wheel to obtain mechanical power. Figure 2 is a diagrammatic sketch of an outward compressing machine and it shows one half of the engine. The engine is essentially symmetrical about its centerline, except that it is usual to arrange the exhaust ports and intake ports of the diesel cylinder to be controlled by separate pistons. For convenience, however, these ports are both shown on the same side. Further, because of the symmetry of the engine, it is convenient to simulate the motion of only one of the pistons. The remainder of the discussion is limited to a consideration of this case.

The piston itself serves to divide the cylinders into three separate expansion chambers. The complete cycle of events in each of these chambers is as follows:

1. With the piston close to the engine centerline, fuel is injected into the diesel cylinder and combustion takes place. The resulting high pressure stops the inward movement of the piston at its inner dead point (IDP) and forces the piston to move outward.

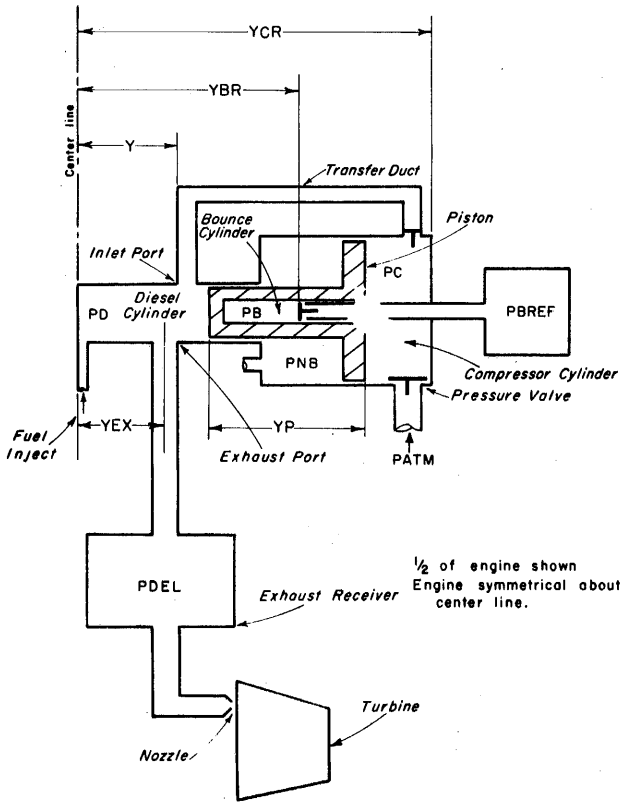


Figure 2. Simplified free piston engine. One half of engine shown. Engine symmetrical about center line.

2. This outward movement of the piston compresses air contained in the compressor and bounce cylinders.

3. As the piston uncovers the exhaust ports, the gas in the diesel cylinder expands through the ports causing a reduction of pressure in the diesel cylinder. This process is normally referred to as the blowdown process.

4. As the piston uncovers the intake ports, the compressor pressure just exceeds the transfer duct pressure and the pressure-sensitive delivery valves open and deliver air to the diesel cylinder via the transfer duct. This air scavenges the exhaust gases from the diesel cylinder, replacing them with fresh air. The mixture of exhaust gas and some scavenge air is stored in the exhaust receiver for subsequent delivery to the turbine at an almost constant rate.

5. Further movement of the piston is resisted by the increasing pressure in the bounce chamber along with piston ring friction forces and it finally comes to rest at its outer dead point (ODP) position.

6. The high-pressure bounce air, coupled with the air trapped in the clearance volume of the compressor chamber, supplies the necessary energy to return the piston to its inner dead point (IDP).

7. On the return stroke, compressor pressure decreases to just less than atmospheric pressure at which point pressure-sensitive suction valves open admitting air from the surroundings to recharge the chamber.

8. As the exhaust ports close, the air trapped in the diesel cylinder is compressed in readiness for the next combustion.

9. The reducing pressure in the bounce cylinder is prevented from falling below a desired minimum value by supplying air from some reference source. Such makeup is required to replace leakage from the bounce cylinder. This cylinder is used as the primary means of controlling piston motion. This completes the cycle.

In the absence of a crankshaft, starting of the engine is accomplished by locking the piston in a position close to the normal outer dead point (ODP) position and charging the bounce cylinder with air at a high pressure. Release of the piston produces a rapid inward movement which compresses air in the diesel cylinder in preparation for injection of the fuel.

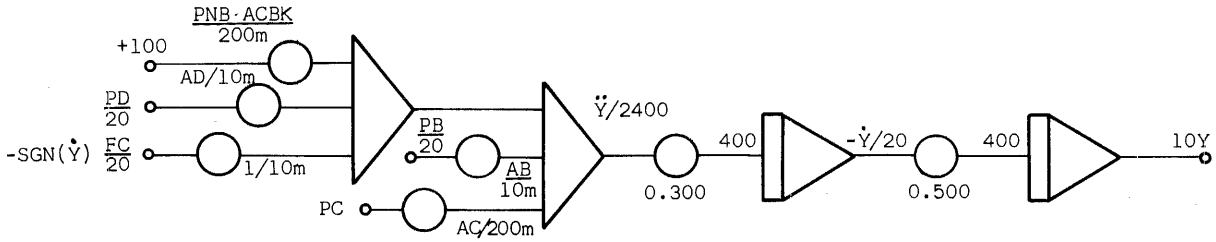


Figure 3a. Analog circuit. Time scale 400:1. Dynamics:

$$\frac{\ddot{Y}}{2400} = PNB \cdot \left(\frac{ACBK}{200m}\right) + \frac{PD}{20}\left(\frac{AD}{10m}\right) - PC\left(\frac{AC}{200m}\right) - \frac{PB}{20}\left(\frac{AB}{10m}\right) - \frac{1}{10m}\left(\frac{FC}{20}\right) SGN(\dot{Y})$$

All pressures in pounds per square inch; all areas in inches squared; mass in slugs; all dimensions in inches.

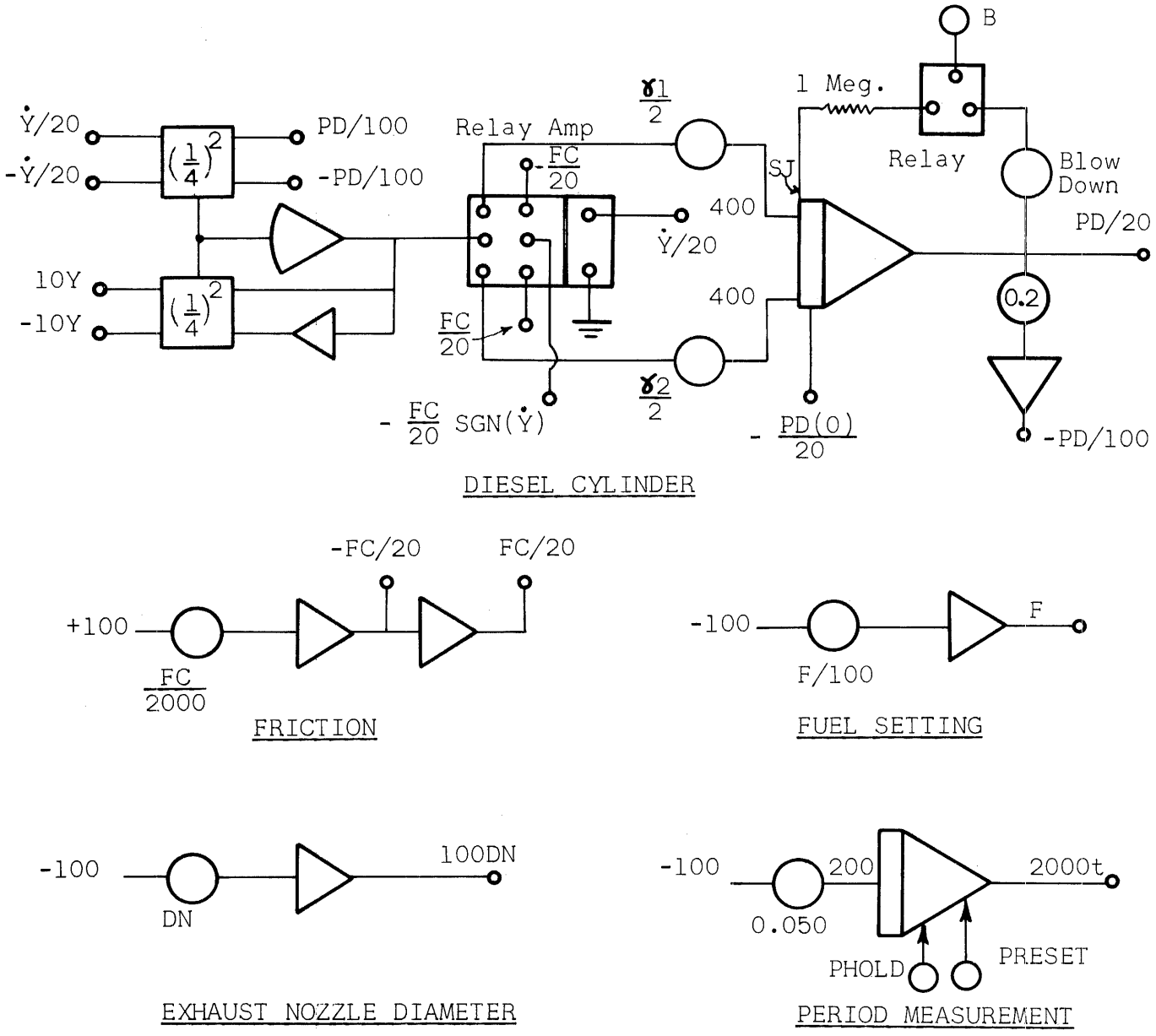


Figure 3b. Analog circuits.

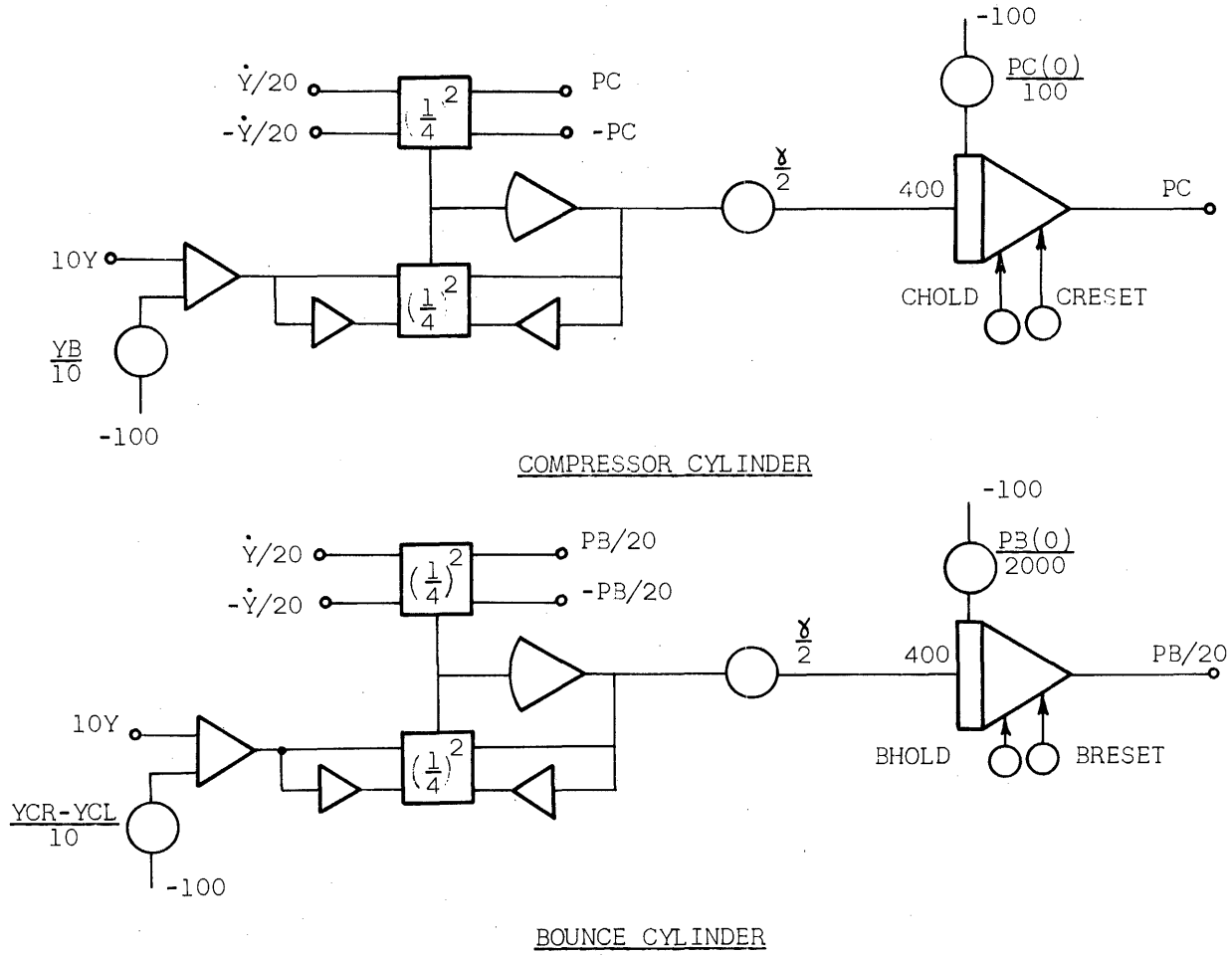


Figure 3c. Analog circuits. Time scale 400:1.

DIESEL CHAMBER EQUATIONS

Solving for the diesel pressure, PD, during the polytropic expansions when the ports are closed, and no combustion is occurring, requires solution of the equation

$$(PD)\,V^n = \text{const}$$

If the expansion and compression processes are assumed to be isentropic, then this equation becomes:

$$(PD)\,y^\gamma = \text{const}$$

where $\gamma = c_p/c_v$ for the gas,
c_p = specific heat at constant pressure,
c_v = specific heat at constant volume, and
y = effective diesel cylinder length uncovered by the piston.

Differentiating this equation, a first-order equation is obtained for diesel pressure

$$\frac{d(PD)}{dt} = \frac{-\gamma(PD)}{y}\frac{dy}{dt} \qquad (1)$$

Although it is known that γ is a function of temperature and hence varies in magnitude during the expansion and compression processes, a suitable mean value is used to simplify the simulation. Such approximations are common in engine calculations and are normally quite acceptable.

Comparator circuits and patchable logic are used to detect when isentropic expansion or compression should be taking place, i.e., when Eq. (1) is valid, and the integrator of Figure 3b which generates (PD) from Eq. (1) is kept in the hold mode at all other times by this logic. Since the inward isentropic compression is of air, and the outward isentropic expansion is of hot gas, both at quite different mean temperature levels, a separate value of γ is used in each case.

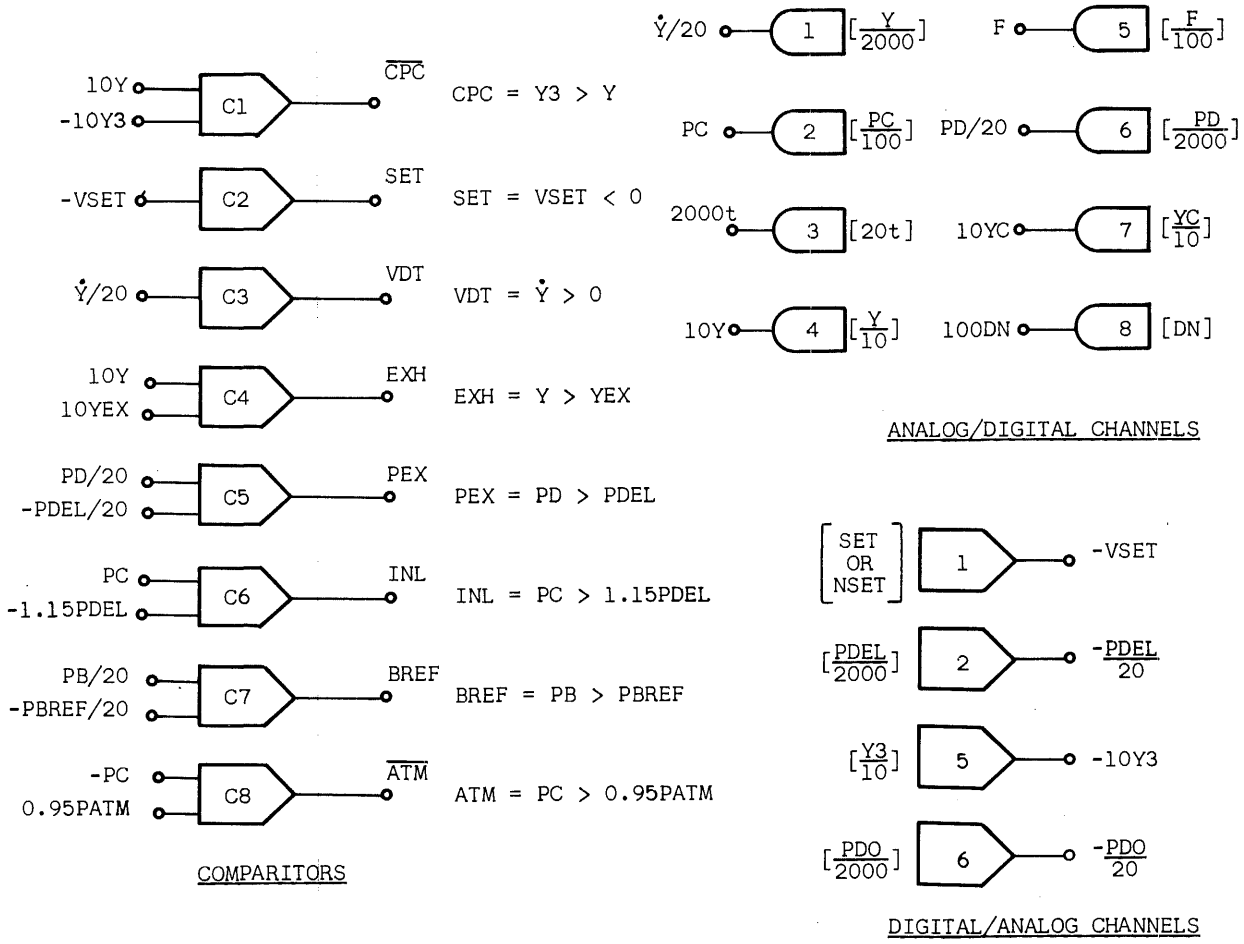


Figure 4. Comparitors and converters.

The assumed indicator diagram for the diesel cylinder is shown in Fig. 7a. As seen in the program flow chart of Fig. 6, the simulation is halted at the IDP and ODP piston positions. At the IDP position, combustion occurs which causes an instantaneous pressure rise followed by a constant pressure expansion before isentropic expansion occurs.

When the exhaust ports open, the diesel pressure falls to the exhaust receiver pressure, and this drop is assumed to occur exponentially.

COMPRESSOR CHAMBER EQUATIONS

By similar arguments, the isentropic portions of the compressor cycle are described by solving

$$\frac{d(PC)}{dt} = \frac{-\gamma(PC)}{y_c}\frac{d(y_c)}{dt} \qquad (2)$$

where PC = compressor pressure and y_c = effective compressor chamber length uncovered by the piston.

Again, suitable logic is used to detect when Eq. (2) is valid and the circuit of Fig. 3c results. The compressor chamber indicator diagram is shown in Fig. 7b. On the inward stroke, isentropic expansion of the clearance volume air occurs until the compressor pressure is just less than atmospheric pressure, causing the suction valves to open and admit air. For the remainder of this inward stroke, PC is constant and is given by

$$PC = 0.95\, P_{atm}$$

where the factor accounts for suction valve pressure drop.

On the outward stroke, isentropic compression occurs until the compressor delivery valve leading to the transfer duct is open, at which time we have

$$PC = 1.13\, P_{del}$$

where P_{del} = exhaust receiver pressure, and the factor accounts for pressure drops through the

compressor delivery valves and the intake and exhaust ports of the diesel.

Provided the exhaust receiver is large compared with the swept volume of the diesel cylinder (of the order of 3 times), it can be assumed that P_{del} is constant over that portion of the cycle when the exhaust port is open, without sacrificing accuracy. (In the actual engine, P_{del} changes in the order of 5% over a complete engine cycle.)

BOUNCE CHAMBER EQUATIONS

The bounce chamber indicator diagram is shown in Figure 7c. This chamber acts essentially as a pneumatic spring in which leakage from the chamber results in a pressure loss during the cycle. This leakage, which occurs past the piston sealing rings, is proportional to the instantaneous bounce pressure in the chamber. Thus the equation to be solved is:

$$\frac{d(PB)}{dt} = \frac{-\lambda(PB)}{y_b}\frac{d(y_b)}{dt} - \alpha(PB) \qquad (3)$$

where PB = bounce chamber pressure,
y_b = effective bounce chamber length uncovered by the piston, and
α = leakage factor.

When, due to leakage, the bounce pressure drops to below a desired bounce reference pressure, P_{BREF}, the bounce chamber is connected to a reference pressure source by a pressure sensitive valve to hold the minimum bounce chamber pressure constant.

The analog circuit generating PB is shown in Fig. 3c.

PISTON DYNAMIC EQUATIONS

The forces acting on the piston are due to the pressures in the various chambers along with piston ring friction. For dimensions in inches, piston mass (m) in slugs, and pressure in pounds per square inch, the equation describing piston motion is:

$$\ddot{y}\left(\frac{m}{12}\right) = P_{NB}A_{CBK} + \left(PD\right)A_d - \left(PC\right)A_c - \left(PB\right)A_b - F_c \operatorname{sign}(\dot{y}) \qquad (4)$$

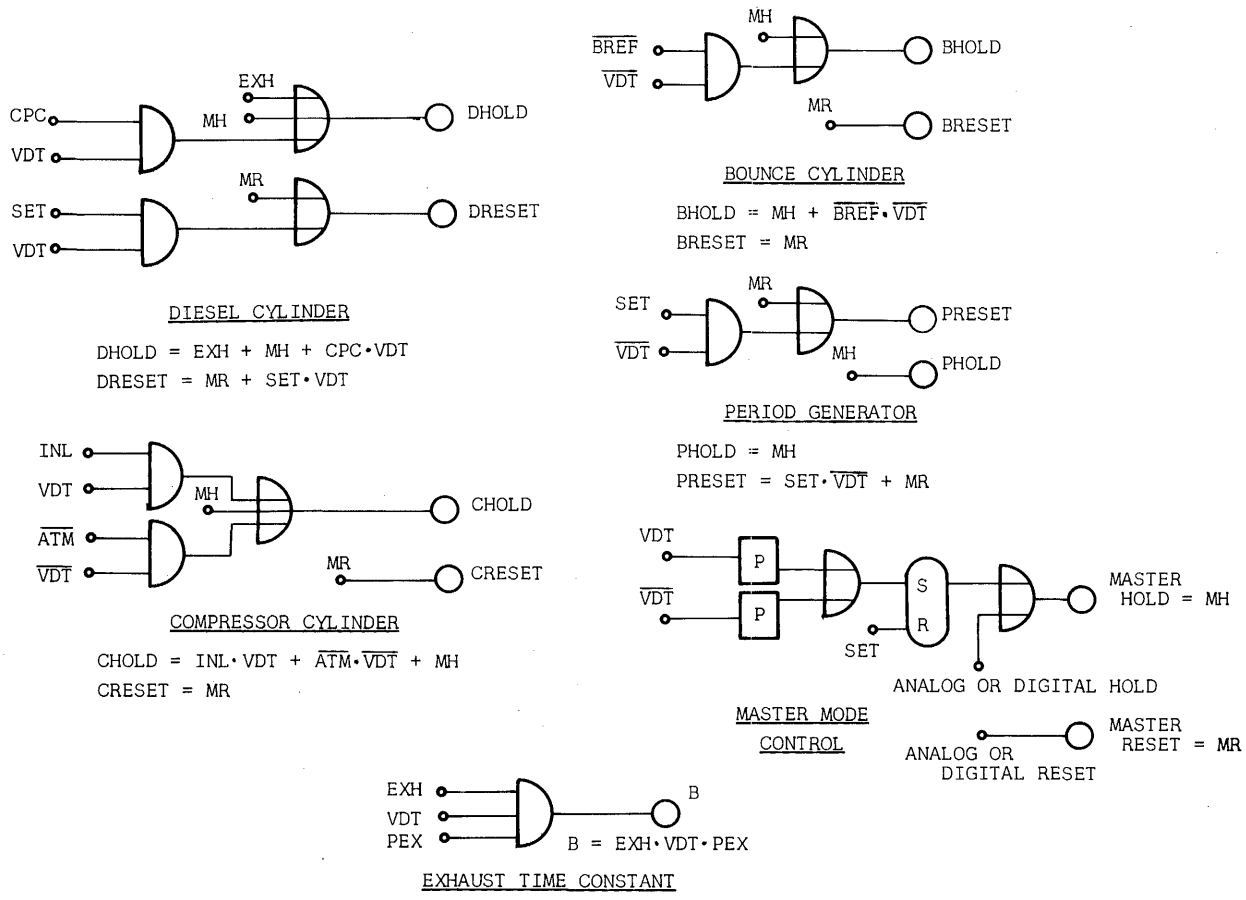


Figure 5. Control logic.

where P_{NB} = negative bounce chamber pressure (constant),
A_{CBK} = cross-sectional area of negative bounce piston surface,
A_d = cross-sectional area of diesel piston surface,
A_c = cross-sectional area of compressor piston surface,
A_b = cross-sectional area of bounce piston surface, and
F_c = frictional force (assumed constant).
The scaled equations and analog circuit are shown in Fig. 3a.

MODE SWITCHING LOGIC

The logic signals necessary to control the simulation are obtained by use of the comparators of Fig. 4. The mode switching for the compressor and bounce chambers are relatively straightforward, and the logic is shown in Fig. 5. The diesel pressure circuit is perhaps less so, and will be used as an example of how the mode switching conditions are obtained.

From Fig. 7a, and starting at the ODP position, the diesel pressure is equal to that existing in the exhaust receiver (i.e., P_{del}) and remains at this pressure until the exhaust port is closed. Thus the diesel integrator is in the hold mode whenever the following is true:

$$\overline{\text{VDT}} \cdot \text{EXH} \qquad (5)$$

When EXH is no longer true (port is closed), the pressure increases adiabatically until IDP is reached; i.e., until the velocity changes sign. This sign change is used to put the whole simulation in hold while the combustion calculations are made. Thus condition (5) must be OR'ed with master hold (MH). The new diesel pressure at IDP resulting from the constant volume portion of the combustion (as described in the Appendix) is established as an initial condition input to the diesel pressure integrator through converters, and this integrator is placed in reset by use of a SET signal from the digital computer. Since this SET signal is also used at ODP to reset the period measuring integrator, and to allow for engine starting, the reset condition on the diesel integrator is given by:

$$\text{DRESET} = \text{SET} \cdot \text{VDT} + \text{MR}$$

where MR is the master reset condition.

If some constant pressure combustion, as described in the Appendix, is to take place, then the voltage y_3 is output to the CPC comparator, and CPC is true during this constant pressure portion of the expansion. Thus the integrator is also in hold whenever the following is true:

$$\text{CPC} \cdot \text{VDT} \qquad (6)$$

When CPC is no longer true, Eq. (1) is integrated to give the isentropic expansion until the exhaust port is again opened, at which time the pressure decreases exponentially to the exhaust receiver pressure. This exponential pressure decrease is achieved by placing the diesel integrator in hold and, at the same time, switching a resistor around the integrating capacitor until the delivery pressure P_{del} is reached. This is achieved by a relay driven by the logic signal B where

$$\text{B} = \text{EXH} \cdot \text{VDT} \cdot \text{PEX}$$

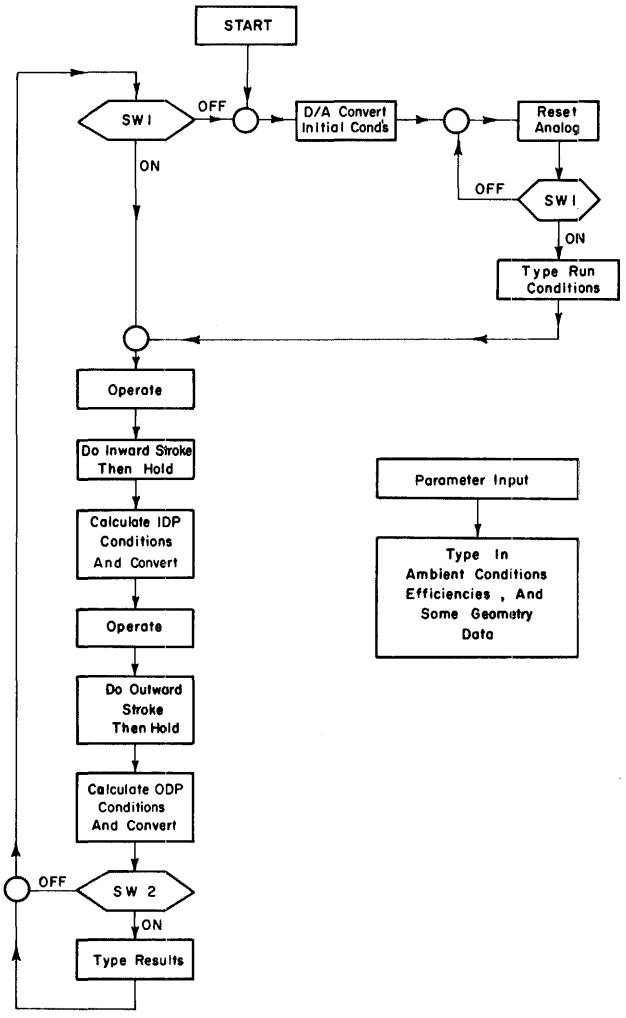
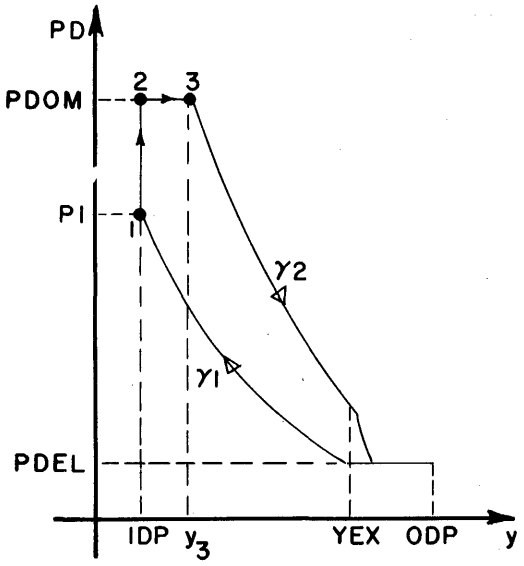
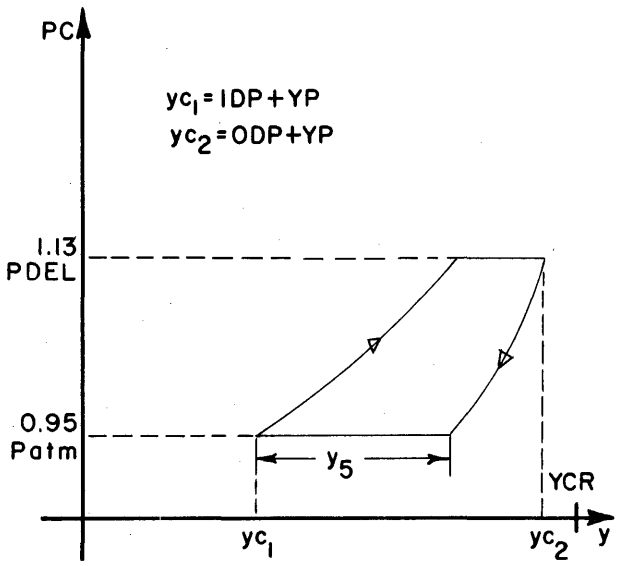


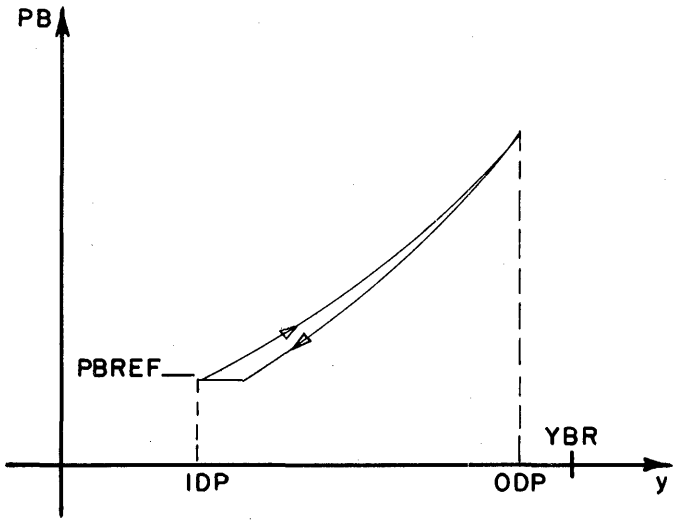
Figure 6. Simplified program flow chart.



7(a) DIESEL CYLINDER



7(b) COMPRESSOR CYLINDER



7(c) BOUNCE CYLINDER

Figure 7. Indicator diagrams.

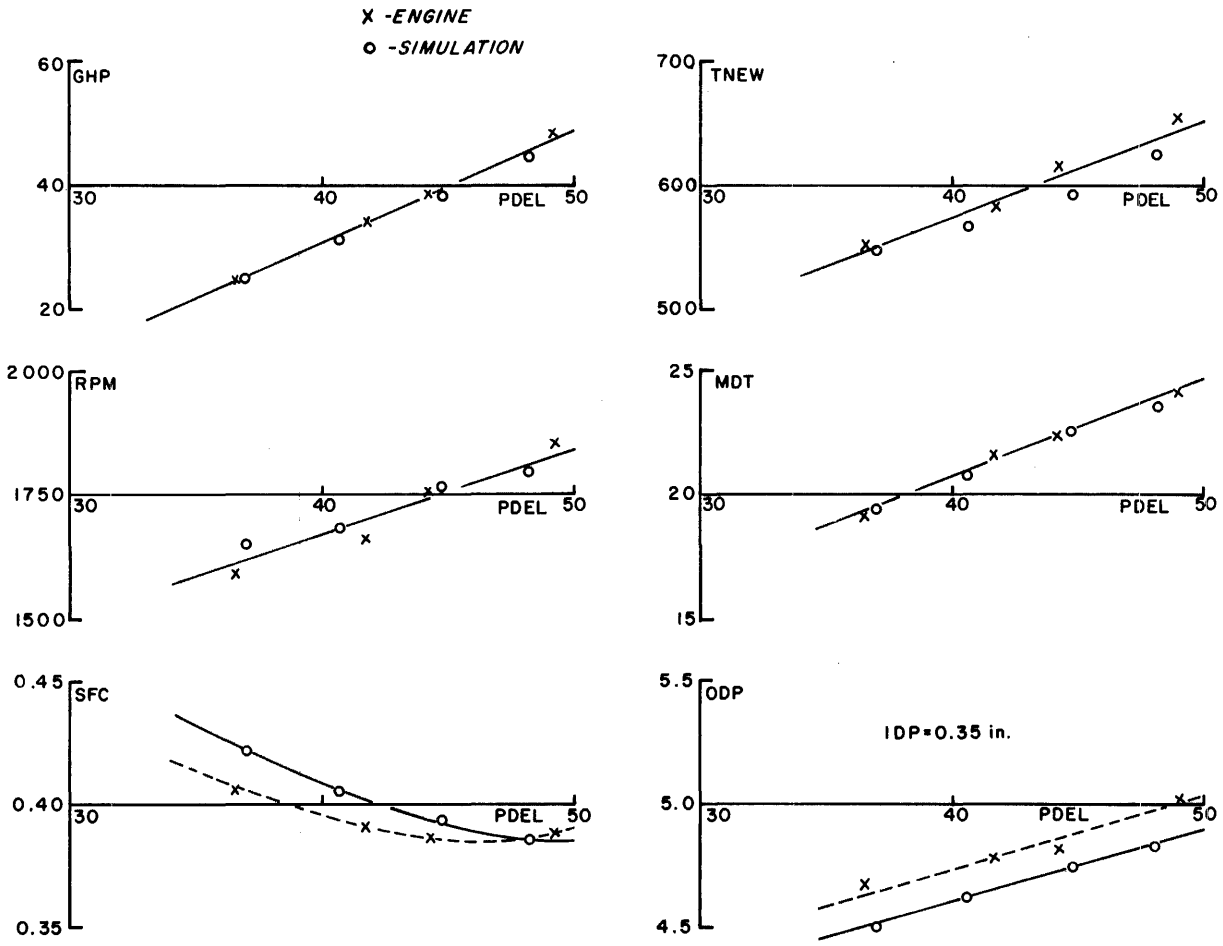


Figure 8. Engine and simulation comparison.

The complete diesel integrator hold condition is given by:

$$DHOLD = EXH + VDT \cdot CPC + MH$$

DIGITAL PROGRAM

A simplified flow chart for the digital program is shown in Fig. 6. The program is entered, and initial conditions are established on the analog console for proper starting under specified environmental conditions. Problem running is controlled by a breakpoint switch (No. 1) which, at any time, can return the simulation to the starting conditions. The engine is started with the piston at the ODP position. After the inward stroke, the program causes the analog console to go to hold at the IDP position after which the combustion calculations of the Appendix are performed. The program then releases the analog to operate until it is again put to hold at the ODP position for the mass flow calculations. Monitoring information type-outs are controlled by a breakpoint switch (No. 2).

RESULTS

The performance data available from actual measurements on a specific engine are shown in Fig. 8. The geometry data and ambient environmental conditions were established on the simulation, and the combustion efficiencies, friction coefficient, leakage and heat transfer coefficients were calculated based on measurable quantities and estimates of the engine properties at the time the results were obtained. These quantities were further adjusted by a few percent to obtain the comparisons shown in Fig. 8.

Since it is the purpose of the simulation to facilitate the design of control hardware to control the engine under transient conditions, some means of evaluating the simulation under transient conditions would have been most useful. As can be appreciated, however, transient behavior of the engine would involve measurements which are extremely difficult if not impossible to obtain, and it was possible to make only qualitative comparisons between the engine and simulation under these conditions. Figures 9 and 10 are examples of the simulation data which allowed these qualitative comparisons to be made. In particular, the transient during start-up, when an operator familiar with the engine also attempted simulated runs, allowed us to conclude that the simulation was adequate for the purposes of the intended control studies.

Appendix

DIGITAL CALCULATIONS

The Combustion Model

The diesel cycle assumed is a modified form of the limited pressure cycle. This cycle assumes that heat release occurs in two distinct phases:

1. At constant volume, the pressure rising to a defined maximum pressure;
2. At constant pressure, the volume increasing until all the remaining fuel energy is expended.

To describe the combustion process the following assumptions are made.

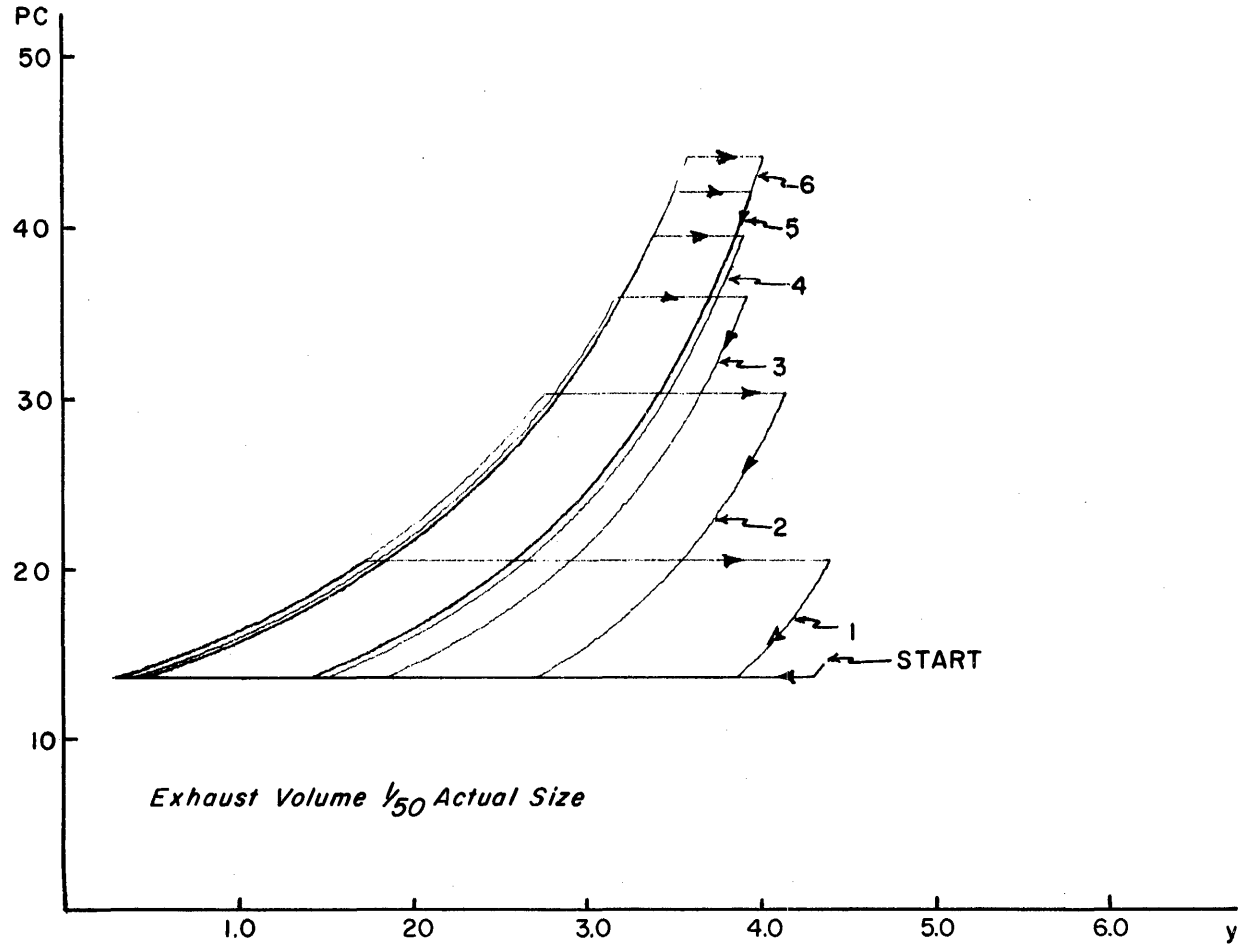


Figure 9. Compressor start-up transient.

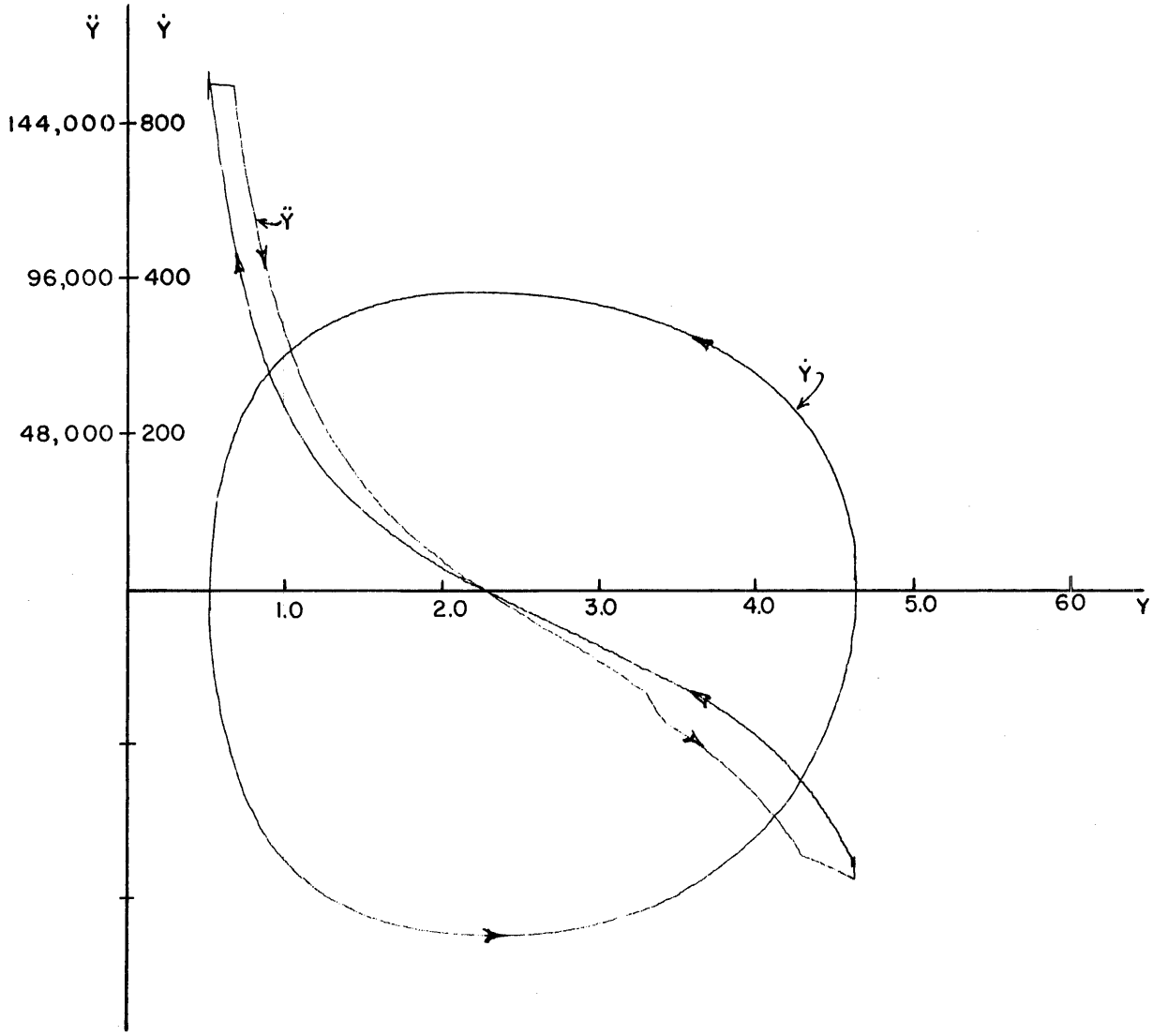


Figure 10. Velocity and acceleration.

- The gas is ideal obeying $PV = mRT$.
- The specific heats for constant volume C_v, and constant pressure C_p, are constant and related by

$$C_p - C_v = R/J$$

where J = Joule's constant.

- All losses due to combustion inefficiency, leakage, and heat transfers can be contained in a single constant efficiency factor η_c.

Under these conditions, and referring to Fig. 7a, an energy balance between states (1) and (3) for an amount of fuel F of calorific value k is given by:

$$F\,k\,\eta_c = mC_v(T_2 - T_1) + mC_p(T_3 - T_2)$$

Using the ideal gas law, the above can be reduced to

$$y_3 = (\mathrm{IDP}) + \frac{1}{P_2}\left[k_5 F - \frac{(\mathrm{IDP})(P_2 - P_1)}{\gamma}\right] \quad \text{(A.1)}$$

where $k_5 = \dfrac{k\,\eta_c\,R}{c_p A_d}$. This equation forms the basis for the assumed combustion process, and is used as follows:

1. If $P_1 >$ PDOM, where PDOM is some defined pressure, then all combustion is

assumed to occur at constant pressure and (A.1) reduces to

$$y_3 = (\text{IDP}) + k_5 F / P_1$$

2. If $P_1 <$ PDOM, then P_2 is calculated assuming all the fuel is burnt at constant volume. If the resulting calculated pressure is less than PDOM, then no constant pressure combustion is achieved, i.e. $y_3 = 0$. If this calculated pressure is greater than PDOM, then P_2 is set equal to PDOM, and the resulting y_3 is calculated using (A.1).

Exhaust Mass Flow Model

The engine performance is measured in terms of the exhaust gas mass flow and its mean temperature and pressure. These properties are determined by applying 1) the continuity equation, 2) the energy equation, and 3) the turbine nozzle characteristic equation to a simple model of the gas exchange process.

The gas delivered to the receiver is given by

$$M_{\text{del}} = k_1 y_s \tag{A.2}$$

where $k_1 = R_L \rho_s A_c$,
y_s = stroke length during which the compressor suction valves are open,
ρ_s = atmospheric air density, and
R_L = leakage factor.

The delivery temperature of the exhaust gas is determined from an energy balance of the complete engine. This results in the equation

$$T_{\text{del}} = T_{\text{amb}} + \frac{k_2 F}{k_1 y_s} \tag{A.3}$$

where $k_2 = \frac{\eta_c k}{c}$,
c = average specific heat,
F = cyclic fuel input, and
T_{amb} = ambient temperature.

Solution of the actual gas transfer dynamics during gas exchange was prohibited by the lack of sufficient hardware to cope with the complex, nonlinear equations describing these processes. Instead, a simple model was assumed in which delivery of gas to the receiver occurs instantaneously at ODP. This assumption, though evidently unrealistic, is not considered to materially affect the results, particularly on a cycle basis.

Delivery of the exhaust pulse to the receiver results in the equations

$$\begin{aligned} M_{\text{new}} &= M + M_{\text{del}} \\ T_{\text{new}} &= \frac{MT + M_{\text{del}} T_{\text{del}}}{M_{\text{new}}} \end{aligned} \tag{A.4}$$

from which

$$P_{\text{new}} = \frac{M_{\text{new}} R T_{\text{new}}}{V} \tag{A.5}$$

where M = mass in receiver immediately before delivery of the pulse,
T = temperature of the gas in the receiver, immediately before delivery of the pulse,
R = gas constant, and
V = receiver volume.

The mass flow rate leaving the receiver is governed by the characteristic equation of the nozzle.

This mass flow rate, $\dot{M}$, can occur governed by

1) subcritical flow: $\left[\frac{P_{\text{atm}}}{P}\right] > \left(\frac{2}{\gamma + 1}\right)^{\frac{\gamma}{\gamma - 1}}$ (A.6)

2) critical flow: $\left[\frac{P_{\text{atm}}}{P}\right] \leq \left(\frac{2}{\gamma + 1}\right)^{\frac{\gamma}{\gamma - 1}}$ (A.7)

where subcritical flow is given by

$$\dot{M} = D_N^2 P \left\{ \frac{\pi}{4} \sqrt{\frac{2g}{R} \left(\frac{\gamma}{\gamma - 1}\right) \cdot \frac{1}{T} \cdot \left(\frac{P_{\text{atm}}}{P}\right)^{\frac{2}{\gamma}} \left[1 - \left(\frac{P_{\text{atm}}}{P}\right)^{\frac{\gamma - 1}{\gamma}}\right]} \right\} \tag{A.8}$$

and critical flow by

$$\dot{M} = D_N^2 P \left\{ \frac{\pi}{4} \sqrt{\frac{2g}{R}} \cdot \sqrt{\frac{1}{T} \left(\frac{\gamma}{\gamma + 1}\right) \left(\frac{2}{\gamma + 1}\right)^{\frac{2}{\gamma - 1}}} \right\} \tag{A.9}$$

where D_N = nozzle diameter,
g = gravitational acceleration, and
P_{atm} = atmospheric pressure, to which nozzle is discharging.

If the receiver volume V is large, it can be assumed that the mean receiver pressure during one

cycle is given by

$$P_{\text{mean}} = \frac{P_{\text{new}} + P_{\text{del}}}{2} \tag{A.10}$$

where P_{del} = pressure in receiver immediately before next pulse.

Calculating $\dot{M}$ from (A.7), (A.8) and (A.9), where $P = P_{\text{mean}}$, $T = T_{\text{new}}$, the remaining mass is

$$M = M_{\text{new}} - \dot{M}P_{\text{erd}} \tag{A.11}$$

and

$$P_{\text{del}} = \frac{MRT_{\text{new}}}{V} \tag{A.12}$$

where P_{erd} = cyclic period of the piston.

These equations are solved iteratively for each cycle to obtain the corresponding nozzle mass flow $\dot{M}$, mean pressure P_{mean} and temperature T_{new}.

Gas Horsepower and Specific Fuel Consumption

The gas horsepower is defined as the horsepower available from exhaust gas when expanded through an ideal turbine. Thus:

$$\text{GHP} = k_4 \dot{M} T_{\text{new}} \left[1 - \left(\frac{P_{\text{amb}}}{P_{\text{mean}}} \right)^{\frac{\gamma - 1}{\gamma}} \right] \tag{A.13}$$

where $k_4 = c_p J/550$. The corresponding specific fuel consumption is

$$\text{SFC} = \frac{3600\,F}{P_{\text{erd}}(\text{GHP})} \text{ lb/GHP hr}$$

ACKNOWLEDGMENTS

The authors are indebted to the Engine Laboratory and Analysis Section of the Mechanical Engineering Division of the National Research Council for providing a helpful and critical atmosphere during all phases of this work. In particular we would like to thank R. F. Henry, K. C. Cowan and M. Grovum who did the early analysis work, a preliminary analog study and a parallel digital simulation, and E. P. Cockshutt who fielded many awkward questions regarding engine characteristics.

REFERENCES

1. R. G. Fuller, "Free-Piston Gas Turbines and Their Application to Steam Power Plants," Canadian Power Show, Dec. 7–9, 1961, Toronto.

2. D. R. Olson, "Simulation of a Free-Piston Engine with a Digital Computer," *SAE Trans.*, vol. 66 (1958).

3. A. R. Bobrowsky, "Analytical Methods for Performance Estimates of Free-Piston Gasifiers," ASME Gas Turbine Power Conf., Detroit, Mich., Mar. 18–21, 1957.

4. D. A. Trayser, discussion on Ref. 2 above, *SAE Trans.*, vol. 66, pp. 681–682 (1958).

5. "ALGO Programming Manual for the Bendix G15 Computer," available from Control Data Corp. (Mar. 1961).

6. F. J. Wallace, E. J. Wright and J. S. Campbell, "Future Development of Free Piston Gasifier Turbine Combinations for Vehicle Traction," *Proc.*, Automotive Engineering Congress, Detroit, Mich., Jan. 10–14, 1966, paper 660132.

HYBRID ANALOG/DIGITAL TECHNIQUES FOR SIGNAL PROCESSING APPLICATIONS

Thomas G. Hagan and Robert Treiber
Adage, Inc., Boston, Massachusetts

INTRODUCTION

This paper describes a number of hybrid techniques, most of them developed as applications for the AMBILOG 200 Stored Program Signal Processor. They are described in fairly general terms, since our purpose is to elucidate the techniques, rather than to describe specific applications or detailed methods of implementation.

The signal processing operations here considered involve the use of analog and hybrid operators which operate simultaneously upon a number of analog and digital operands. A stored program controller controls the array of parallel operators and provides for the transfer of operands back and forth between the operator array and core memory. The AMBILOG 200 system is specifically organized to perform these control and transfer operations efficiently. Since its organization for these purposes has been described in a previous paper,[1] we shall not discuss it further here.

In many signal processing applications, time is of the essence. The term "signal" itself, as we use it here, implies that the dimension of time is significant. The signals operated upon by the various processing techniques described below are often in analog form at some point. Although ease of accommodating both analog and digital signals as inputs and outputs is indeed an important virtue of the hybrid approach, its validity depends upon the amount of processing it affords per unit time per unit cost, not upon whether the signals to be processed happen to occur in analog form. Indeed, in many of those cases where hybrid techniques are appropriate, both input and output data are in digital form.

As we shall see in several instances below, the performance advantage afforded by the hybrid technique (as compared with an all-digital approach using a conventional sequential stored program system) results from the high degree of parallelism achievable at relatively low cost. For a technique involving use of multiple parallel arithmetic operators in conjunction with a sequential core memory, a very rough "figure of merit" is simply the number of arithmetic operations accomplished per fetch of an operand from memory. In the operation of a conventional highly sequential digital computer, a maximum of one arithmetic operation per fetch of an operand from memory is usually the limit. In the hybrid techniques we discuss below, many arithmetic operations can usually be accomplished per memory fetch—5, 10, or even 100 or more.

WAVEFORM MEASUREMENT

We use the term waveform measurement to signify the measurement of various waveform parameters based upon axis crossings, peak values, slope maxima and minima, discontinuities, and other such singularities. The effectiveness of hybrid techniques for such measurements is illustrated by a

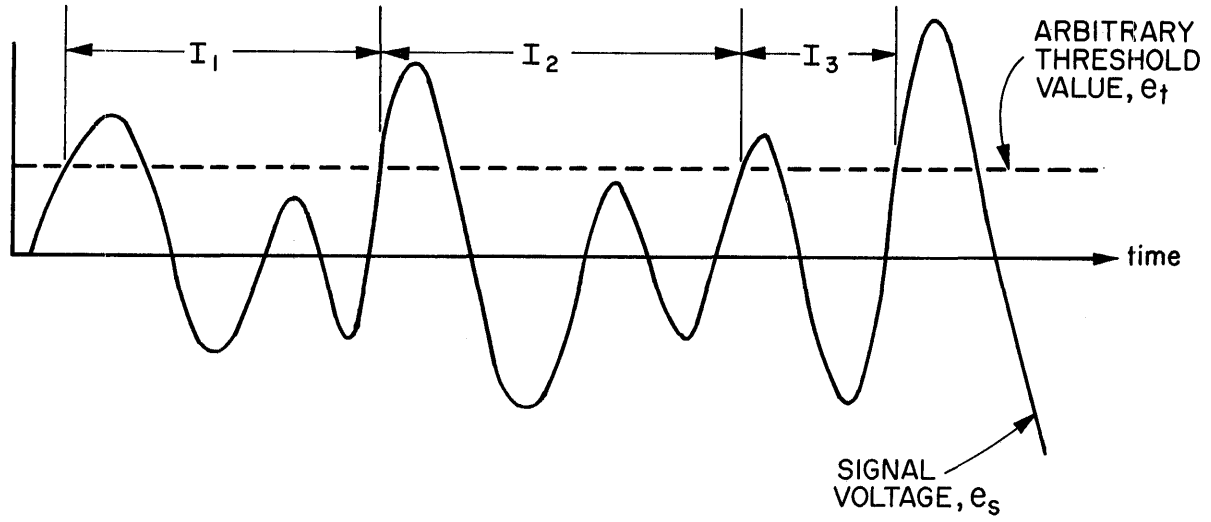


Figure 1. Time intervals I between threshold crossings of a signal.

method that has been used for the determination of histograms of threshold crossing intervals of a speech waveform. The problem (see Fig. 1) is to monitor an analog waveform derived from a microphone, to measure the time intervals between successive positive-going crossings through an arbitrary programmable threshold value, and, by counting the number of intervals that occur for each of a number of different interval lengths, to develop a histogram (in memory) which can be plotted to show number of intervals versus interval length.

The measurement of interval length could be accomplished by connecting an A-D converter to a digital computer and executing a program that included steps necessary to increment a counter, perform an A-D conversion, transfer the result from the A-D converter into the computer, subtract the threshold value from the A-D converter result, and then test for sign of the remainder. Resolution of the interval length measurement would be limited by the length of time required to execute these steps.

A hybrid technique that has been used to accomplish this operation with fewer program steps is shown in Fig. 2. The threshold value is transferred to a DAC, whose output is thenceforth subtracted by analog means from the analog input signal. The difference signal feeds an axis crossing detector, whose output is monitored by a sense line testable by the program. The "inner loop" which limits resolution of the interval measurement need only

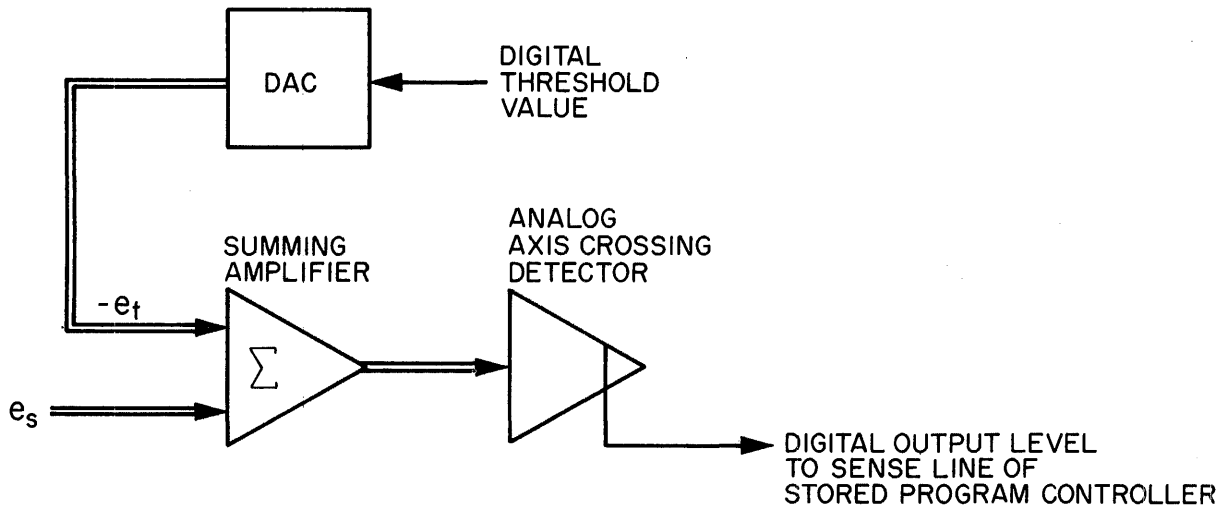


Figure 2. Hybrid technique for measurement of time intervals between threshold crossings.

contain program steps for testing the sense line, incrementing a counter, and closing the loop. A program written to implement this technique, which also provided for testing the counter and exiting from the inner loop when the count reached a prescribed limit, resulted in interval measurement resolution of less than 16 microseconds.

Waveform measurement techniques of this general type have been successfully applied to the real-time analysis of multichannel physiological signals including electrocardiograms (ekg), vascular pressures, and breathing waveforms.[2]

where x_1 through x_n comprise a set of input signals, y_1 through y_n are output signals, and a_{00} through a_{mn} are coefficients in the transformation matrix. Each output signal y is achieved by summing the appropriately weighted input signals x.

A hybrid technique for accomplishing this transformation is illustrated in Fig. 4. Digital coefficients can be loaded under stored program control into storage registers contained in the hybrid multiplying elements (HME's).

The HME's thenceforth provide the weighting or multiplication function by multiplying the analog

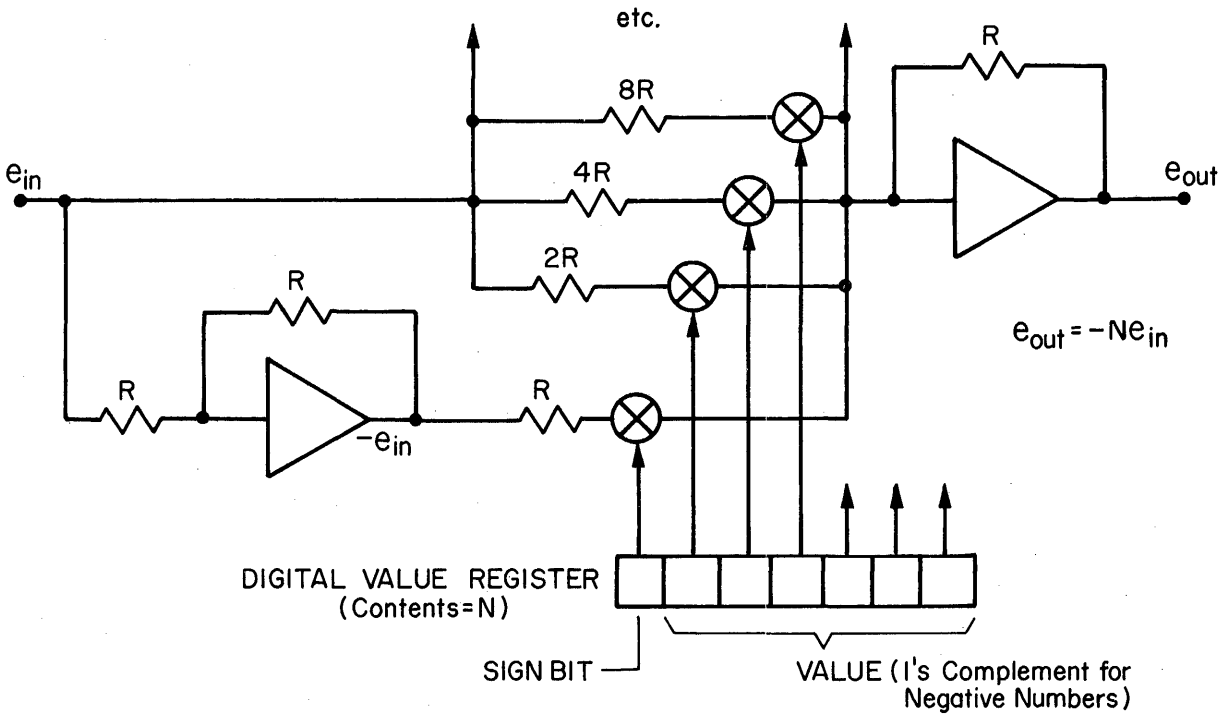


Figure 3. A hybrid multiplying element (HME) for multiplying an analog signal by a digital coefficient.

LINEAR TRANSFORMATION

Numerous measurement and simulation problems require a signal processing technique to implement the linear transformation implied by the equations:

$$y_1 = a_{00} + a_{01}x_1 + a_{02}x_2 \cdots + a_{0n}x_n$$

$$\vdots$$

$$y_n = a_{m0} + a_{m1}x_1 + a_{m2}x_2 \cdots + a_{mn}x_n$$

input by the stored coefficient. The weighted outputs are analog signals which can be summed with summing amplifiers. The diagram of an HME is shown in Fig. 3.

The hybrid technique achieves a high "figure of merit" for the linear transformation operation in that once the coefficients of the transformation matrix have been fetched from memory and loaded into the array of DAC's and HME's, the parallel array can perform the equivalent of a very large number of multiply and add operations as the input analog signals vary.

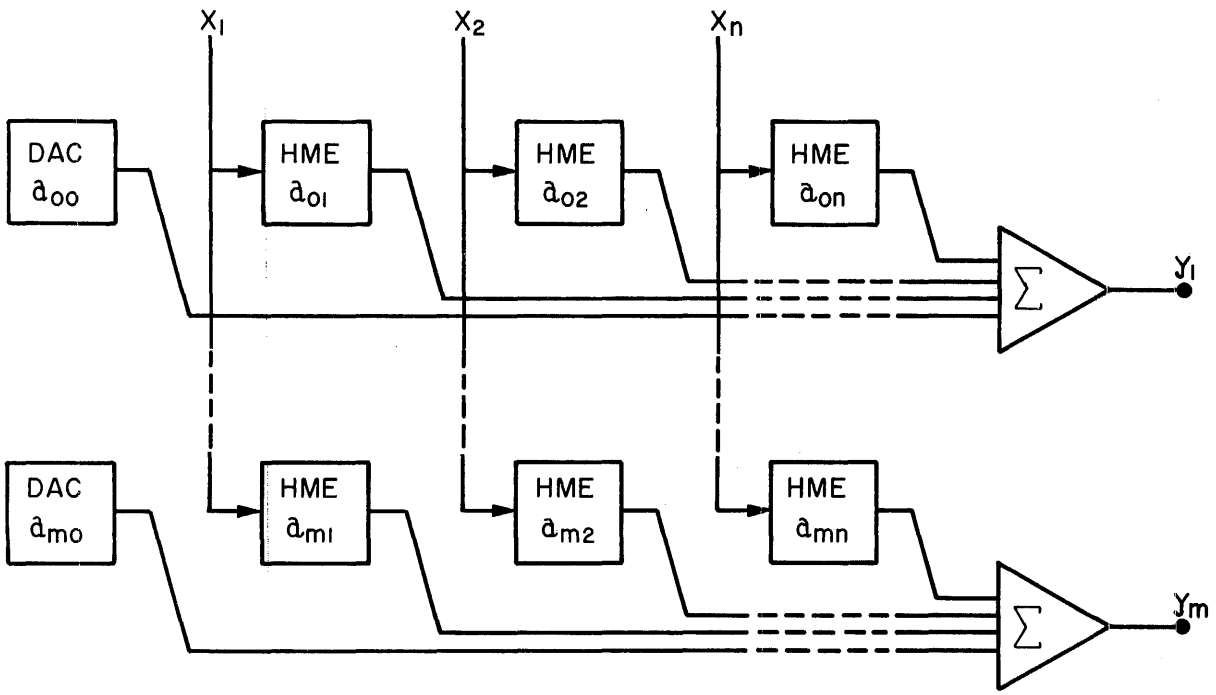


Figure 4. A hybrid array for performing linear transformations.

SPATIAL COORDINATE TRANSFORMATION

The array shown in Fig. 5 can be used to accomplish transformation of spatial coordinates. The three analog input signals x,y,z represent the coordinates of a point in one coordinate system; the three outputs x',y',z' represent the coordinates of the same point in a different coordinate system, displaced and rotated with respect to the first one.

Motion of the coordinate systems with respect to each other entails updating of the transformation matrix to reflect changes in displacements and angles between the two systems. For a given set of values in the transformation matrix, the analog inputs are transformed into appropriate analog outputs, and as the inputs vary, the outputs vary.

VISUAL DISPLAY

The coordinate transformation method described above can be used for performing translation and rotation of objects being displayed on a CRT. A given three-dimensional object can be represented by the coordinates of a set of points in memory. An isometric view of the object can be generated by calling the points from memory one by one, subjecting them to a coordinate transformation, and generating a picture by connecting straight lines between points. The viewpoint from which the object is seen can be altered by changing the coefficients of the coordinate transformation to which each point is subjected as it is called from memory. Coefficients of the transformation matrix remain constant throughout the process of generating one complete view (frame) of the object. For viewing a visual display on a CRT, a frame repetition rate of about 30 frames per second is desirable. Using a vector generator that develops a line segment connecting two points in 30 microseconds, it is possible to depict objects representable by drawings consisting of about 1000 straight line segments at a 30-frames-per-second rate.

Arbitrary rotation and translation of the object viewed requires that the necessary arithmetic operations to accomplish the coordinate transformation be performed for each point in turn. A hybrid array suitable for performing the operations required for developing an isometric projection of a solid object onto a plane is shown in Fig. 6. The output variables x' and y' are developed from the input coordinates x, y, z by operating upon the inputs with a transformation matrix consisting of coefficients loaded into the operator array. Coefficients of the transformation matrix are computed once per frame and loaded into the array. Thenceforth the

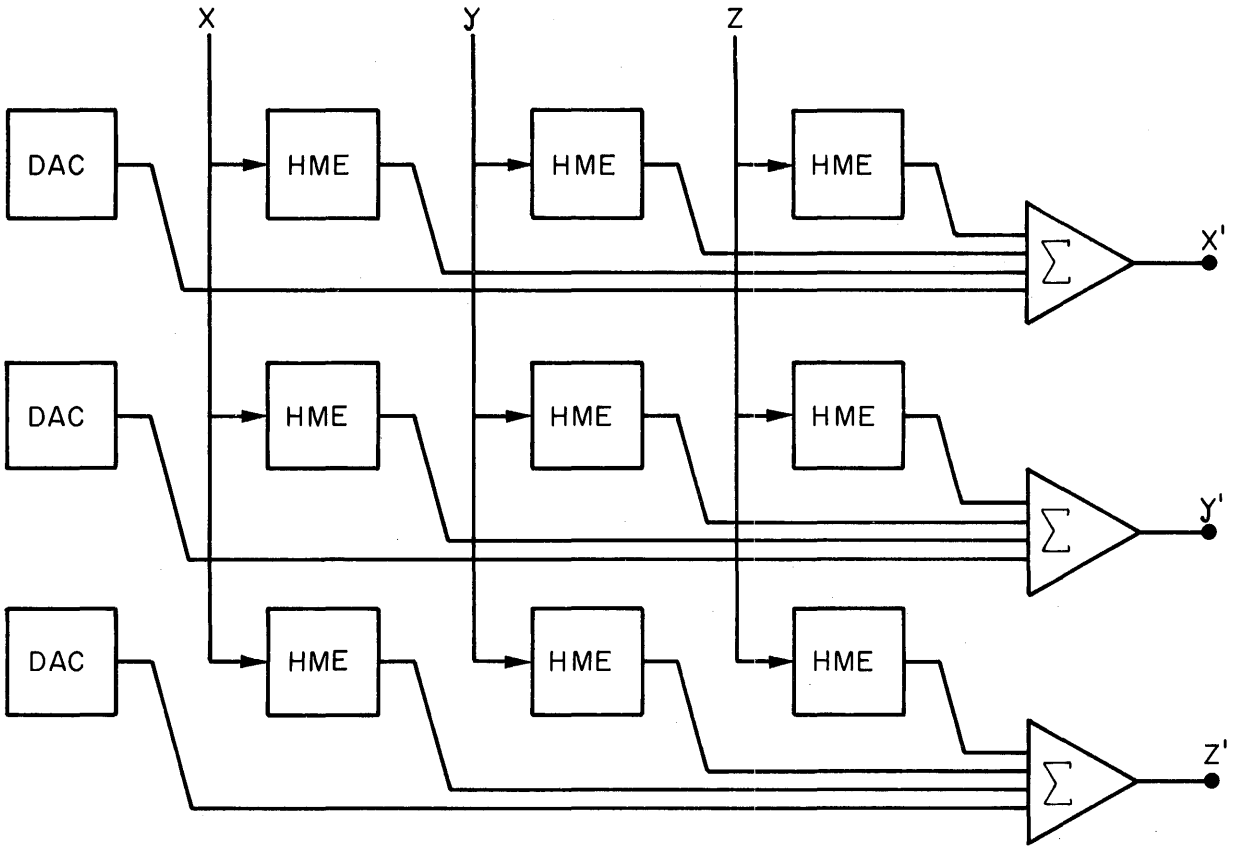


Figure 5. An array for coordinate transformation.

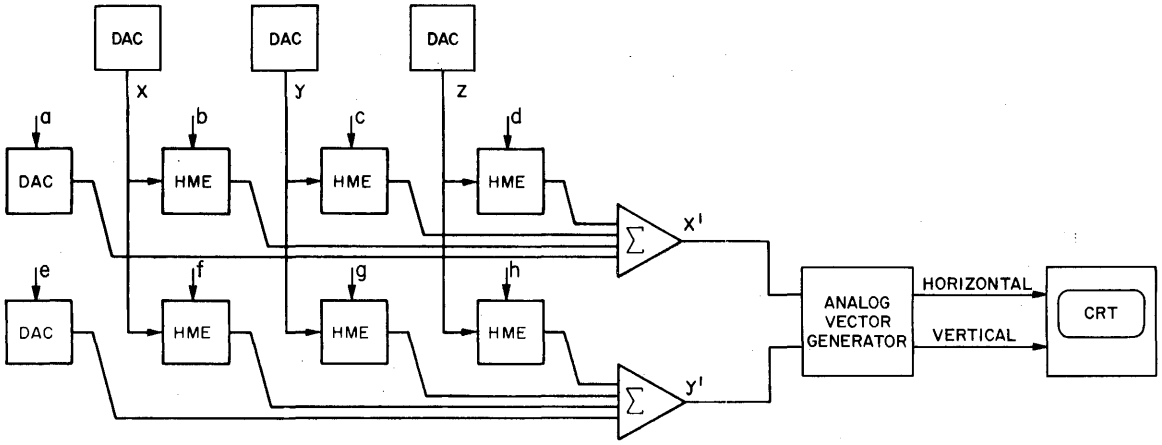


Figure 6. Hybrid technique for generation of visual displays with isometric translation and rotation.

necessary multiplications and additions are accomplished as the coordinates for each successive point are fetched from memory and loaded into the input DAC's.

Geometrical relations for true perspective representation of a three-dimensional object are shown in Fig. 7. Coordinates of the object are stored in memory in terms of the coordinate system xyz. It is desired to display the object on a CRT by plotting the projections of points on the viewing plane $x'y'$,

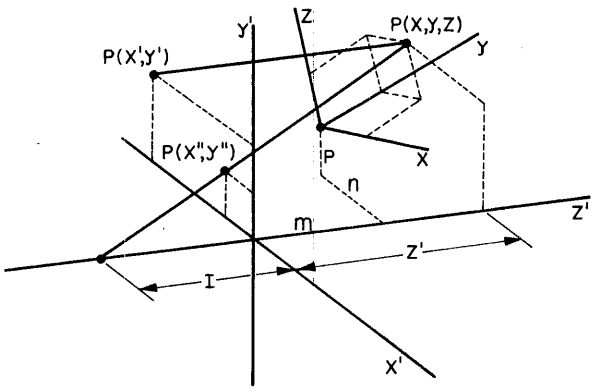


Figure 7. Geometrical relations for true perspective display.

a "window" through which an observer located on the z' axis at I views the object. The coordinate system of the object is arbitrarily translated and rotated with respect to the coordinate system of the viewing plane. A hybrid technique suitable for determining x'' and y'' is illustrated in Fig. 8. Coordinates of the point xyz in the object are transformed into the quantities Ix', Iy' and $I + z'$. Analog division is then used to produce x'' and y'', coordinates of the intersection of the line of sight (between observer and object) and the plane of the window. Again, an analog vector generator connects successive points to generate a line drawing.

For "continuous" (to the eye) motion of the displayed object, coefficients in the matrix must be recomputed for every frame. With inputs in the form of translations, (m, n and p) and angles (or direction cosines) between the two coordinate systems, the coefficients can be computed in less than 500 microseconds. With a 30-microsecond vector generator, therefore, objects represented by more than 1000 line segments can be displayed at 30 frames per second. For each fetch of the three operand values x, y and z representing a point, the hybrid array performs nine multiplications, 12 additions, and two divisions—slightly more than seven arithmetic operations per operand fetch.

FUNCTION GENERATION

Hybrid techniques have proven themselves especially useful for the generation of arbitrary functions of one or more variables. Various methods have been worked out, including techniques in which quadratic or cubic interpolation is achieved at high speed by using hybrid arithmetic elements in parallel to accomplish evaluation of the polynomial algebraic expression required for the interpolation. For most applications, straight-line approximations to arbitrary functions are satisfactory, however. When a function is approxi-

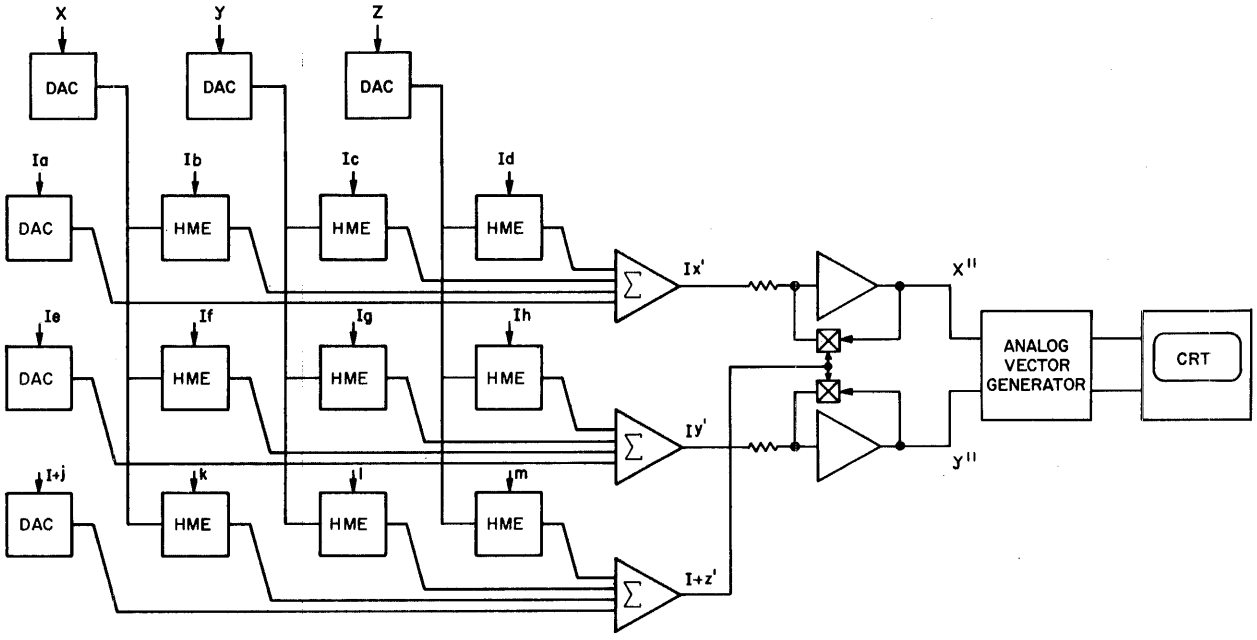


Figure 8. A hybrid technique for generation of visual displays in true perspective with arbitrary translation and rotation.

mated by a series of straight lines, the input/output relation is simply

$$y = mx + b$$

with different values of m and b for each line segment.

In one method for generation of an arbitrary function of one variable, different values of m and b corresponding to different line segments are stored in a table in memory. The input signal x is monitored periodically to determine which region it is within. When x is found to have moved into a region corresponding to a new line segment, the appropriate new values of m and b are transferred from memory into an HME and DAC, whose outputs are summed to produce y.

This basic technique for establishing an arbitrary nonlinear relationship between an analog input and an analog output can be extended in various ways to accommodate functions of two or more variables. A number of different methods have been explored by Chapelle,[3] which are based upon hybrid techniques for linear interpolation between breakpoints, resulting in a series of discrete outputs between breakpoints. One of several methods developed for use with the AMBILOG 200 is similar to Chapelle's techniques; however, it provides a continuous output between breakpoints, eliminating errors due to the discrete steps and the delays encountered in signal conversions from analog to digital and back again. To accomplish this, the independent variable is used directly in the analog domain in order to obtain the appropriate interpolating factors.

Consider a function of two variables $F(x,y)$ whose values at equally spaced breakpoints in x and y are known. Using double linear interpolation, the expression for $F(x,y)$ is:

$$F(x,y) = F_{n,n}\bar{\delta}_x\bar{\delta}_y + F_{n,n+1}\bar{\delta}_x\delta_y + F_{n+1,n}\delta_x\bar{\delta}_y + F_{n+1,n+1}\delta_x\delta_y$$

where $x_n < x < x_{n+1}$

$y_n < y < y_{n+1}$

and $\delta_x = \dfrac{x - x_n}{x_{n+1} - x_n}$

$\delta_y = \dfrac{y - y_n}{y_{n+1} - y_n}$

$\bar{\delta}_x = 1 - \delta_x$

$\bar{\delta}_y = 1 - \delta_y$

The values of F are the values of the dependent function at the corresponding breakpoints. The δ terms may be considered as interpolating factors. Figure 9 shows the case where:

$$x_0 \le x \le x_1 \qquad \text{and} \qquad y_0 < y < y_1$$

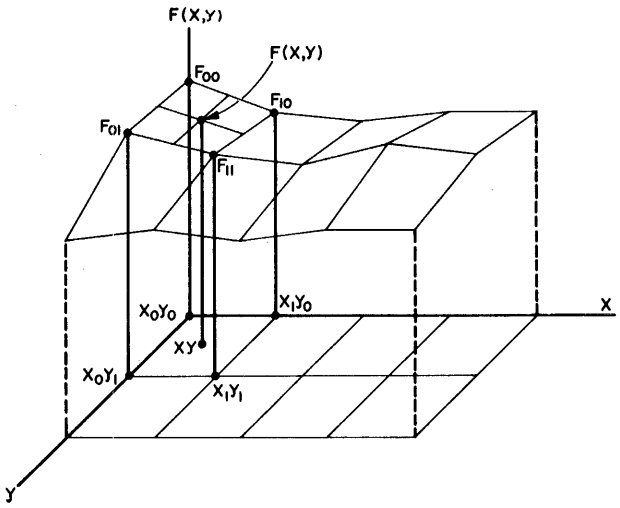


Figure 9. A function of two variables $F(x,y)$ with equally spaced breakpoints.

For this case, the bounding values of the independent variables, i.e., x_0, x_1, y_0, and y_1 are determined programmatically by monitoring x and y. By means of DAC's, the values of x_0 and y_0 are transmitted to the analog computing elements where the δ products are developed as shown in Fig. 10. These δ products are then returned as inputs to the array of HME's. The HME's are loaded with the digital values of the dependent variable at the breakpoints in order that each may compute one term of the equation for $F(x,y)$. These four HME outputs are summed to obtain the desired result, $F(x,y)$. As shown in Fig. 10, the eight δ products may be applied to other sets of HME's to obtain other arbitrary functions of the input variables x and y.

Note that as long as

$$x_1 < x < x_0 \qquad \text{and} \qquad y_1 < y < y_0$$

continuous interpolation takes place. This significantly relieves the stored program processor of the major computing chore normally associated with hybrid function generation. The primary tasks of the AMBILOG 200 System Control Unit in performing function generation using this

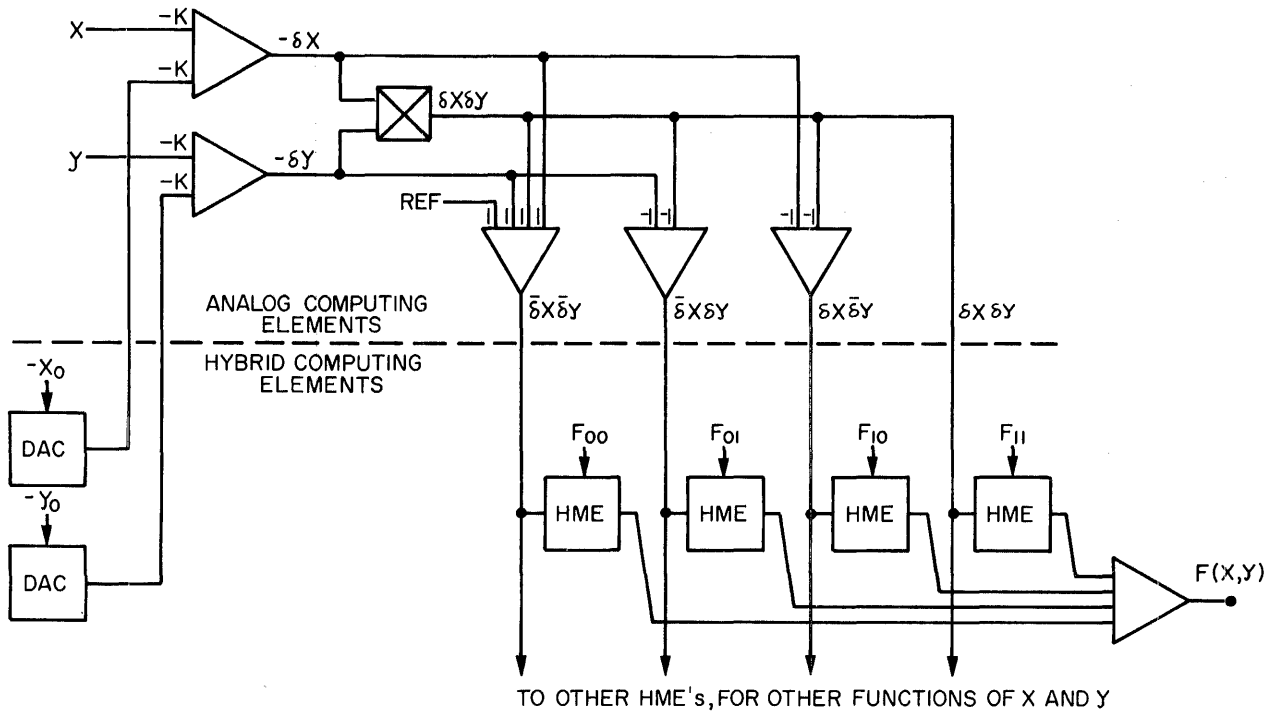


Figure 10. Generation of a function of two variables.

technique are:

1. Bracketing or locating the independent variables with breakpoints;
2. Fetching the appropriate values of the dependent variables at the breakpoints and loading them into the HME's; and finally,
3. Monitoring the independent variables to detect the crossing of breakpoint boundaries.

This technique, illustrated for a function of two variables, applies equally well in the general case of a function of n variables. In this case, there are n independent variables and 2^n terms in the equivalent algorithm corresponding to the 2^n values of the dependent variable which must be interpolated to obtain $F(x_1, x_2, x_3 \cdots x_n)$.

It is worthwhile noting that the technique utilizes the value of the dependent variables at breakpoints directly. This is desirable because 1) the breakpoint data may be used without modification, and 2) the dependent variable tables occupy a minimum of core storage. Furthermore, the continuous interpolation between breakpoints minimizes errors and reduces the computing load on the AMBILOG 200.

Typically, utilizing the above technique with the AMBILOG 200 in performing an aerodynamic simulation involving force and moment equations, 10 functions of two variables may be computed in less than one millisecond. This capability represents a significant improvement over conventional techniques and in practice allows for faster-than-real-time simulation of many problems. This faster-than-real-time capability in turn enhances the use of statistical parameter search techniques in simulation studies.

CONVOLUTION

Convolution and correlation operations provide another fruitful area for application of hybrid techniques. A hybrid method for performing convolution integration is shown in Fig. 11. An eight-term weighting function is loaded into a set of HME registers. The HME analog inputs are derived from DAC's whose flip-flops are interconnected as shift registers. Successive samples of a digitized signal are fed into the first DAC. With each new sample, all DAC values shift right one position, so that successive samples are stepped sequentially through the shift register array. Eight cross products are multiplied and summed simultaneously each time a new sample of the input signal is taken.

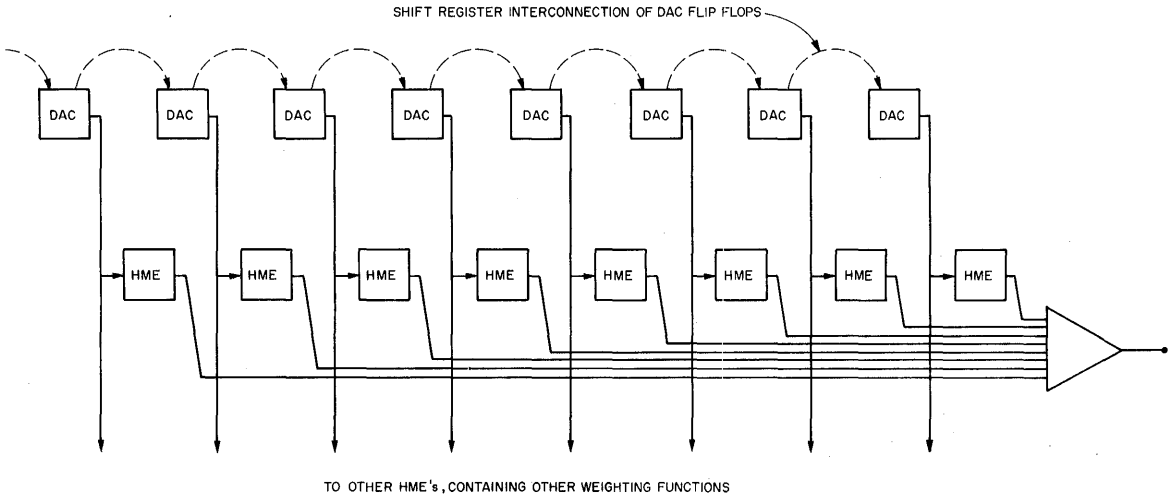


Figure 11. A parallel array for convolution operations.

A new sample can be entered about every 10 microseconds, so the array shown has a multiply/add rate of about 800 kc. That rate can be increased by expanding the array, either by adding more DAC/HME pairs, or by adding more rows of HME's. Overall multiply/add rates of 10 megacycles or more are practical.

INTERCONNECTIONS

In each of the cases above where multiple operators are useful in parallel, a specific interconnection of analog signal paths between elements is shown. The techniques illustrated can indeed be implemented by permanently interconnecting a set of operators, which then become committed to performing only those operations implied by the interconnections. However, the approaches illustrated here do not necessarily entail permanent commitment of the operators to specific tasks. In fact, the opposite is usually the case, and in actual implementation considerable flexibility of interconnection is achieved, sometimes by means of patch panels and more generally under fully programmatic control.

For arrays numbering 16 or 24 operators, the highly desirable flexibility of fully programmatic control of analog interconnections among operators in the array is economically feasible. Interconnections are established by signal steering elements which consists of digitally controlled, solid state, precision analog switches. To allow for all possible interconnections of n operators, $n(n-1)$ switches would be required. For arrays of about 24 operators of two or three different types, very little loss of generality results if the number of switches used is reduced by $\frac{1}{2}$ or even $\frac{3}{4}$. Thus, use of about 200 digitally operated analog signal steering switches permits very general stored program control of an array of 24 operators.

Actual applications of some of the techniques described above have involved use of parallel arrays of more than 100 operators. In those cases, some of the operators in the array are interconnected by high-speed programmable signal steering switches, supplemented by conventional analog computer-type patch panels which provide means of interconnection combining economic feasibility with a fair degree of flexibility.

CONCLUSIONS

A number of signal processing techniques have been developed, based upon the use of multiple analog and hybrid elements operating in parallel, all under close stored program control. They have proven both feasible and useful for a variety of applications.

In particular, the same basic array of DAC's, hybrid multipliers, and summing amplifiers, all organized to evaluate sums of cross products, has been shown to be useful for generalized linear transformation, spatial coordinate transformation, generation of visual displays of three-dimensional objects in true perspective, generation of functions of two or more variables, and convolution and cor-

relation operations. Depending upon the nature of the problem and the extent of the array, multiply/add operations at overall rates up to about 10 million per second are possible with present-day components.

Comprehensive, high-speed, programmatic control of analog interconnections among operators in an array is feasible for arrays of up to about 24 operators. Beyond that number, analog computer patch panels are generally used. Systems with over 100 operators of the DAC and hybrid multiplier types are being built, and in fact are not uncommon.

Various applications of the AMBILOG 200 Stored Program Signal Processor have provided considerable experience with hybrid techniques of the general type discussed in this paper. That applications experience now includes real-time acquisition and analysis of seismic waveforms, real-time analysis of physiological signals from postoperative heart surgery patients, speech analysis, sonar analysis, wind tunnel data reduction, flight test telemetry data reduction, aerospace vehicular simulation, and simulation in real time of a secure speech communication system.

ACKNOWLEDGMENTS

We wish to acknowledge the contributions of Mr. Sol Max and Mr. Len Sacon of Adage to the perspective display technique, and of Mr. Art Rubin of the Martin Company, who is largely responsible for the function generation technique described herein.

REFERENCES

1. J. D. Grandine and T. G. Hagan, "A Parallel/Sequential, Stored Program Hybrid Signal Processor," *Simulation,* Jan. 1965.

2. J. F. Dammann et al, "Data Acquisition and Interpretation System for Postoperative Patients," *Proceedings,* San Diego Symposium for Biomedical Engineering, 1964.

3. W. E. Chapelle, "Hybrid Techniques for Function Generation," *Proceedings,* Spring Joint Computer Conference, 1963.

HYBRID SIMULATION OF A REACTING DISTILLATION COLUMN

R. Ruszkay
E. I. duPont de Nemours & Company, Inc., Wilmington, Delaware
and
E. E. L. Mitchell
Electronic Associates, Inc., Princeton, New Jersey

INTRODUCTION

The recent availability of hybrid computers to the simulation engineer has made possible the solution of many complex reactor models that hitherto had to be either crudely approximated, or else required extremely long solution times on all-digital machines. Using the high speed of the parallel analog equipment and the logic and storage capabilities of a digital computer has meant that single blocks of analog equipment can be time-shared where physical similarity exists between process modules resulting in large savings in both analog equipment and running time.

We have written this paper to describe the simulation of a distillation-reactor column which, because of plate-to-plate similarity, has natural advantages when multiplexing a single analog tray model throughout the column.

Experience has shown that such a simulation can be run at approximately 120 times plant time so that one minute of computer time becomes equivalent to two plant hours. With variable amplitude scaling built into the simulation, large dynamic ranges in the magnitudes of the concentrations can be handled so that a complete system start-up can be studied. Because of the exothermic reaction, runaway effects were possible, and a study of the dynamics of the column was important both from a control and efficiency point of view.

The basic simulation embodies continuous analog representation of the reboiler and condenser which interact with a digital simulation of the column. In order to solve the plate dynamics at sufficiently high speed, an analog subroutine was used to model one plate and then switched at high speed up and down the 26 plates in the column. Since the digital program entailed mainly storage and logic operation, a relatively unsophisticated computer would suffice together with adequate analog to digital and digital to analog data channels.

GENERAL DESCRIPTION OF PROCESS

A schematic of a typical system which might be simulated is shown in Fig. 1. The main piece of equipment in the process would be a conventional distillation column having 26 trays. Reaction as well as separation is carried out in this column where the bottom 20 trays constitute the main reaction zone. The feeds to the column consist of two liquid reactants, A and B. Reactant B is introduced on the 20th tray while reactant A is introduced on the 21st tray.

A and B react together to form products C and D according to the reversible equation

$$A + B \underset{k_R}{\overset{k_F}{\rightleftharpoons}} C + D$$

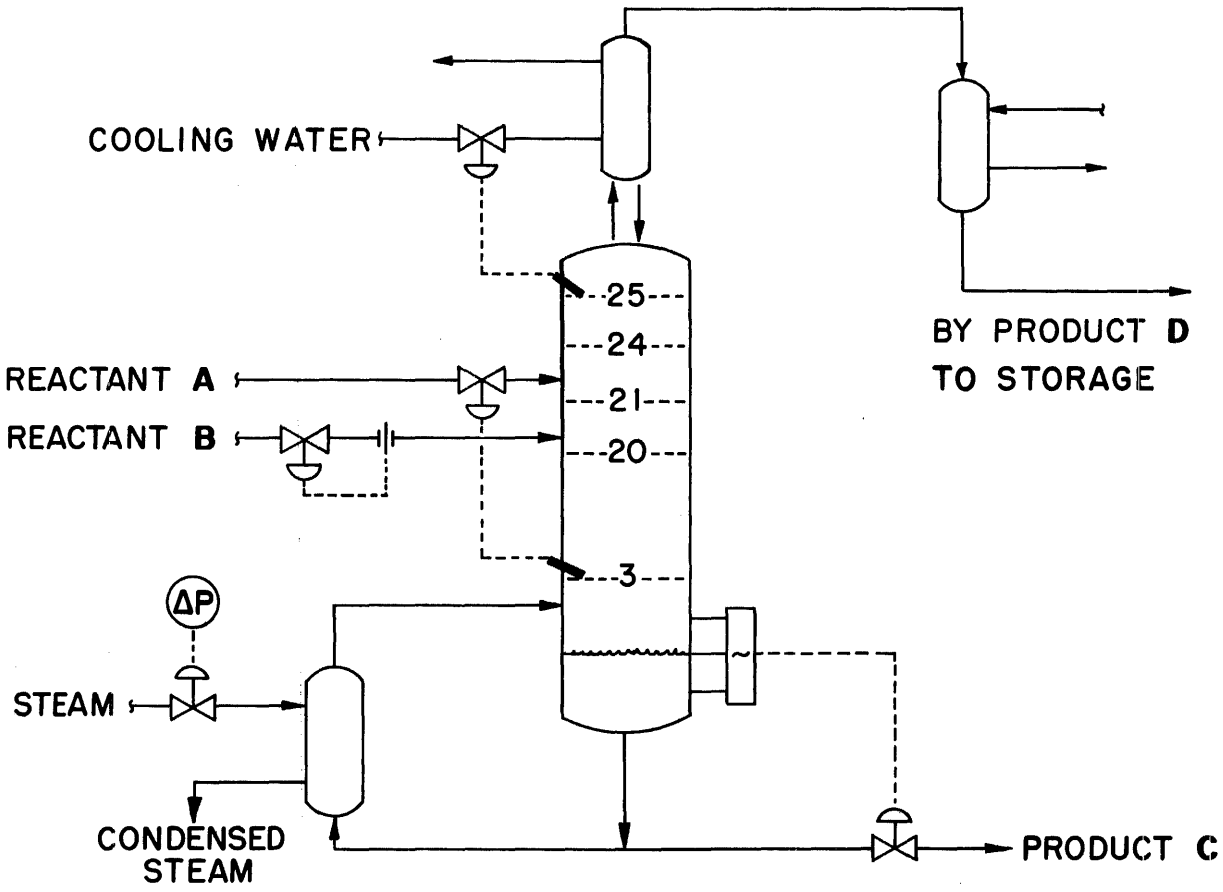


Figure 1. Schematic of the system.

The reaction is moderately exothermic and the rate constants, k_F and k_R obey the Arrhenius relationship

$$k = Ae^{-B/T}$$

The vapor flow leaving the top tray (#26) is sent to a partial condenser which further purifies the by-product D, and provides liquid reflux to the top tray. The vapors leaving the partial condenser are sent to a total condenser and then to liquid storage tanks. The heat required to provide the column boilup is supplied through a shell and tube heat exchanger (reboiler) by condensing steam on the shell side. Liquid product is removed from the bottom of the column. Reactant A is added in excess and removed with product C together with trace quantities of reactant B. By-product D is so volatile relative to the other components, that its concentration in the bottoms is very small. Conversely, the concentration of B and C above the feed plates is also very small. The six trays above the main reaction zone constitute a clean up zone to separate reactant A and by-product D.

A tentative control scheme is also shown in Fig. 1. The temperature on plate #25 controls flow of cooling water to the partial condenser. Temperature on plate #3 controls the feed flow rate of reactant A. The feed flow rate of reactant B is flow controlled. The column pressure drop controls steam flow to the reboiler. The level in the bottom of the column controls the bottom's flow rate.

MATHEMATICAL MODEL OF THE PROCESS

In order to simulate the process, a realistic mathematical model must be developed to describe the pertinent physical phenomena which are taking place.

The principal equations used to describe the tray are statements of the material and enthalpy balances, vapor-liquid equilibrium and reaction kinetics.

Each component is described by a difference equation of the form:

$$\frac{d}{dt}[M_n X_n^i] = L_{n+1}X_{n+1}^i + V_{n-1}Y_{n-1}^i - L_n X_n^i - V_n Y_n^i (\pm) R_n + F_n^i \quad (1)$$

where M_n = liquid hold-up on the nth tray (moles),
Y_n^i = mole fraction of ith component in vapor leaving nth tray,
L_n = liquid flow rate leaving nth tray (moles /unit time),
V_n = vapor flow rate leaving nth tray (moles /unit time),
R_n = rate of production (or consumption) of component due to reaction, and
F_n^i = feed rate of the ith component (if any) on to nth tray.

These material balance equations are solved for the individual component hold-up $M_n X_n^i$.

The total material balance is written as:

$$M_n = \sum_i M_n X_n^i \quad (2)$$

Enthalpy balance provides the total enthalpy hold-up on a tray by:

$$\frac{d}{dt}[M_n \bar{C}_n T_n] = L_{n+1}\bar{C}_{n+1}T_{n+1} + V_{n-1}\bar{H}_{n-1} - L_n\bar{C}_n T_n - V_n\bar{H}_n + \Delta HR_n \quad (3)$$

where T_n = temperature on nth tray (°C),
$\bar{C}_n$ = average specific heat of liquid on nth tray (PCU/mole °C),
$\bar{H}_n$ = average enthalpy carried by vapor from nth tray (PCU/mole), and
ΔH = heat of reaction (PCU/mole).

The vapor-liquid equilibrium can often be expressed using Raoult's Law and a plate efficiency factor.

Using a contact value of 0.70 for the plate efficiency, the vapor mole fraction Y_n^i leaving the tray can be related to the vapor mole fraction entering, Y_{n-1}^i, the equilibrium mole fraction Y_n^{i*} by the relation

$$Y_n^i = 0.7\,Y_n^{i*} + 0.3\,Y_{n-1}^i \quad (4)$$

Using Raoult's Law the equilibrium mole fraction is obtained by:

$$Y_n^{i*} = \frac{P_i^0(T_n)\cdot X_n^i}{\pi_n} \quad (5)$$

where $P_i^0(T_n)$ = vapor pressure of ith component on nth tray which is a function of temperature, and π_n = total pressure on nth tray.

The reaction kinetics are described by:

$$R_n = M_n k(T_n)\left[X_n^A X_n^B - \frac{X_n^C X_n^D}{K_{EQ}(T_n)}\right] \quad (6)$$

where $k(T_n)$ = reaction rate constant, and
$K_{EQ}(T_n)$ = equilibrium constant

The pressure above the nth tray, π_n, is equal to the pressure above the $(n+1)$th tray plus the pressure drop across the $(n+1)$th plate, i.e.:

$$\pi_n = \pi_{n+1} + \Delta P_{n+1} \quad (7)$$

The pressure drop across the plate is made up of the hydrostatic head plus the dynamic drop due to the vapor flow. We can assume the weir height is large relative to changes in crest over the weir so a constant average value, ΔP_0, can be used for the liquid seal pressure drop. The dynamic vapor drop is a function of the vapor flow rate, which can be obtained from the tray manufacturer's data, so

$$\Delta P_{n+1} = \Delta P_0 + f(V_n) \quad (8)$$

The main computational difficulty is in determining the molar vapor flow rate leaving the tray. If the equations are solved digitally, an iteration must be performed to adjust the vapor flow until the constraint equation that the sum of the Y^i's is unity is satisfied, i.e.:

$$\sum_i Y_n^i = 1 \quad (9)$$

In the analog simulation, the constraint is satisfied by feeding the error in this equation into a high gain amplifier and calling the output V_n. The circuit rebalances automatically in the operate mode through the energy balance and equilibrium relationships to give the correct value for this vapor flow rate.

HYBRID SIMULATION

The mathematical model just described for the representation of a single plate is fairly simple, but well represents the physical behavior of trays. Simple as it is however, in mechanizing the equations computationally, a significant quantity of analog equipment would be utilized. The same applies to an all digital simulation but here the important parameter denoting feasibility is time. Because of the implicit algebraic loops involved, a number of iterations are required within a time step so that the

analog representation, with effectively instantaneous solution of implicit equations, shows to advantage.

For the analog mechanization, however, a representation of a complete distillation-reactor column with a full simulation of each tray becomes impractical, since the equipment complement required could be multiplied by factors ranging from 20–100. There are a number of ways to avoid this enormous equipment duplication,[1,2] but the one we intend to discuss is that of time-sharing. Essentially, we want to use the high-speed capability of the analog computer which is at its best when solving implicit algebraic loops or continuously integrating time-dependent, nonlinear differential equations. The sequential nature of the digital computer suggests using an analog single plate representation as a subroutine, performing the appropriate calculations on each plate in turn.

The solution of a set of differential equations depends on the initial conditions established on the state variables and the time history of the forcing functions acting on the system.

Consider the representation of a single plate in Fig. 2 where the simulation solves a set of five first-order, differential equations—Eqs. (1) and (3) above—under the influence of adjacent trays, the outputs of which may be considered as forcing functions. The lower plate acts through the vapor flow and there are five functions corresponding to component and enthalpy fluxes into the plate. Similarly, the plate above only interacts via a liquid flow down the downcomer, which again consists of five component and enthalpy fluxes. Because of the existence of feed plates, component and enthalpy feeds must also be considered in enumerating forcing functions acting on the tray.

Now we make the fundamental assumption that is necessary to allow any sequential solution to be meaningful. This assumption is that if the differential equations representing the plate are allowed to integrate for a sufficiently short time, then the magnitudes of the forcing functions (fluxes) acting on a tray will not have changed appreciably and so in a simulation can be held constant. The main problem in any simulation becomes the interpretation of how small a time step is necessary in order that the complete solution will adequately represent the dynamics of the actual operating column.

Basing results on a stirred tank time constant τ $(=M/L)$ one can say that a time step of 0.1τ is probably as low as it is necessary to go, and that a value of 0.5τ would normally be acceptable for an overall check of column dynamics.

Varying the time step does not influence stability, the only effect being to increase solution time when very small time steps are chosen since establishing initial conditions and determination of end point values takes a fixed time determined by the equipment used.

The operation of the simulation with the assumption of constant forcing functions is to establish as analog voltages initial conditions on the five state variable integrators. Component and enthalpy fluxes—both vapor and liquid—forcing the plate are established on 10 track/store amplifiers and the program allowed to integrate for about 0.2τ (say). At the end of this time, a hold mode is established, and the vapor and liquid fluxes leaving the tray determined for use as forcing functions for adjacent trays. The new state variable values are also measured and these replace the old values in the state variable table which represents the condition of the column.

Each tray can be operated on sequentially from bottom to top, thus assembling within the memory of the digital computer a set of tables describing both the state variable magnitudes (hold-up) and the fluxes between trays, representing coupling or interaction. If the operation is repeated again, starting from the bottom, each tray will integrate to steady state. The whole system will then operate as a dynamic representation with a time scale of nT, where T is the equivalent integration period of a single tray and n is the number of trays operated on.

There are two problems with this approach, which are, first, the delay involved in propagation of disturbances down the column and, second, the dynamic range of the variables over the height of the column.

When the trays are treated sequentially from bottom to top (or vice versa) with the sequencing direction maintained, disturbances introduced into a tray (i.e., at the top) can only propagate downwards at a rate of a single tray per sweep. That is, an LX calculated on a tray (resulting from an LX from the tray above) would not be used, for the tray below, until the next upward sweep. Changes in liquid or vapor flow rates occur rapidly, compared with temperature changes say, because the weir time constant can be effectively neglected. Thus an increase in reflux should appear fairly rapidly as a higher molar flow into the reboiler, without waiting for n sweeps to allow the change to propagate down n trays.

A compromise then, to incorporate faster than usual transients, is necessary and is accomplished

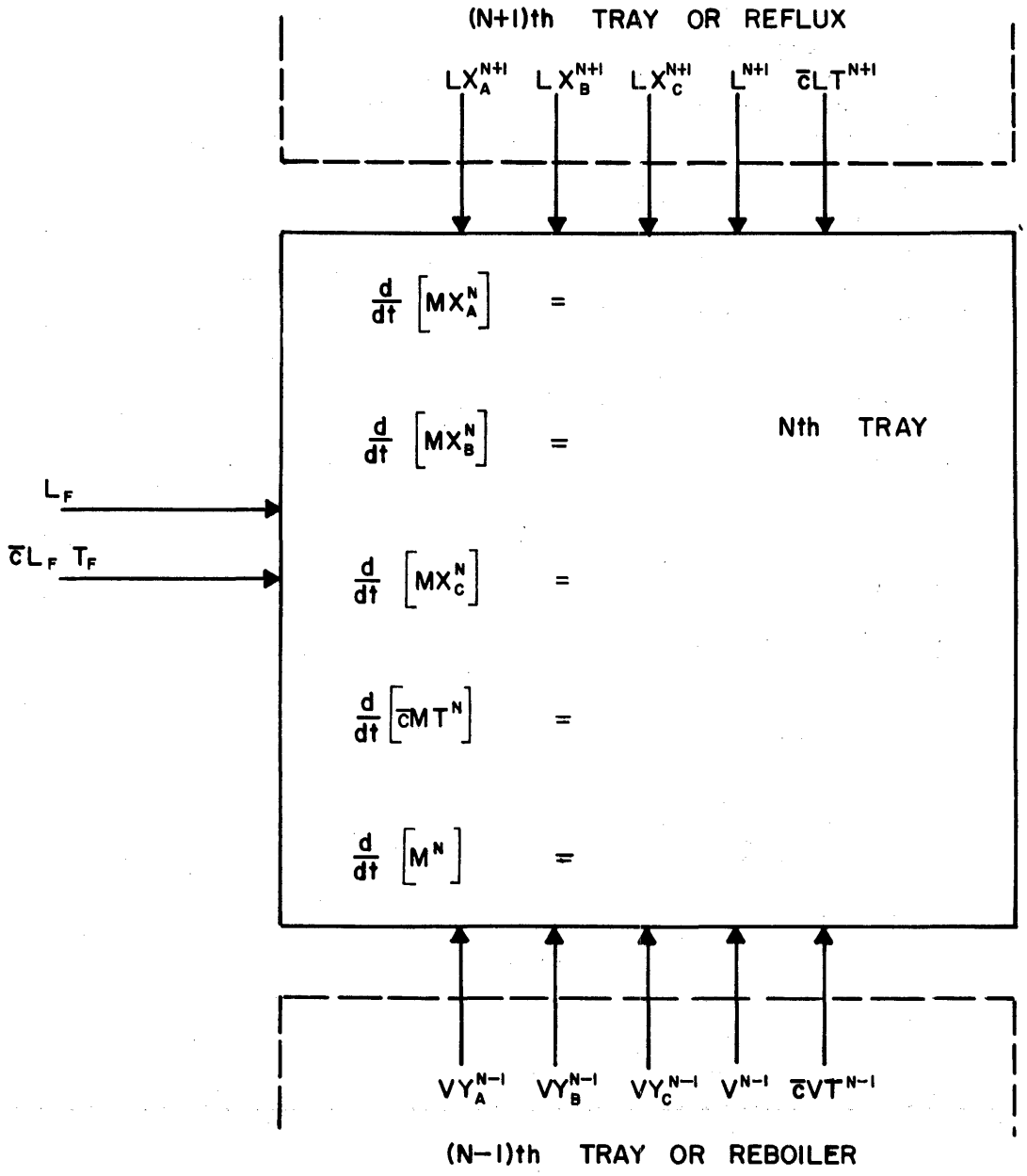


Figure 2. Interaction of plate with surroundings.

by sequencing the operating tray first up, and then down the column. On the way up VY's computed as leaving a plate are used immediately to force the tray directly above and on the way down LX's computed as leaving a tray are used to force the plate directly below.

A rather subtle effect appears here in that any LX calculated on the way up is never used, being superseded by a value calculated on the way down. Similarly, a VY calculated on the way down is superseded by one calculated on the way up. Consequently, the forcing function inputs to the nth tray are maintained constant for two column sweeps, the LX's for down and up sweeps, the VY's for up and down sweeps.

Rescaling is necessary in most columns, since dynamic ranges of 10^4 or 10^5 are quite common and the removal of trace materials usually determines the quality of the product.

Rescaling is only necessary in part of the simulation. If a component goes to zero, the contribution to enthalpy balance or total vapor flow is negligible

and does not have to be included with high accuracy. The accuracy requirement hinges on the calculation of mole fractions in vapor and liquid and corresponding changes in component flux into and out of a tray. It will be shown in the next section that this rescaling operation can be conveniently performed by a digital multiplication or division which preserves the full precision available.

In order to maintain constant scaling in part of the problem appropriate attenuation factors are switched in and out (with D/A switches) at different regions in the column.

Consider, for instance, the liquid flux of D, the volatile component (Fig. 3) which could be scaled up from $[LY^D]$ leaving the 20th tray to $[10LY^D]$ forcing the 19th tray by

$$10\,[LY^D_{20}] \rightarrow [10\,LY^D]_{\substack{\text{FORCING}\\19}}$$

Similarly, $[10\ VY^D]$ leaving the top of the 19th tray must be scaled down so that

$$\frac{1}{10}[10\,LY^D]_{19} \rightarrow [LY^D]_{\substack{\text{FORCING}\\20}}$$

ANALOG PLATE SIMULATION

Figure 4 shows the loops controlling the molar mass balance for component A on a single plate and identical circuits are used to determine molar hold-

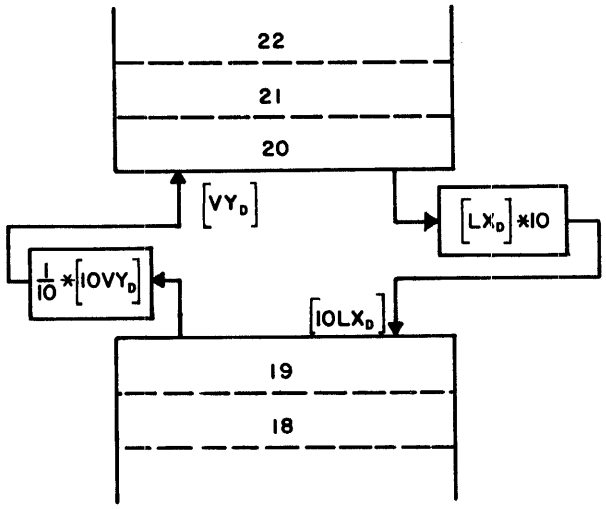


Figure 3. Rescaling of volatile component.

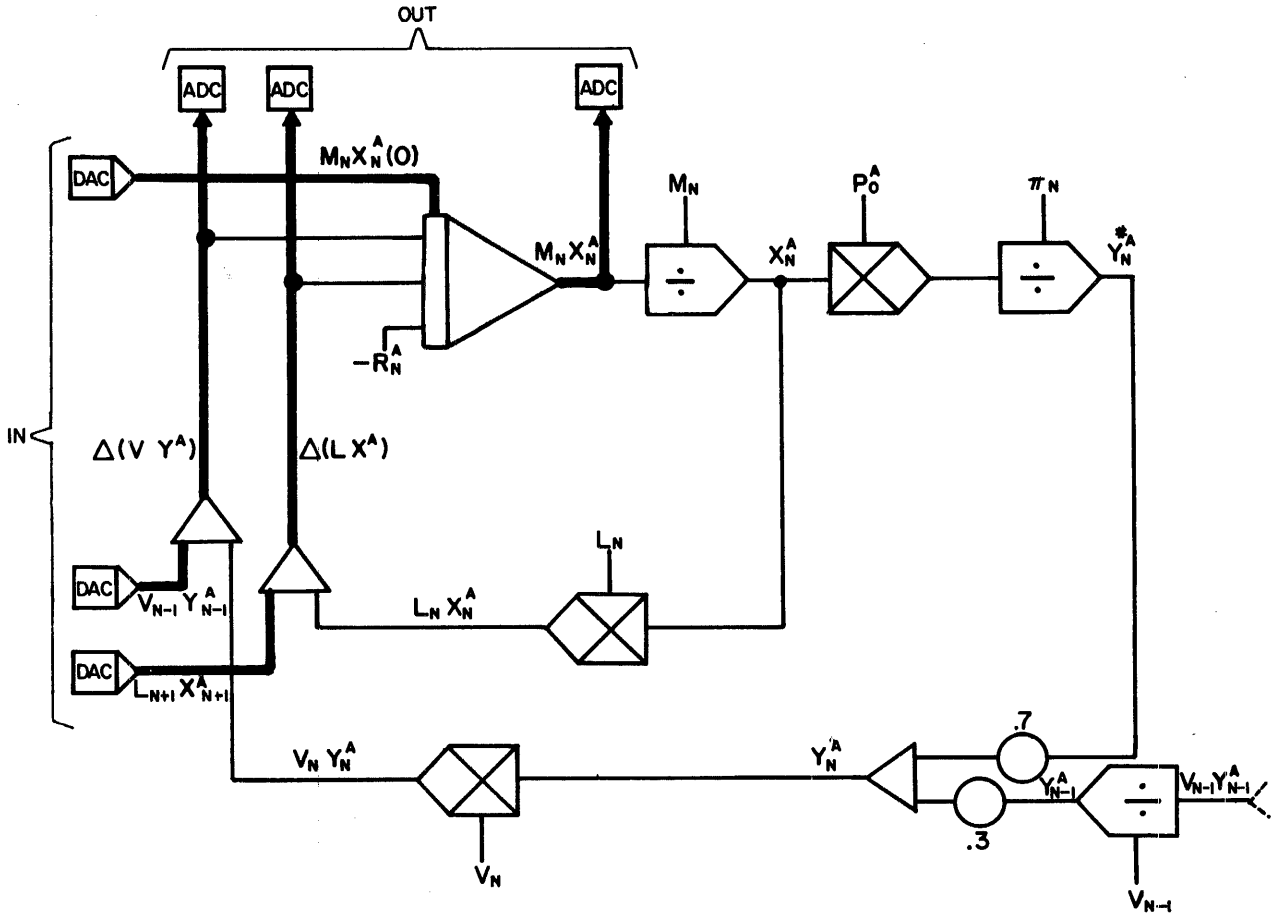


Figure 4. Main loops controlling mass balance.

ups for the other three components. Inputs from the digital memory through D/A converters are required for the original molar holdup prior to the current integration period, $M_n X_n^A(0)$, and also the molar fluxes carried by the vapor from below, $V_{n-1} Y_{n-1}^A$, and by the liquid from above, $L_{n+1} X_{n+1}^A$. At the end of the integration period the molar fluxes out of the tray and the current molar holdup can be sampled by the A/D converter and stored in the digital memory.

The equations solved by this section are the mass balance

$$\frac{d}{dt}[M_n X_n^A] = V_{n-1} Y_{n-1}^A - V_n Y_n^A + L_{n+1} X_{n+1}^A - L_n X_n^A + R_n^A$$

and Raoult's Law for the equilibrium vapor concentration

$$Y_n^{A*} = \frac{P_0^A}{\pi_n} X_n^A$$

where P_0^A is the equilibrium vapor pressure of component A, considered to be a function of temperature.

The enthalpy balance is obtained (Fig. 5) in a similar way to the mass balance circuit. In addition however, the molar vapor flow V_n provides a tight loop because of the large enthalpy carried by the latent heat of vaporization. This vapor flow rate is obtained by forcing the molar concentrations in the vapor phase to add up to unity. Note that where components interact, a changing scale factor implies a changing gain, so the amplifier inputs identified by an asterisk (*) become either relay contacts or digital to analog switches, providing gain variation in decade steps.

The reaction rate constants can also be obtained as functions of temperature in this section. The reaction rates themselves are obtained by standard logarithmic multiplication of the respective concentrations and reaction rate constants and are not shown in detail.

DIGITAL LOGIC AND CONTROL PROGRAM

The digital computer is responsible for establishing voltages corresponding to inputs to the tray simulation and subsequently after an integration period reading the results of the computation back into memory.

The basic timing sequence (Fig. 6) consists of a repetitive IC/OPERATE/HOLD cycle which we will assume is a 4-msec IC, 8-msec OPERATE and 4-msec HOLD. The actual times used depend on hardware speeds and time scaling, but we believe these would be typical. During the IC period, voltages corresponding to inputs from above and below the tray are established on track/store amplifiers; state variable magnitudes are established on the tray integrators. During the OPERATE period, the simulation integrates for 8 msec, and this would be time-scaled to correspond to about 0.2 M/L. During the HOLD period, the new voltages determining tray outputs—vapor and liquid fluxes—and state variable magnitudes are read into appropriate slots in the tables established in digital memory. These voltages are determined by stepping round a multiplexer connected to a single analog-digital converter.

In order to follow the logical control for sequencing the tray representation up and down the column, consider Fig. 7.

Within the digital memory, data tables are defined which are TOP+2 entries long, where TOP is the number of plates in the column. The two extra entries are required to hold the boundary conditions. A total of 16 tables are required; 5 for the LX's and LT, 5 for the MX's and MT, 5 for the VY's and $\bar{c}VT$ and 1 for the plate pressure, which is not an excessive memory requirement.

Before operation, these tables must be initialized to correspond to some physically meaningful state of the column. If the operating conditions aren't known, then a cold steady state can be inserted and the computer used to calculate the operating conditions by simulating a plant start-up.

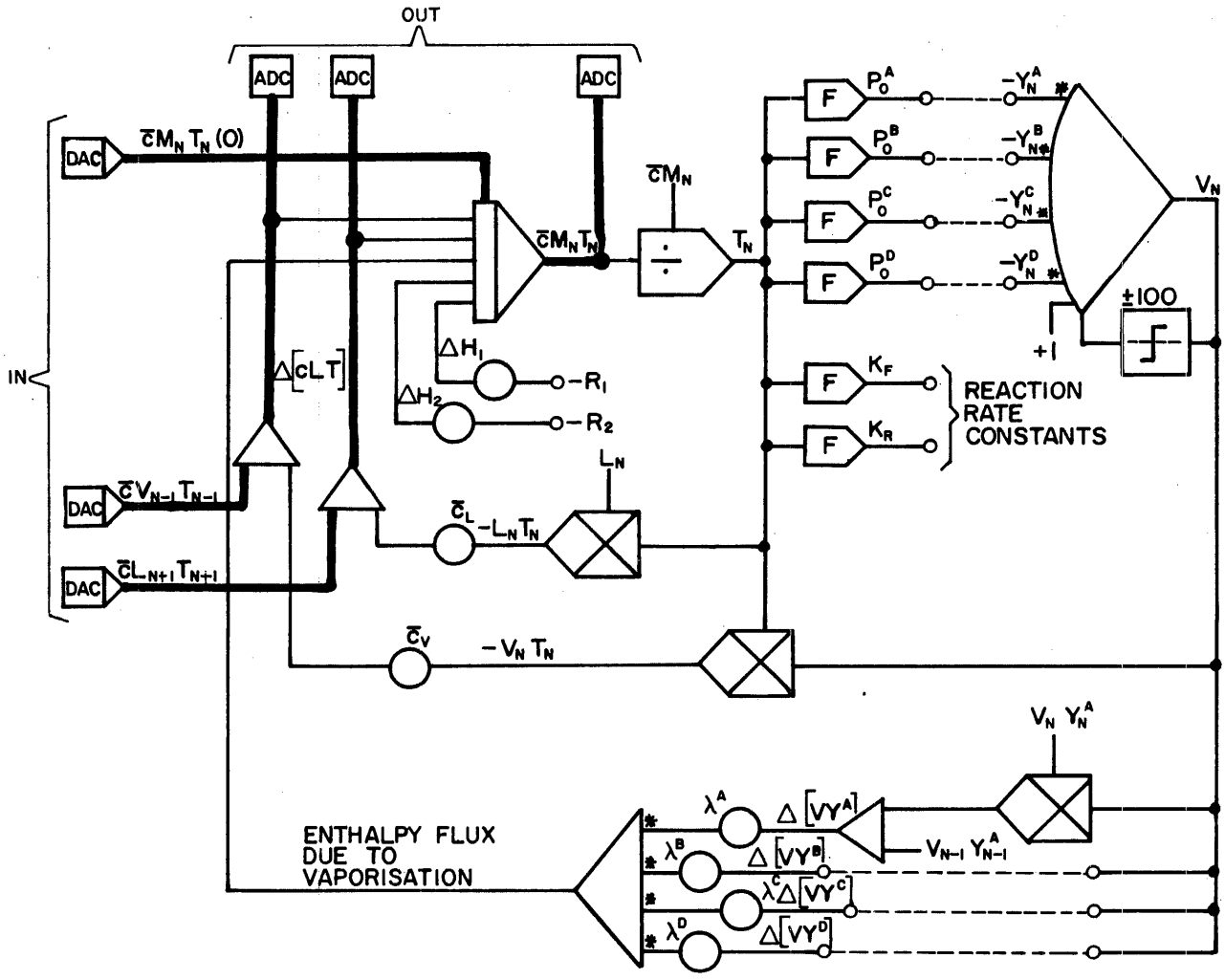
Only two control locations are needed by the digital program, which are:

T—the current tray number ($\leq$ TOP), and

UP—this is one when the simulation is proceeding up the column and zero going down.

The computation loop is entered from the bottom of the column, going up, so these cells are suitably initialized (Fig. 7).

Three tests follow which determine whether boundary condition information is needed; i.e. the continuous reboiler simulation will be computing VY_A, VY_B, VY_C, V and $\bar{c}VT$ which must be read via a multiplexer and A/D converter into the $T = 0$ slots in the appropriate tables. Similarly, LX_A, LX_B, LX_C, L and $\bar{c}LT$ must be read from the continuous condenser-reflux control simulation for entry into the T = TOP+1 slots in the tables. At the feed trays, feed controller output is determined



N.B. ✱ DENOTES GAIN DEPENDENT ON SCALE FACTOR

Figure 5. Main loops controlling enthalpy balance.

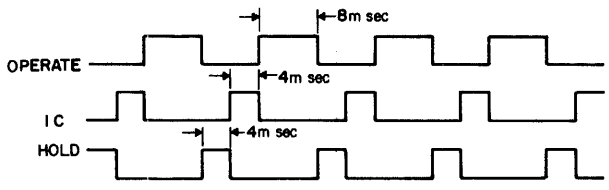


Figure 6. Basic timing sequence.

and the net component and enthalpy fluxes into the tray computed.

Communication of the column with the outside world is via the vapor leaving the column top and liquid leaving the column bottom. In order that the column simulation can force the external continuous simulation for reboiler and condenser, component fluxes in these streams leaving the column must be sampled when the top and bottom trays are simulated. In order to accomplish this, the digital computer controls banks of track/store amplifiers which themselves act as forcing functions for the simulation of the peripheral equipment. Fig. 8 shows the control logic where the output line TOP is set by the digital computer during operation on the top tray and BOTTOM is set during operation on the bottom tray. Since these signals are ANDed with the HOLD signal, the final values are stored.

The digital section, after establishing these boundary conditions, then establishes forcing functions and state variables acting on tray T, via digital-analog converters and track-store amplifiers.

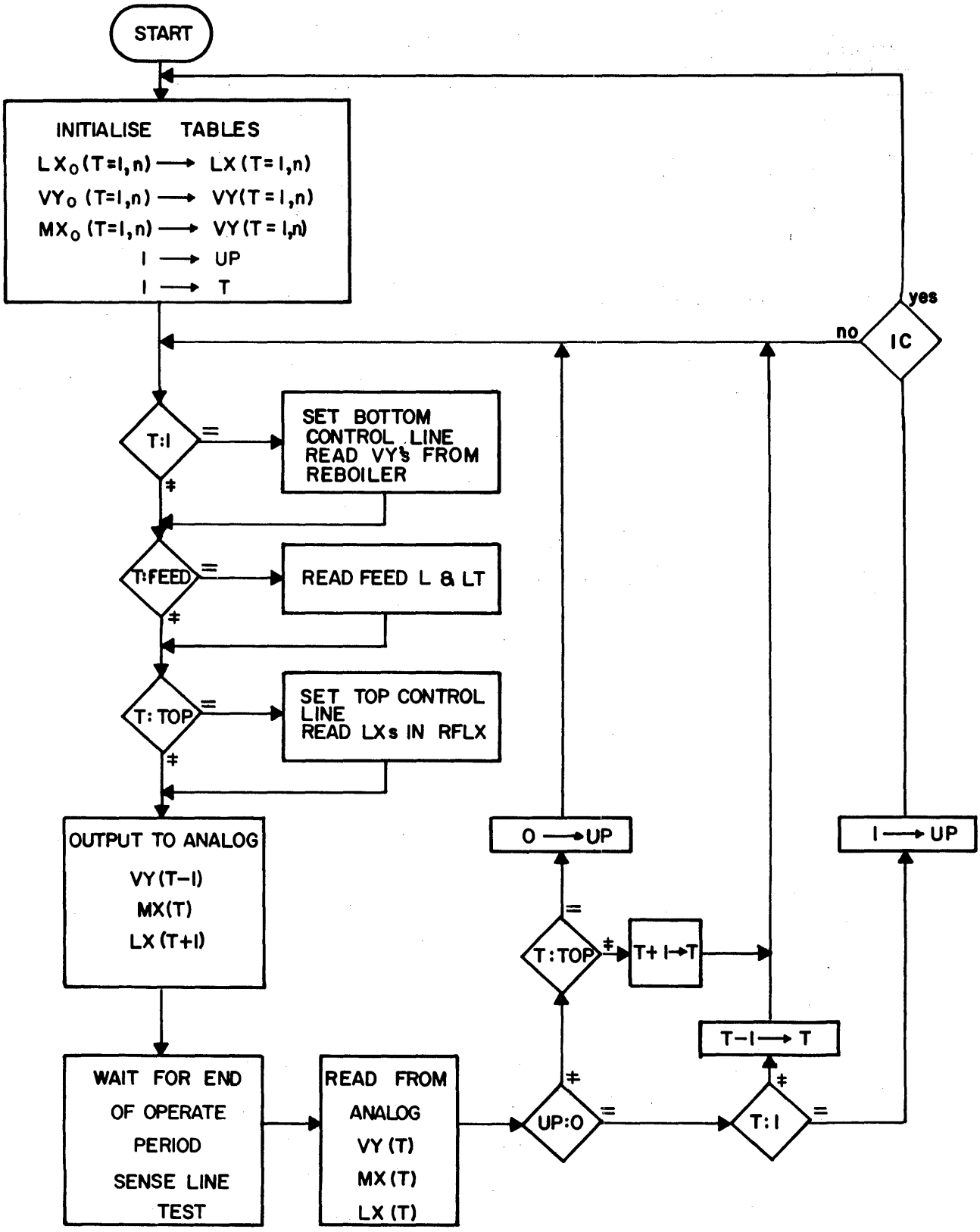


Figure 7. Digital control program.

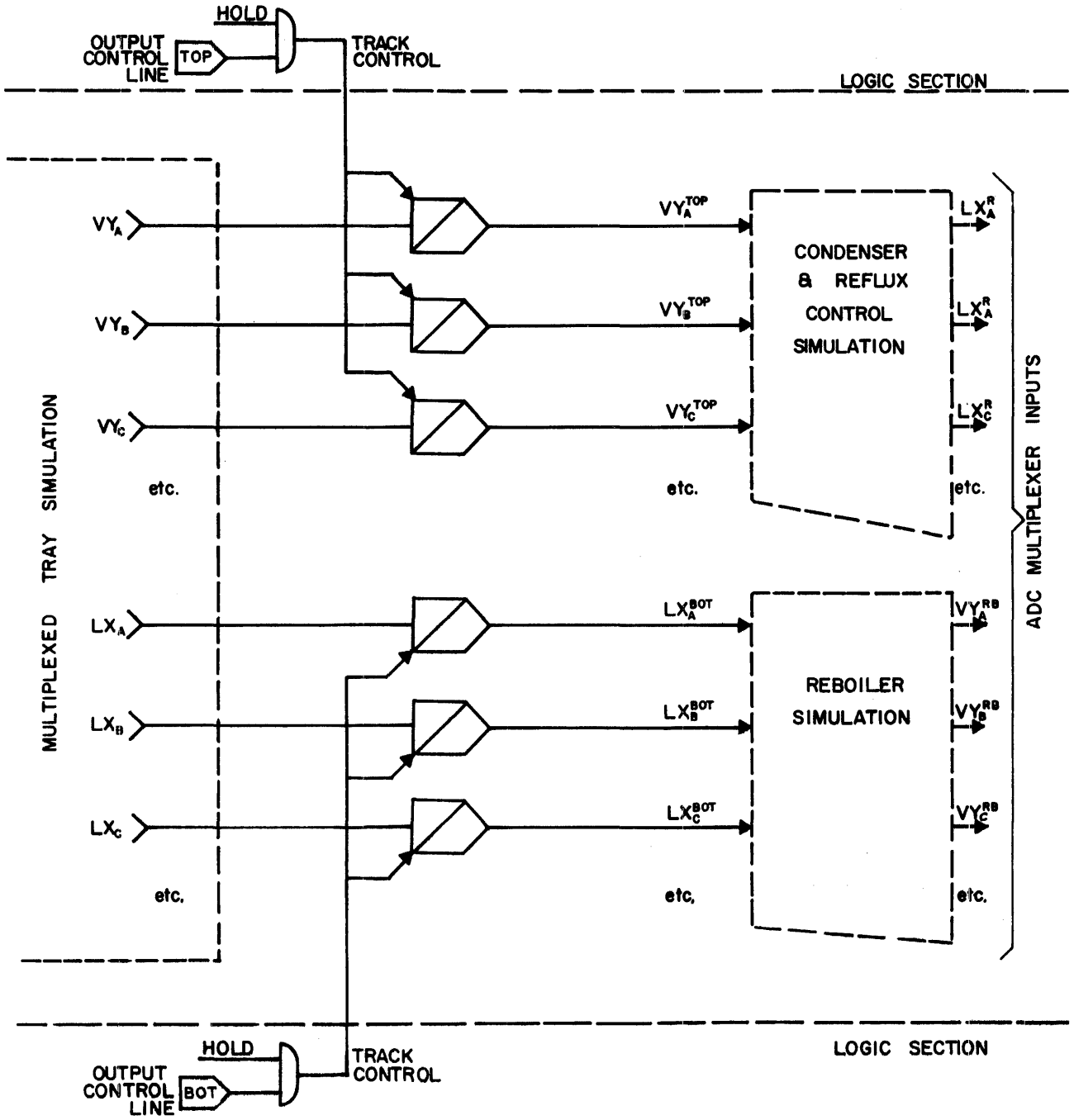


Figure 8. Sampling fluxes from top and bottom tray simulations.

Vapor flow into the tray comes from tables

$$VY_A(T-1),\ VY_B(T-1),\ VY_C(T-1),$$
$$V(T-1),\ \bar{c}VT(T-1)$$

State variables (hold-ups) come from tables

$$MX_A(T),\ MX_B(T),\ MX_C(T),\ MX_D(T),\ MT(T),\ P(T)$$

Liquid flow onto the tray comes from tables

$$LX_A(T+1),\ LX_B(T+1),\ LX_C(T+1),$$
$$L(T-1),\ \bar{c}LT(T+1)$$

When these levels have become established ($\sim$ 2 msec), the analog computer integrates for the 8-msec OPERATE period which is then folowed by

the HOLD mode. During this time, the new state variables and fluxes from the tray that will be used to force adjacent trays can be read and stored—in the T slot in the tables.

Scaling considerations suggest that it is better to read a change rather than a magnitude and so the analog computer is programmed to calculate the change in flux through the plate, i.e., VY_A^{T+1} . VY_A^T, etc., so that the addition to obtain the actual magnitude can be carried out at the full precision of the digital section.

The logic to control the sequencing up and down the column is straightforward. Normal operation increments or decrements T according to whether UP is one or zero respectively. At the ends when $T = 0$ or $T =$ TOP, UP is changed over appropriately.

Tests for scale factor changes are not shown, but at fixed trays, the scale factors in the tables could be changed by factors of 10. An interesting possibility is dynamic rescaling where the digital computer would itself determine whether overflow were likely and automatically change scale. Now the interfaces between scale factor change would depend on the column operating condition which would help considerably in evaluating transient conditions.

With the scale factor changes, D/A switches have to be controlled, and this is accomplished by extra output control lines (OCP's).

CONCLUSIONS

We have shown how an extremely complex representation of a reacting distillation column and ancillary equipment can be represented by multiplexing a high-speed analog single tray representation up and down the column under digital control. In actual fact, the demands on the digital section are minimal since very little logic or arithmetic is required, the principal requirement being for storage. Even for this, a memory size of less than 4K would be adequate. Another point worth mentioning is that all control and data transfer operations are performed during the reset or hold periods of the analog section, so no high-speed interaction with an operating system is required. Thus, providing the conversion channels were sufficient in number ($\sim 30 \times 30$), one of the newer small input/output digital computers would serve for the control and storage section of the simulation.

The technique for adjusting scale factors dynamically or at fixed points in the column certainly bears consideration since in this way dynamic ranges over 100,000 to 1 can be achieved—and these actually occur in typical industrial distillation columns.

REFERENCES

1. G. Marr, "Distillation Column Dynamics, Suggested Mathematical Model," AICHE Baltimore, 1962.

2. A. Frank and L. Lapidus, "Hybrid Computation Applied to Countercurrent Process," *Chemical Engrg. Progress*, vol. 60, no. 4, pp. 61–66 (Apr. 1964).

TRANSIENT NEUTRON DISTRIBUTION SOLUTIONS BY COMPRESSED AND REAL-TIME COMPUTER COMPLEXES

J. E. Godts
Westinghouse Atomic Power Division
Pittsburgh, Pennsylvania

INTRODUCTION

Core transient neutron flux distribution currently constitutes one of the most intensively investigated phenomena in nuclear technology. Independently, nuclear physicists and system analysts attempt to develop solutions to the problem. The arsenal of mathematical tools presently available to them covers most of the spectrum ranging from precise multigroup digital computations to adiabatic core analog analysis. The former involve a series of steady state or quasi-steady state situations, hence are not acceptable to the transient analyst. The latter are too inadequate for detailed safeguard or control analysis, thus forcing the designer to increase his conservative margin.

However, as seen in Fig. 1, a workable compromise has been achieved by the author based on the modal synthesis. Although it is not claimed that this constitutes the best possible approach, it is considered to be an extremely useful tool for solving present problems. Figure 2 demonstrates the compromise which the proposed method offers in the modal-nodal plan. Figure 3 illustrates, by graphical

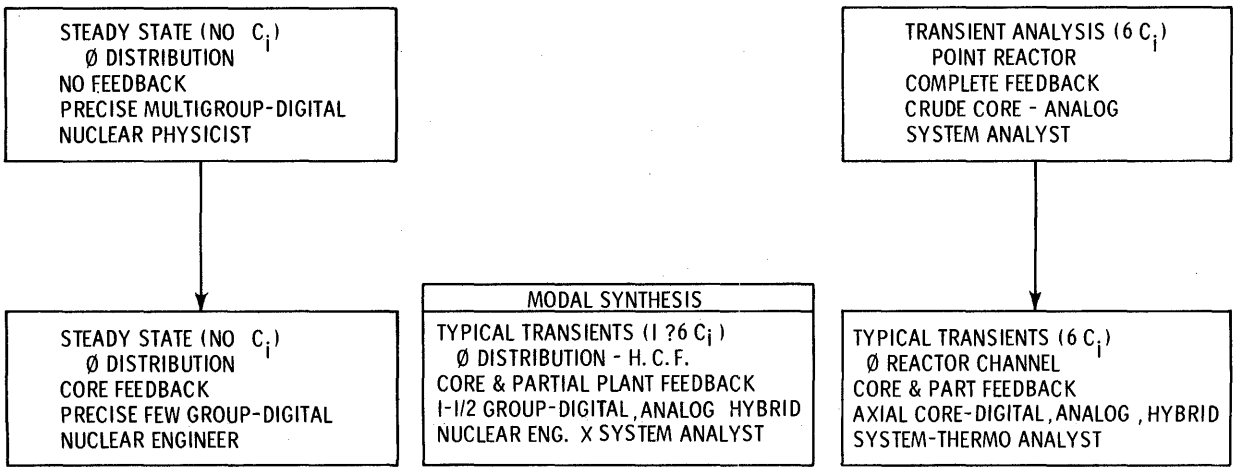


Figure 1. C_i = delayed neutron group; ϕ = neutron flux; HCF = hot channel factor.

equivalence, the various flux descriptions. The mathematical model is the same in all approaches; only the proposed solutions differ largely in their form and in their goal. The choice between difference (or nodal), modal (or series of eigenvalues) and synthesis (or series of functions) is influenced by the major discipline of the analyst and his goal. Quasi-steady state solutions are best sought by difference digital methods, while control design problems are generally best approached by methods based upon use of an analog computer.

From the above general discussion emerges a mathematical solution based on a modal-synthesis method.[1,2] For one spatial dimension, no such modal-synthesis is needed, since the analog equipment can directly perform the iterations for the boundary conditions for the solution described by the neutron diffusion equation, as shown in this paper. For two spatial dimensions, decomposition in basic modes is necessary.

This paper proposes to show that the different basic processes necessary to solve a two-spatial-dimension transient diffusion equation can be performed satisfactorily with present equipment. These basic operations are:

1. Iteration processes on a boundary condition of the diffusion equation in a discontinuous medium.
2. Double iteration processes on the above (two conditions).
3. Iteration of a time-dependent diffusion equation in one dimension.

The circuitry and results are briefly described and some of the computer solutions obtained are shown. This type of computation need not be limited to repetitive real-time analog computer systems, but can be well adapted to hybrid computation. In the near future, such simulations will be performed with the memory extension achieved digitally and with the iterations made on the analog computer. The main (and indirectly the only) reason for reserving the analog computer for the iterative process

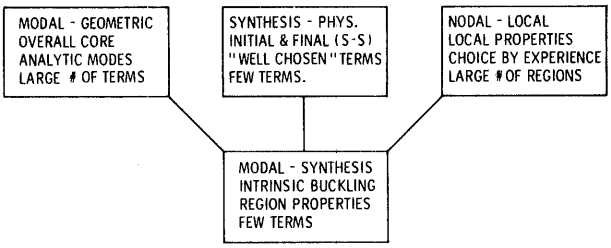


Figure 2.

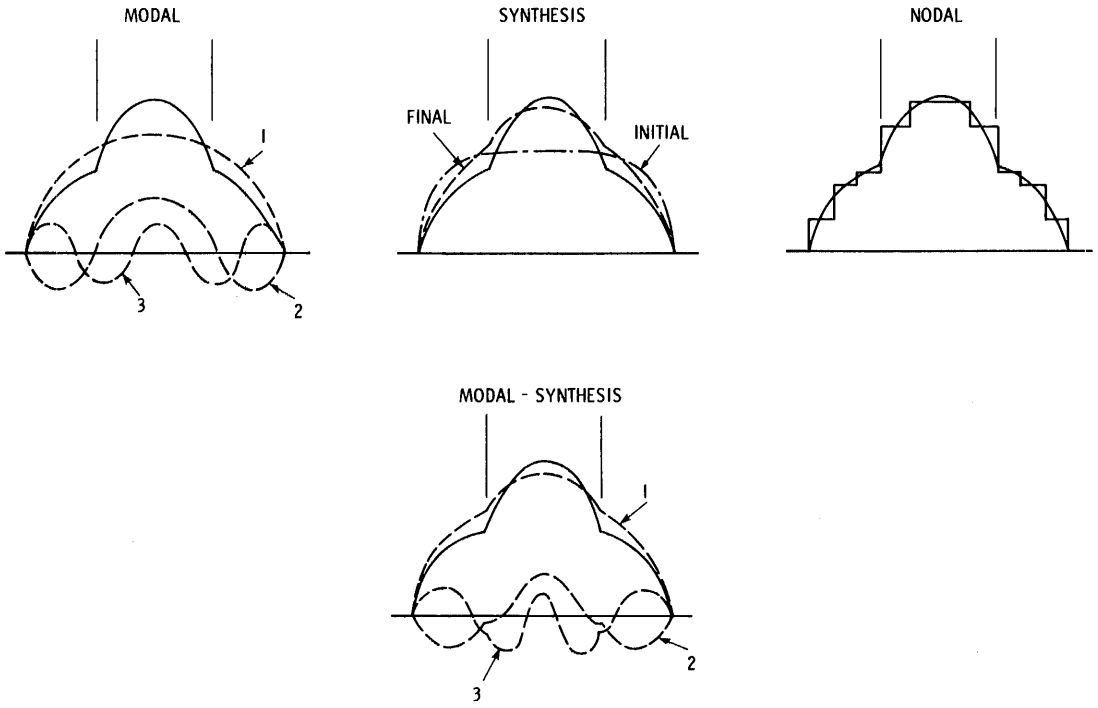


Figure 3.

and plant simulation is that the control designer must, at least at present, be aware of the plant reactions. He must then modify the design according to the response trend as directly observed on the display of the computer solution. The busy control designer cannot afford to wait for the answers of large digital codes, where only a few cases can be seen at a time. Furthermore, the design processes are not sufficiently perfected to where machines can perform the complete tasks. The nuclear plant designer must continually alter the mathematical model details in order to decrease the overdesign margins imposed by conservatism due to insufficient knowledge. Although such new models are easily introduced in analog simulations, they necessitate costly modifications to existing codes.

For these reasons, analog-hybrid simulations have a possible life expectancy of 10 to 20 years, thus some hardware expenditures may be justified.

MODEL DEFINITION

The system analyst confronted with the transient study of a nuclear power plant faces two problems:

1. *Safety problems:* Checking to make sure that no safety limitations are exceeded.
2. *Control problems:* Designing the controller so as to insure that no design limitations are exceeded.

Both problems are studied on the same or on different models. The trend to limit design over con-

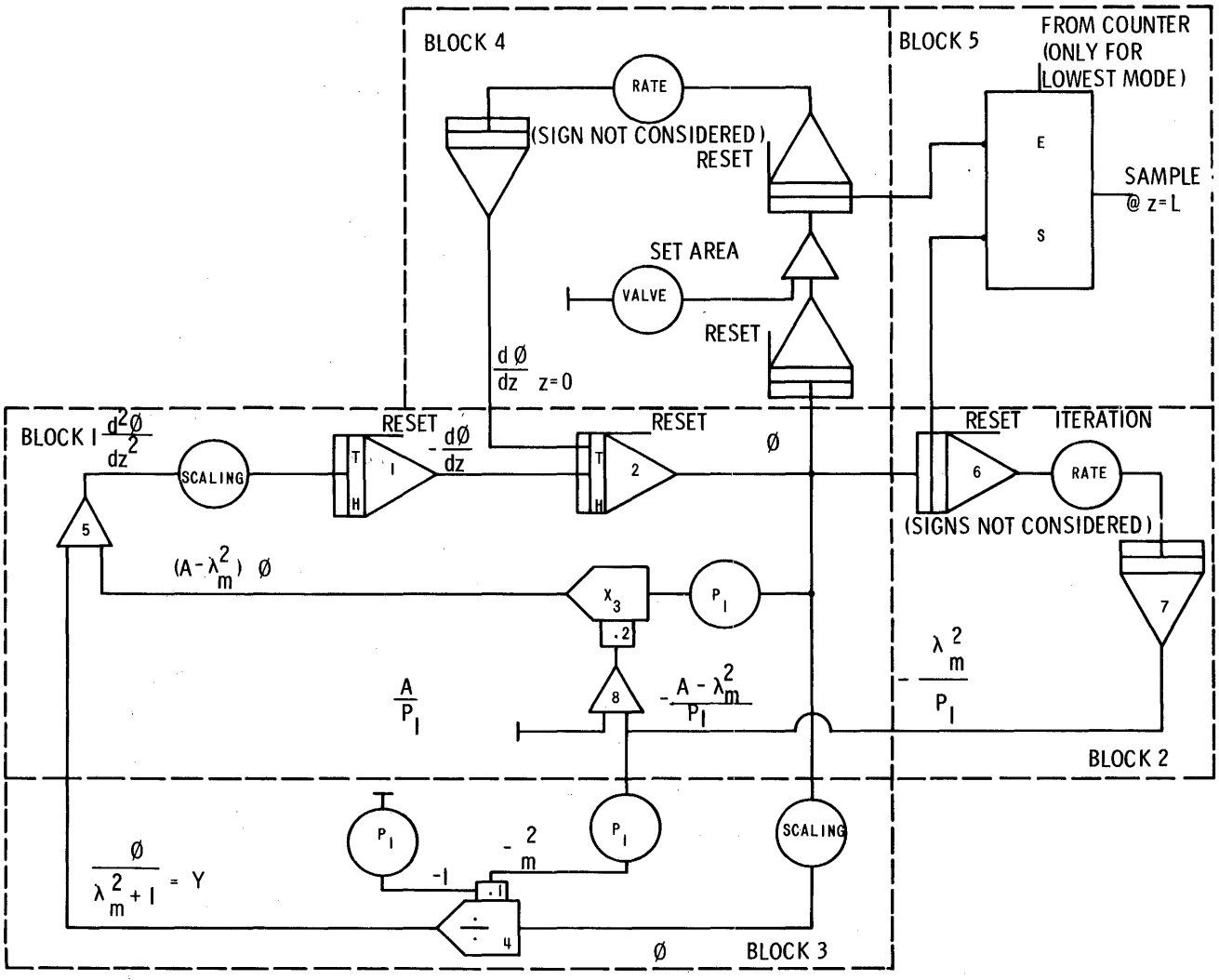


Figure 4.

servatism present in certain core designs makes necessary the transient spatial description of the neutron flow. The development of highly sophisticated hybrid computer complexes provides the system engineer with more versatile tools.

In order to solve the three basic problems stated in the Introduction, the study was divided into four parts.

The first part deals with solving the one-dimensional model, which consists of the neutron kinetics equation eigenvalue problem. No delayed neutron equation is considered in this case.

The second part introduces the delayed neutron equation and studies the eigenvalue problem of the system of the equation proposed.

The third part employs the equation without delayed neutron and operates a second series of iteration on the normalization of the result.

The last part solves a one-dimensional time-dependent diffusion equation.[4]

For the first three parts, only the compressed time repetitive computer (GPS) is used. Only in the fourth part is the simulation performed on the computer complex consisting of real time (EA) and compressed time (GPS) elements.

Simple Problem Models, Circuitry and Displays

The eigenvalue problem corresponding to the one-dimensional model, with or without delayed neutron and with or without normalization, is expressed in a general matricial form as:[3]

$$L\Phi - \lambda_m^2\Phi = 0$$

where L = a square matrix operator,
Φ = a column matrix of flux and delayed neutron flux, and
λ_m^2 = the mth eigenvalue solution.

The problem can also be expressed as:

$$\frac{d^2\Phi}{dz^2} + A(z)\Phi + \gamma = \lambda_m^2\Phi \qquad (1)$$

$$\Phi - \gamma = \lambda_m^2\gamma \qquad (2)$$

The eigenvalue λ_m^2 must be found by iteration, and the circuitry used is developed in steps.

If no delayed neutrons are present, and no normalization is operated, Eq. (1) is expressed in a much simpler form:

$$\frac{d^2\Phi}{dz^2} + A(z)\Phi = \lambda_m^2\Phi \qquad (3)$$

The fact that A is a function of z was illustrated in our example by a step in value at a given value of the independent variable.

Using the circuitry illustrated in Fig. 4 (Blocks 1 and 2), the displays obtained are given by Figs. 5, 6 and 7. To illustrate the work performed by the machine during a very slow iteration process, the convergence towards the solution from two different initial conditions is shown by Figs. 8 and 9.

The mentioned "control rod depth" corresponds to the location of the step in the value of A.

To check the stability and reality of a given solution, the inversion of the properties along the independent variable z was performed (Figs. 10 and 11). The obtained results were perfectly symmetrical. (The iteration is always performed at the right boundary.)

The large discontinuities of material, corresponding to the control rod boundaries in nuclear reactors, result in large discontinuities in the second

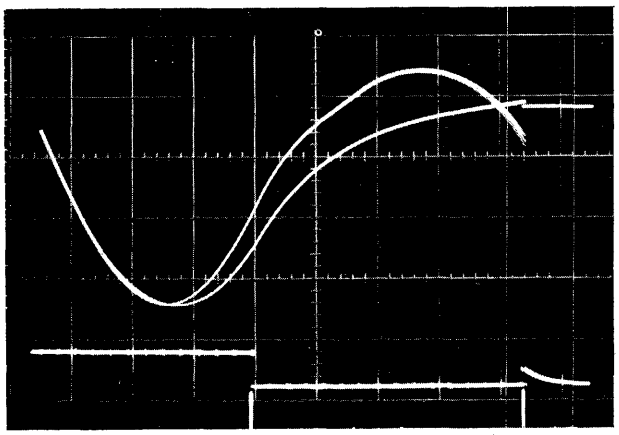


Figure 5. The first two modes. Vertical white line (*fourth square from left*) indicates control rod depth.

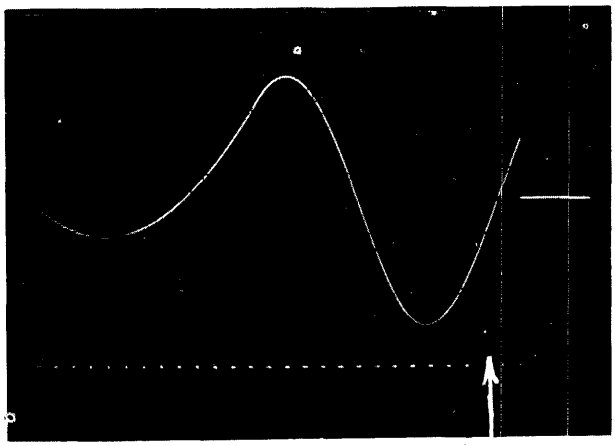


Figure 6. Third mode (iteration not completed). Arrow points to location of $x = 2L$.

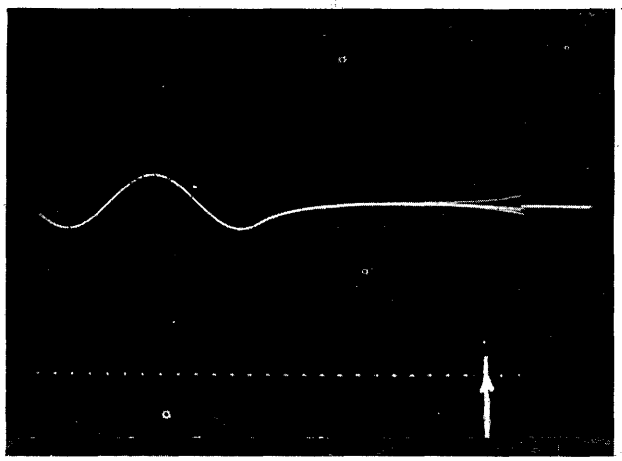


Figure 7. The fourth mode. Arrow points to location of $x = 2L$.

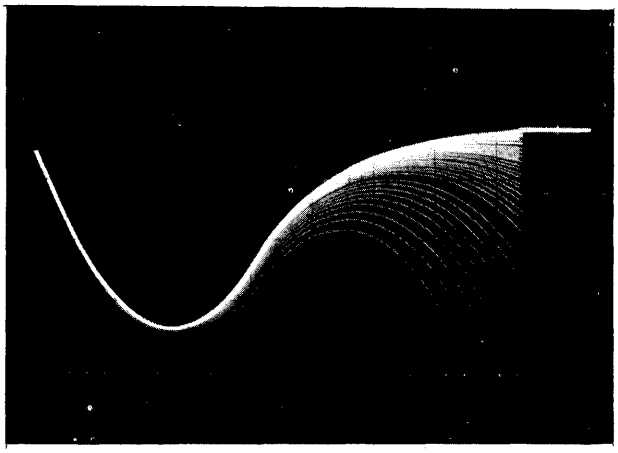


Figure 8. Boundary value search for fixed initial tangent search on λ_m^2, starting point with a large λ_m^2.

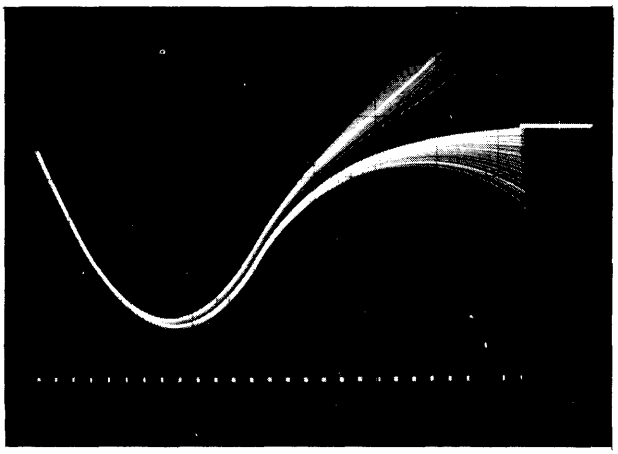


Figure 9. Boundary value search for fixed initial tangent search on λ_m^2, starting point with a small λ_m^2.

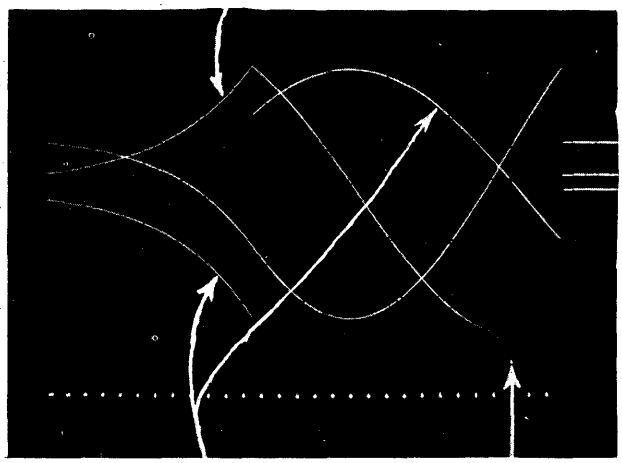


Figure 10. The first mode, its first and second derivative. Arrow (*top*) points to first derivative; double arrow (*lower left*) to second derivative; arrow (*right*) to location of $x = 2L$.

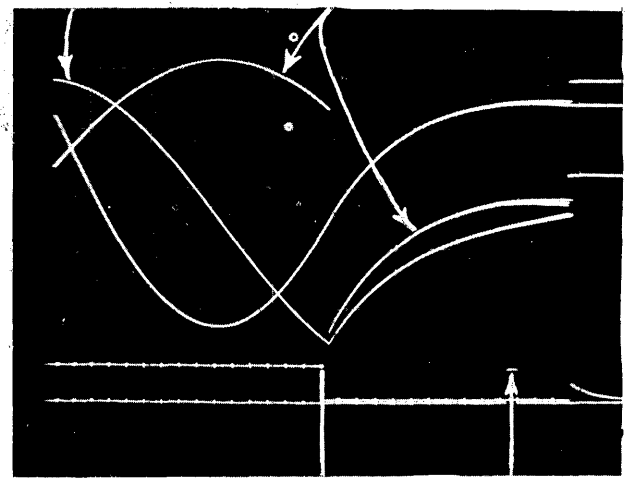


Figure 11. Reversed iteration of Fig. 10. Arrow (*top left*) points to first derivative; double arrow (*top center*) points to second derivative. White vertical line (*lower left*) indicates rod insertion depth. Arrow (*lower right*) points to location of $x = 2L$.

derivatives (e.g., Fig. 10). If the choice of the spatial variable direction is free, the preferred solution has the highest possible gradient at the boundary of iteration (Fig. 10). Since the solution will converge more rapidly, it will be more precise and more stable.

The higher mode solutions were somewhat more difficult to obtain than the fundamental, and methods should be devised to scan automatically the successive values of λ_m^2 (Figs. 5, 6 and 7).

To show the adaptability of the circuit, a small slug of reactive material was introduced into a scattering material. Successive modes were easily obtained (Figs. 12, 13 and 14).

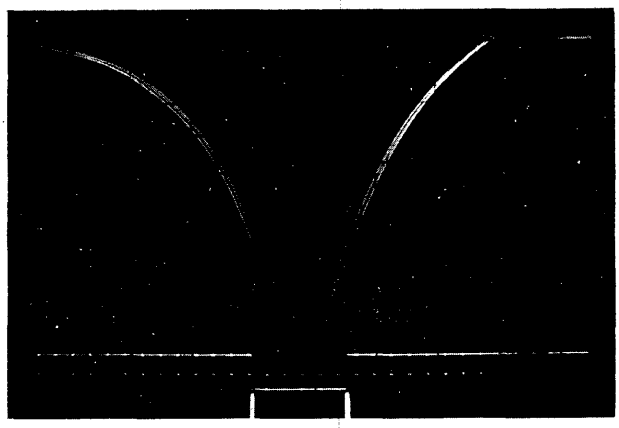


Figure 12. First mode slug case. White vertical lines (*bottom*) indicate position of the slug.

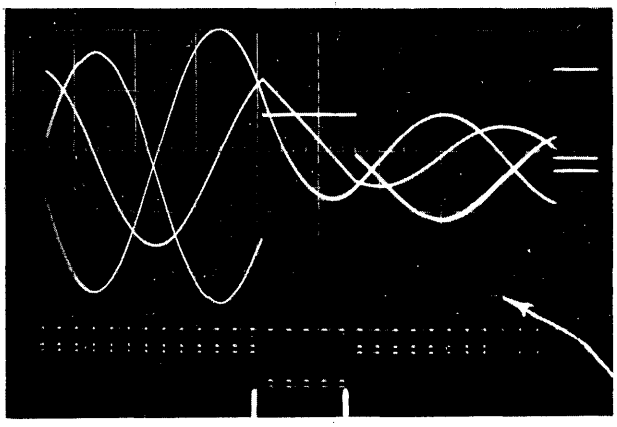


Figure 13. Fourth mode slug case, first and second derivatives. White vertical lines (*bottom*) indicate position of the slug. Arrow (*right*) points to location of $x = 2L$.

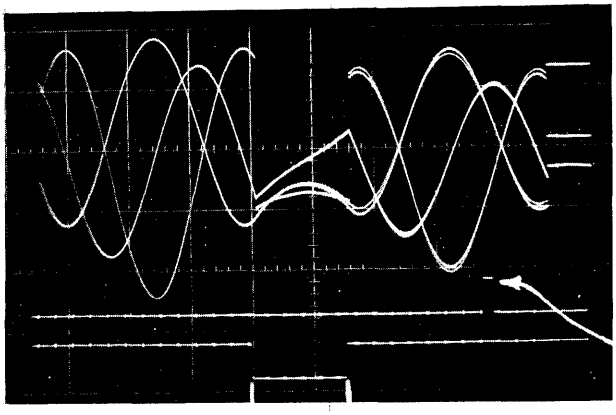


Figure 14. Sixth mode slug case, first and second derivatives. White vertical lines (*bottom*) indicate position of the slug. Arrow (*right*) points to location of $x = 2\ L$.

Introducing Delayed Neutron into the Equation

This introduction is performed by adding block 3 (Fig. 4) to the circuit. If one delayed neutron group (precursor) is introduced, each nodal solution comprises two orders.

The physical meaning of the two orders is found in the initial conditions imposed in the problem. In the case of one delayed neutron group problem, the suitable initial conditions consist of a flux distribution and a delayed neutron distribution. It is assumed that such distributions follow the first mode curve. The relative amplitude of flux and delayed neutron distribution must be determined and the initial condition must fit them. This is only possible if two orders exist, since two conditions must be satisfied. It can be further demonstrated that the relative sign of the delayed neutron and the flux distribution is the same in one of the orders, and that the delayed neutron distribution has the

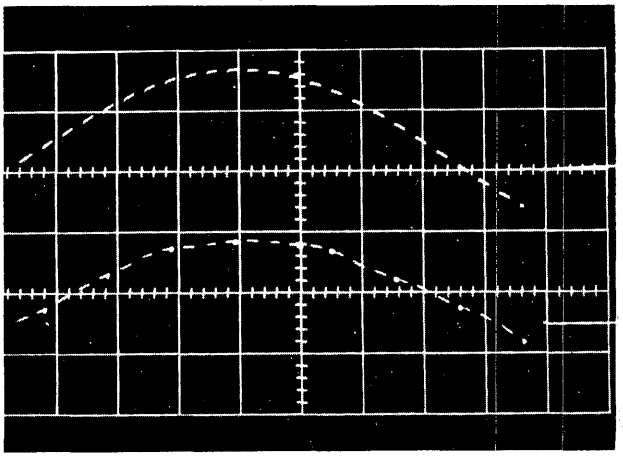


Figure 15. First order of first mode.

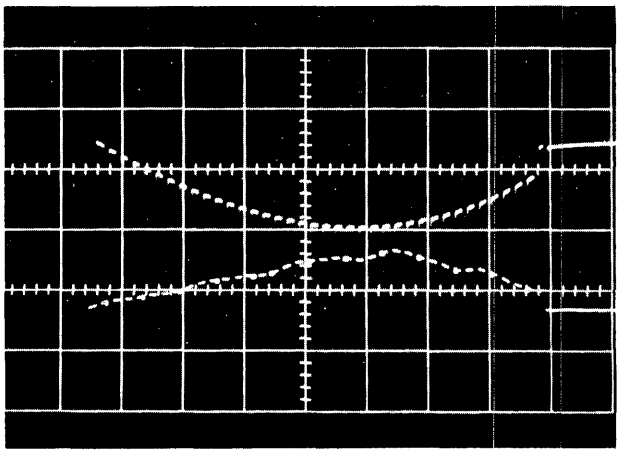


Figure 16. Second order of first mode.

opposite sign of the flux distribution in the other order.

It is proved mathematically that the orders give different relative sign of the flux and the delayed neutron.[2]

In the search for the second order, the sign of the divider must thus be inverted. The iteration process will then proceed normally towards the solution. To avoid making a complete revision of the scaling, no changes were made in the simulation; only the display amplification was changed. This explains the large amount of noise present in one of the solutions (Fig. 15 and 16).

The Solution with Two Iterations

To perform this operation, the delayed neutron circuit was rendered inactive (block 3 and block 4 were introduced).

Only the fundamental mode is of interest for this transient study. An absorbing rod was suddenly removed from the core, and the subsequent conditions imposed are that:

1. The flux vanishes at the outer boundary.
2. The area enclosed by the flux distribution curve is constant.

Figure 17 shows the resulting display. The two iterations are performed alternatively, each for 15 machine cycles, by the introduction of block 5 (Fig. 4). In practice, this large number of iterations cannot be maintained; it was successfully reduced to three or four machine cycles per iteration.

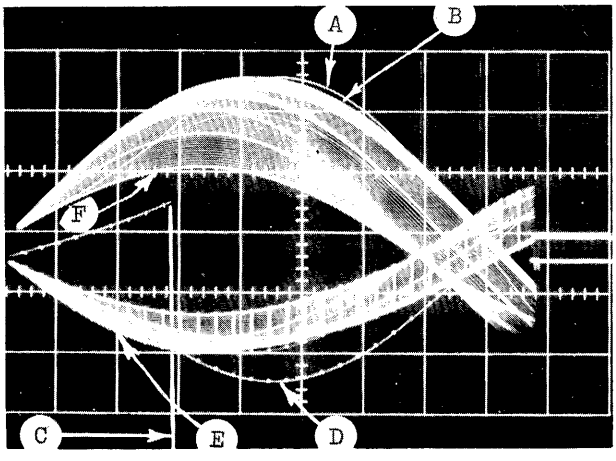


Figure 17. Constant area problem during rod removal process. A—initial flux distribution; B—final flux distribution; C—initial rod insertion depth; D—initial second derivate curve; E—influence of the iteration process on the second derivate distribution; F—initial flux jump.

When the procedure performs adequately, a modification of the area should not change the extrapolated boundary conditions. A test was conducted to check this statement by inverting the sign of the required area. Only four machine cycles per iteration were retained. The result appears in Fig. 18.

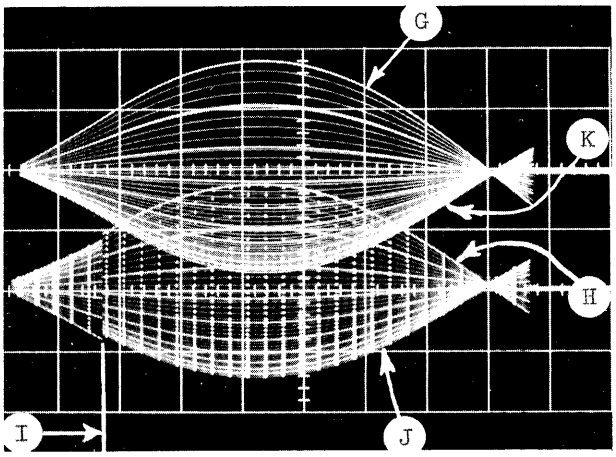


Figure 18. Area inversion at fixed rod position. G—initial flux distribution; H—initial second derivative curve; I—rod insertion depth; J—final second derivative curve; K—final flux distribution with area inverted.

One-Dimensional Time-Dependent Diffusion Equation

The problem was introduced in the following mathematical form:

$$\frac{\partial^2 \Phi}{\partial z^2} + A(z,t)\Phi = a\,\frac{\partial \Phi}{\partial t}$$

The solution is approached by using a time difference equation of the following type:

$$\frac{d^2\Phi(t+\Delta t)}{dz^2} + \left(A - \frac{a}{\Delta t}\right)\Phi(t+\Delta t) = \frac{a}{\Delta t}\,\Phi(t)$$

The function $\Phi(t)$ is stored in a memory after each iteration is performed, and thus can be used for the next time step. The block diagram used in shown in Fig. 19.

Each new circuitry has already been explained. The $A(z,t)$ function was formed by the intervention of the fast repetitive computer for the z dependency, and the real-time computer for the time dependency.

The task of the real-time computer was to simulate the whole plant response. This was done on a 1:1 time basis. The memory used was formed with elements of the GPS computer[1] and does not pre-

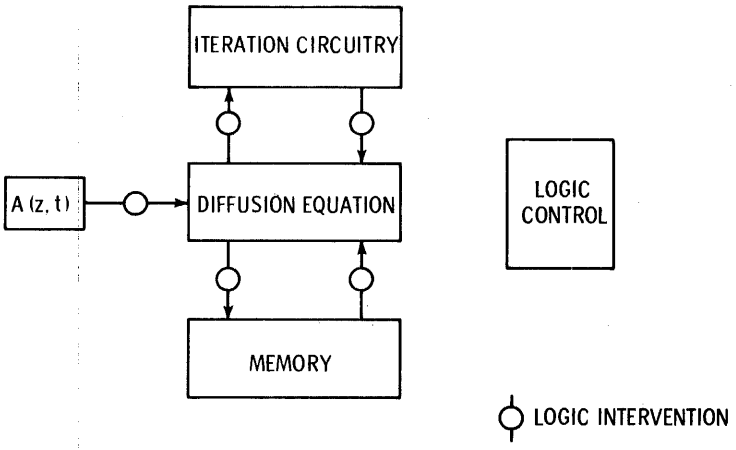


Figure 19.

sent any originality. This is certainly the part of the circuitry that most needs to be perfected. For future simulations, it is intended to use digital memories of various types.

In its present form, the memory circuitry is inadequate since the number of retention points is too small and necessitates a far too large an amount of equipment.

Nevertheless, it was tried on a rod insertion and withdrawal problem. The processes provide the displays shown in Figs. 20 and 21.

CONCLUSIONS

The search for the modes and orders of the neutron diffusion equations can be performed satisfactorily on analog computers.

The one-dimensional transient flux distribution can be successfully simulated as part of a complete nuclear plant simulation. It is, nevertheless, necessary to improve the memory capacities of the present machines by further hybridization of the complexes.

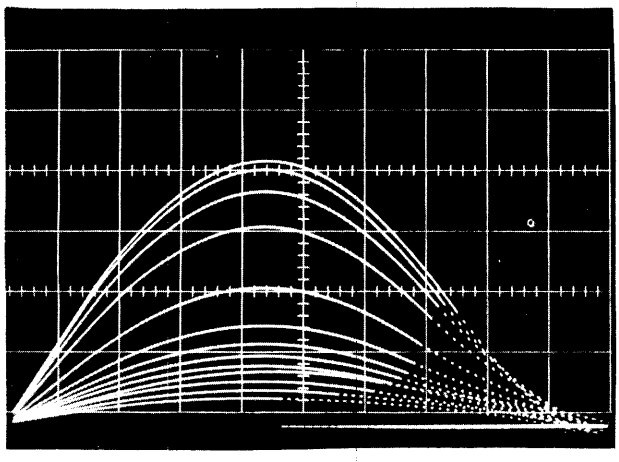


Figure 20. The rod insertion process. Dotted lines represent extent of rod depth.

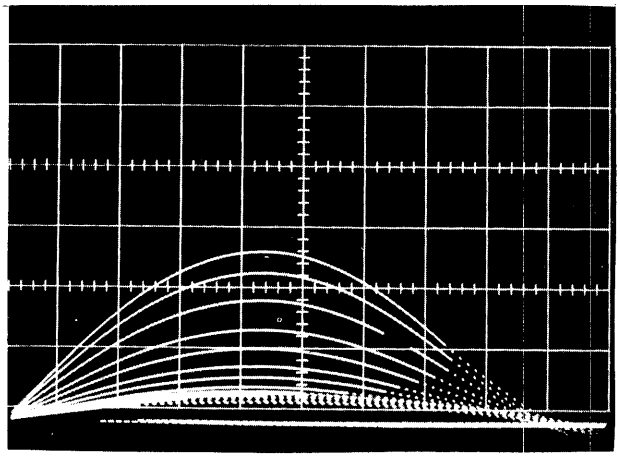


Figure 21. The rod removal process. Dotted lines represent extent of rod depth.

The two-dimensional transient flux distribution can be performed, but present equipment requires too large amounts of circuitry for a safety and control analysis of nuclear reactors. Nevertheless, average methods, as described in Refs. 1 and 2, should prove adequate for present needs. They are based on the studies described in this paper.

The introduction of simplified circuitry or functionals in the simulation will permit future simulation of the two-dimensional flux distribution within a complete nuclear plant study.

Simplified circuitry and memory equipment might even enable the introduction of more than one group of delayed neutrons.

No difficulty was encountered in the interconnec-

tion of the slow real-time computer (EA 231R) and the fast repetitive computer (GPS). An auxiliary problem of a rod ejection accident in a thermal reactor was successfully tried on the circuit. The answers were rapidly obtained and with sufficient accuracy for qualitative evaluation.

The limiting factors for such large problems (a complete problem uses two GPS, two EA 16-31R and one EA 231R plus logic and memory) are the necessary zeroing time and the equipment failure rate. It is foreseeable that units could be designed with self-zeroing and self-checking features. These operations could be performed automatically during the off times of the machines. With the use of the above-mentioned functionals reducing greatly the amount of equipment, large multivariable problems will then be within the reach of the analog hybrid simulators and will certainly be welcomed by the nuclear system engineer.

REFERENCES

1. J. E. Godts, "Analog Analysis of Transient Neutron Flux by Discontinuous Synthesis," Ph.D. thesis, Carnegie Institute of Technology, 1965; also WCAP-2828, Westinghouse Atomic Power Division (July 1965).

2. J. E. Godts and E. F. Restelli, Jr., "Transient Flux Distributions by Discontinuous Modal Synthesis," WCAP-2755, Westinghouse Atomic Power Division (July 1965).

3. J. E. Godts and A. S. Weinstein, "Transient Flux Distributions by Discontinuous Modal Synthesis," Conference on Safety, Fuels and Core Design in Large Fast Reactors, Argonne National Laboratory, Oct. 11–14, 1965.

4. J. E. Godts and E. F. Restelli, Jr., "Transient Temperature Distribution in Fuel Elements," ibid.

BIBLIOGRAPHY

Kaplan, S., "Extensions and Applications of the Method of Finite Transforms," Ph.D. thesis, University of Pittsburgh, 1960.

———, "Some New Methods of Flux Synthesis," *Nuclear Science and Engineering*, vol. 13, pp. 22–31 (1962).

Radkowsky, A., *Naval Reactor Physics Handbook*, Vol. I, U.S. Atomic Energy Commission, U.S. Government Printing Office, 1964.

PATTERN RECOGNITION STUDIES IN THE BIOMEDICAL SCIENCES*

Robert S. Ledley, Louis S. Rotolo, Marilyn Belson, John Jacobsen,
James B. Wilson, and Thomas Golab
National Biomedical Research Foundation
Silver Spring, Maryland

The biomedical sciences characteristically deal with huge masses of data, which must be organized, reduced, analyzed, and generally processed in many different ways.[1] Much of this data is in the form of pictures: photomicrographs, electron micrographs, X-ray films, Schlieren photographs, X-ray diffraction patterns, autoradiographs, time-lapse films, cineradiographs, or the like. Individual pictures hold a great wealth of precise numerical information, such as the morphological and structural characteristics of lengths, areas, volumes, and densities. From sequences of pictures, quantitative results can be derived, such as the kinematic and dynamic characteristics of trajectories. Such pictures relate to almost every field of biomedical research: chromosome karyograms in cytogenetics, angiogram cineradiographs in cardiology, Schlieren photographs in ultracentrifugal molecular-weight determinations, autoradiographs of polymorphonuclear leukocytes in the study of leukemia, Golgi-stained neuron photomicrographs in the study of the ontogeny and phylogeny of the brain, X rays of bones in studies of calcium density distribution in orthopedic diseases, X rays of epiphysial plates of the hand in investigations of accurate physiological age, X-ray crystallographic plates in protein structure determination, electron micrographs in the investigation of the fine structure of virus particles, motion pictures of marine crustaceans in the detection of their sensitivity to polarized light, tissue-culture time-lapse films in the investigation of cancer-cell motility, and many others.

In this paper we shall describe selected illustrations of work already accomplished by the authors in the pattern-recognition analysis of biomedical pictures. The technique involves two main steps: first, a scanning instrument, called FIDAC (Film Input to Digital Automatic Computer), scans the picture on-line into the high-speed memory of a digital computer; second, two computer programming systems (called FIDACSYS and BUGSYS) are used to recognize the objects to be measured and to process the quantitative data, according to the requirements of the particular biological or medical problem under consideration. This FIDAC system was designed specifically for the processing of biomedical pictures.

THE FIDAC INSTRUMENT

The FIDAC instrument is responsible in the first place for putting the picture into the computer's memory; it is an on-line computer-input device which can scan in real-time at very high speed and with high resolution, under computer-program feedback control (see Fig. 1). These capabilities make feasible the successful application to biomedical problems. FIDAC has a high-speed scan of less

*This work was supported by National Institutes of Health grants GM-10797, GM-10789, NB-04472, GM-11201, and AM-08959.

than 0.3 seconds per frame, which makes possible the rapid processing of pictures for statistical analysis and screening purposes. During the scan, 350,000 points per picture (700 × 500 point raster) are sampled in the seven-level gray mode, which uses three memory bits per picture point, or 800,000 points per picture (1000 × 800 point raster) are sampled in the black-and-white mode, which uses one memory bit per picture point. As a comparison, during the scan the FIDAC can load the computer's core memory 10 to 50 times faster than can conventional magnetic tape input units. (The IBM 7094 computer, equipped with a direct data channel, is used.)

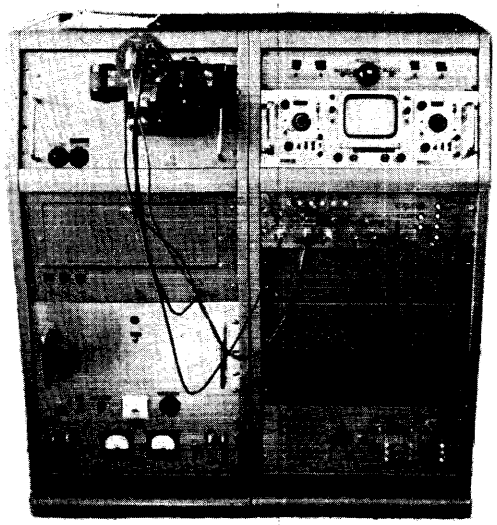


Figure 1a. The FIDAC Instrument.

FIDAC has a high resolution (greater than that of the optical microscope), which enables the retention of all information when photomicrographs are scanned. This is because when the specimen is seen at a magnification of 1000 ×, the field has a diameter of about 50μ; thus the 750 points sampled by FIDAC across this field gives about 700/50 = 14 points per micron on the specimen, or about 3 points in the 0.2μ, that is, the optical resolution of a microscope at 1000 power.

The capability of real-time operation enables program control of the FIDAC by the computer—that is, when the processing of a film frame has been completed, the program signals the FIDAC to move automatically to the next frame. The fact that FIDAC is on-line with the computer, with no intermediate magnetic-tape recording, results in extreme flexibility, convenience, and economy of storage of the original data. For instance, a single 100-foot roll of 16mm film, which fits in a 3½-inch diameter can, contains 4000 frames and will record over 4 bil-

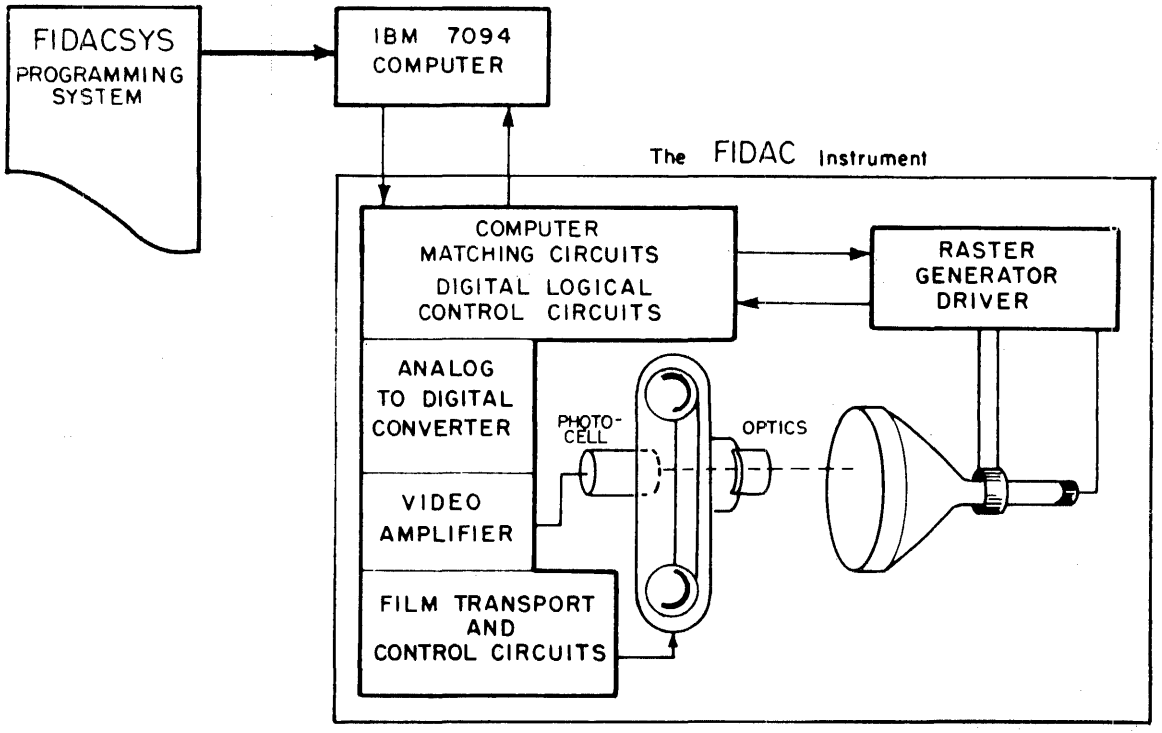


Figure 1b. Block Diagram of FIDAC System.

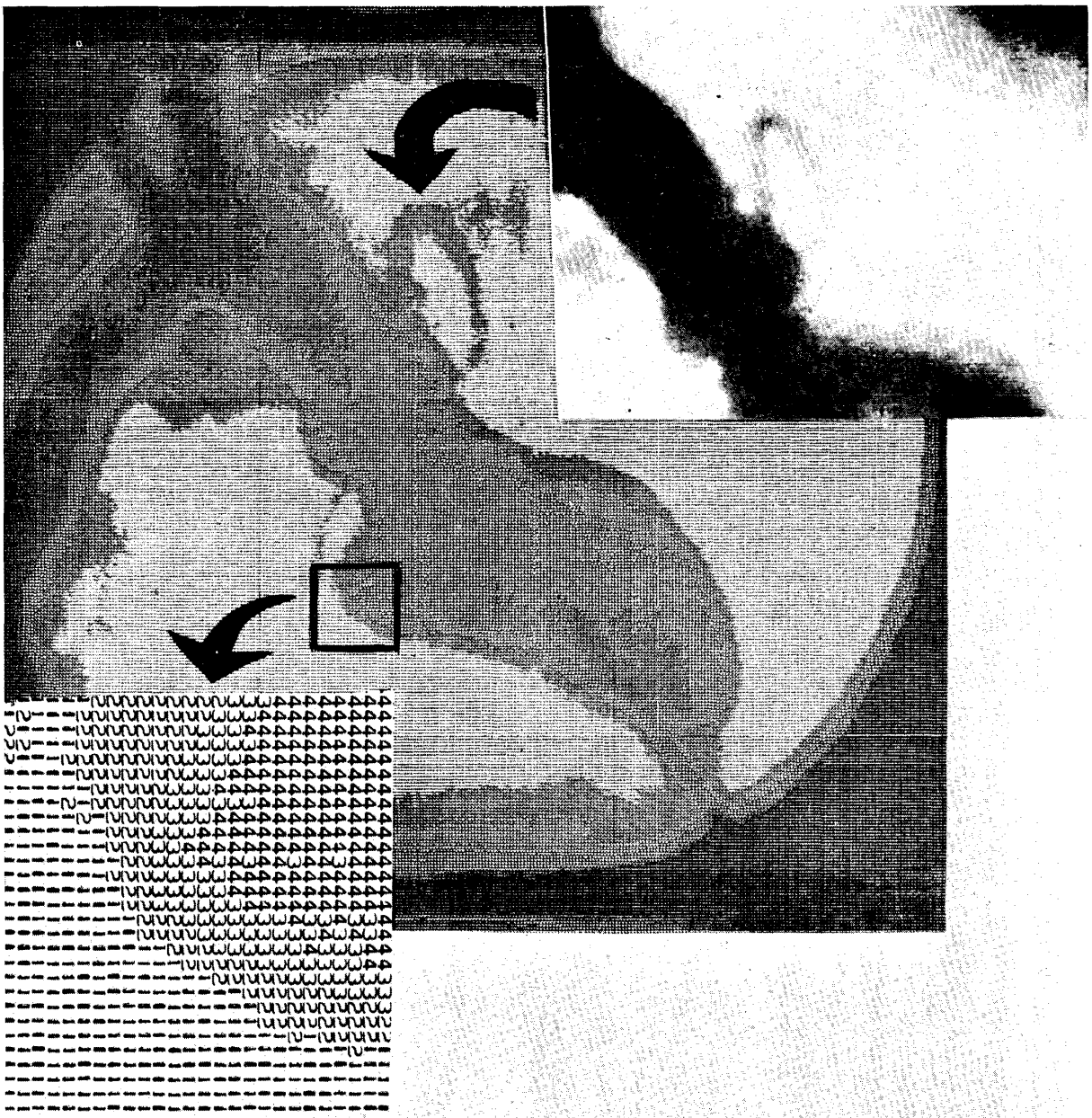


Figure 1c. Cineangiogram of a dog's heart as recorded in the computer's memory. The original photograph is scanned by FIDAC and a 700 × 500 raster of spots is sampled. The density of the film at the location of each of these spots is determined as one of seven gray-levels. The picture in the computer's memory, therefore, consists of codes for these density levels. To show what is in the computer's memory, the computer translates the code for each spot into a numerical gray-level value, and prints the value in its correct relative position for each of the 350,000 spots that comprise the picture. This is what is shown in the figure. In the lower-left inset we illustrate an enlarged portion of the figure showing the density levels. The other inset shows a portion of the original cineangiographic picture.

lion bits of information; this would require more than 55 conventional digital-magnetic tape reels, making a stack more than 4 feet high.

THE PROGRAMMING SYSTEM FIDACSYS

Once the picture is recorded in the computer's core memory, as a grid of points, each with one of seven gray-level values, the computer analysis proceeds by means of the two programming systems. The FIDACSYS system[2] is the basic picture-handling and object-recognition system; the BUGSYS language is used mainly to facilitate picture analysis and measurement.

The first problem for the FIDACSYS system is to read the picture into the computer's memory. The program does this by signaling the FIDAC through the direct data channel. The program monitors the read-in, line by line, keeping track of the lines and the size of the picture. Of course, the program must select the proper FIDAC mode. The program also moves or positions the film. Usually the film is advanced to the next frame by the program immediately after the picture has been read in. This is done during the time the analysis program is being executed, and if the analysis time is greater than the film movement time, namely $1/5$ of a second, then no provision need be made for film movement time. A roll of film is processed frame by frame in this manner. The program keeps track of the number of frames to be processed, and when all of the frames have been processed, the program proceeds to compute the statistical results applicable to the film roll.

In processing each frame, the FIDACSYS system first determines whether or not the frame is blank; if it is not blank, then any required overall picture processing, such as differentiation, is carried out. Next the program locates and processes one object at a time; this is accomplished by an internal, programmed scan of the picture, starting at the top left-hand corner and continuing row by row. We say that this internal scan is accomplished by a "bug," which is looking for a picture spot that has a gray level greater than the "cutoff" gray level. The cutoff gray level is defined such that the interior points of any object will have gray-level values greater than that cutoff level. When an object is found, it is then processed. In this way each object is sequentially processed, until finally, when the scanning bug reaches the lower right-hand corner of this picture, all of the objects have been processed and the program proceeds to the statistical analysis applicable to that frame.

When an object has been found, its processing is carried out by the FIDACSYS program in terms of a boundary analysis. The bug is moved around the boundary of the object in such a direction that the interior of the object is kept to the right. The next boundary point is determined by looking clockwise around the present boundary point, starting from the previous boundary point (see Fig. 2a). When a certain number N of boundary points (that is, a certain boundary length) has been traversed, a segment is defined. The segment is then characterized by the coordinates of its center point, the components of a leading vector, and the components of a trailing vector (see Fig. 2b). The length of the segment chosen must be short enough that the angle between the leading and trailing vectors is approximately a measure of the curvature of the segment. Then the vector sum of the leading and trailing vectors is approximately the tangent to the segment at its center point and gives a measure of the direction of the segment. There are three user parameters associated with a segment, varied to suit the particular problem under consideration. These are the segment length N, the arrow length A, and the distance D between centers of successive segments. A boundary-characterization list is constructed as each boundary segment is analyzed successively, until the original boundary entry point is reached again.

As each boundary point is traversed, its gray-level value is changed by the bug to the value 7, so that the object is completely enclosed in a string of 7's. Only seven gray levels per spot were used in the scan so that the eighth level, namely 7, could be reserved for this purpose. Thus no object is reprocessed, because only objects not enclosed in such a string of 7's are found by the search. If the analysis involves the investigation of holes within the object, then the holes are discovered by carrying out an erasing process. When a hole in the object is discovered during the erasure procedure, its boundary is characterized by segments. Islands within holes are found by filling in the holes, and holes within islands within holes are found by erasing, ad infinitum. (However, only simple holes, with no islands, are found in most applications.)

The recognition of the object by the FIDACSYS programming system is accomplished by means of the boundary-characterization list. A syntax-

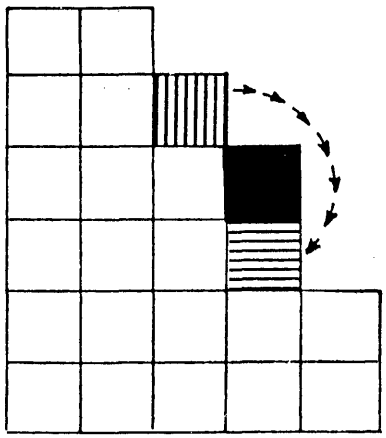


Figure 2a. The black box is the present boundary point, the vertically hatched box the previous boundary point, and the horizontally hatched box the next boundary point.

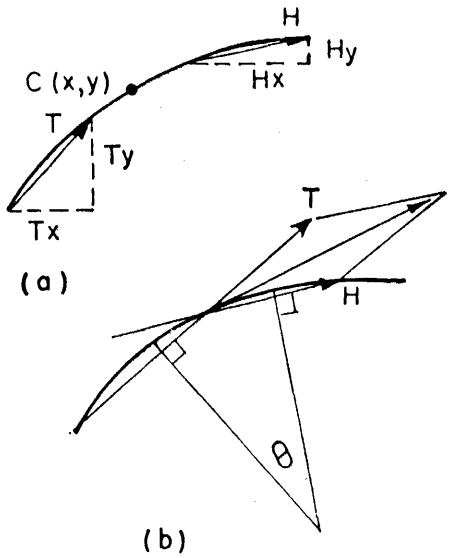


Figure 2b. A segment (above) illustrating the center of the segment $C(x, y)$, the trailing arrow components Tx and Ty, and the leading arrow component Hx and Hy. The tangent vector $T + H$ (below) and the angle $\theta = L/K$, giving $K = L/\theta$, where K is the curvature and L is the segment length (and θ is in radians).

directed pattern-recognition technique is utilized. (An introduction to this technique is described elsewhere.[3,4])

Since in the IBM 7094 memory the picture takes up more than 25,000 words of the 32,000-word memory, the FIDACSYS programming system has to be broken into links, each link being executed separarely in the memory. The initial link enables the user input parameters to be inserted into the program. The second link is concerned with reading-in the picture and "compacting" it into the proper part of the memory, in appropriate form.

The next link will manipulate the picture, will move the film to the next frame if desired, and will print the picture as it appears in the memory if desired. Optional manipulations of the picture include masking, that is, widening the margins of the picture; reducing, that is, decreasing the density of spots of the picture as read into the computer; differentiating, that is, processing the picture to emphasize changes in gray level; and other related processes, as well as printing the picture.

The fourth link scans a picture and processes the objects, as described previously, and is carried out in the three phases. Phase 1 concerns the locating, boundary tracing, boundary-list characterizing, and optional erasing of an object. Phase 2 is concerned with the syntax analysis leading to the identification of the parts of the object. And Phase 3 is concerned with actually making the measurements on the object. The fourth link also accomplishes the overall *statistical processing* for each frame.

THE BUGSYS LANGUAGE

BUGSYS is a picture processing and measuring programming language for the analysis of the picture in the computer's memory.[5] The main concept of the system is the use of a collection of programmable pointers, which are visualized as a family of "bugs." A bug can be "initiated," or "placed," and once initiated a bug can be "moved." In addition, a bug can "change" the gray-level value of the spot on which it is located, and it can lay down a so-called "stick" across a thick line in the picture as an aid to locating the middle of the line. In addition, two bugs together can "probe" along the direction of the line between them or along a direction perpendicular to the line between them. These probes can sense an extension of an object or the width of an object, and so forth. "Globs" of objects can be assessed by laying down squares and determining the percentage of the area of the square intersected with the object. The system is composed of many such statements as will be illustrated below.

For each bug initiated by means of the macro PLACE, there is associated a list which gives the

x and y coordinates of the current position of that bug, the actual core location, the spot position (0–11) within this location (there are three bits per spot, or 12 spots per 36-bit IBM 7094 computer word), and the gray-level value of the current position of the bug. For example, if the bug named "ZIPPY" is in the position $x = 6$, $y = 8$, then the list for ZIPPY might contain the following:

Address	Contents	Comment
ZIPPY(1)	6	x
ZIPPY(2)	8	y
ZIPPY(3)	6009	5553 (first location of picture) + 456 (57 words per line times y) + 0 (x divided by 12 spots per word and truncated to an integer)
ZIPPY(4)	6	remainder of x divided by 12
ZIPPY(5)	4	gray-level value or contents of spot

As a bug is moved about the picture by a program, the list for the bug is kept current. In fact, in essence, the list *is* the bug.

We will now proceed to describe some of the statements of the BUGSYS language. The PLACE statement initiates, or sets up, a bug by assigning a name and initial coordinate to it. Thus we have

< PLACE statement > :: = PLACE < bug name >, < x coordinate >, < y coordinate >

where the bug name is a FORTRAN label and the x and y coordinates are either unsigned integers or integer variables.

The BUGS statement allocates five storage locations to each bug named; its form is

< BUGS statement > :: = BUGS (< bug-name list >)

where the bug-name list is a string of bug names separated by commas.

A bug can be moved a specified distance (i.e., number of spots in the picture) in either the x or y direction by the following statement:

< MOVE statement > :: = MOVE < bug name >, < direction >, < distance >

where the direction is given by the literals LEFT, RIGHT, UP, or DOWN, and the distance is an unsigned integer or integer variable. The statement "MOVE ZIPPY,RIGHT,15" moves the bug named ZIPPY to a new location having the same y coordinate but a new x coordinate 15 spots to the right of the present x coordinate. Each time a bug is moved, of course, the list corresponding to the bug name is adjusted for the new values.

Many of the statements involve multiple-way branches, for provision must be made to be sure that the bug will not be moved out of the picture. For instance, in the MOVE statement if the bug would be moved out of the picture, then it is not moved at all, and the next sequential instruction is taken as a next executed instruction. Otherwise (i.e., if the bug will still be in the picture after the move) the bug is moved and the next sequential instruction is *skipped.*

The BUGSYS language also includes a GO.TO statement and a series of TEST statements. For example, the statement

< TEST statement > :: = TEST < bug name >, < relation >, < gray cutoff >

would test the value of the gray level on the spot of the picture at which the bug is located, with respect to the gray cutoff (which is specified by a literal digit from 0 through 7). The relation can be EQUAL or GREATR or LESSN. If the relation is not satisfied, then the next instruction following the TEST is executed. Otherwise (if the gray-level value of the bug location is in *true* relation to the cutoff gray level) the next sequential instruction is skipped and the second following instruction is executed.

The statement

< CHANGE statement > :: = CHANGE < bug name >, < gray value >

can change the value of the bug location spot to the literal specified as the gray value. When using the BUGSYS program, it is usually convenient to change the value of the bug location to a 7. In this way the bugs will leave "footprints" as they move around the picture during the analysis, and these footprints can be utilized as an aid in checking out results. The BUGSYS language also includes provisions for bounding an object. This is the same subprogram as the bounding routine which appears in the FIDACSYS system, except that in BUGSYS it can be utilized in a macro called BOUND.

BUGSYS also includes a specialized macro called CRANK, defined as follows:

< CRANK statement > :: = CRANK < bug name >, < direction >, < gray cutoff >, < neighbor number >

where the "direction" is either the literal CLOCK or the literal CCLOCK, the "gray cutoff" is a literal digit from 0 through 7, and the "neighbor number" is an octal-integer variable. The purpose of CRANK is to help initiate a bounding procedure. As we mentioned above, finding the next boundary point involves knowing the previous boundary point; but in initiating a bounding process no previous boundary point is available. CRANK will produce a previous boundary point for the macro BOUND and will assign this as the value of the "neighbor number." The bounding, and hence the cranking, can be clockwise or counterclockwise around the object, accounting for the term "direction."

The "stick" and "probe" statements enable a bug to investigate its surroundings in various ways. For example, consider the normal-probe statement NPROBE, defined as

<NPROBE statement> :: = NPROBE <bug name>,<bug name>,<bug name>, <distance>,<gray cutoff>

Here the probe will be made perpendicular to the direction between the first two bugs named. The third bug named will hold the result of the probe. This probe consists in effect of moving a bug along a line perpendicular to the line between the first two bugs mentioned until this moving bug comes to the edge of the object. The "distance" is included so that if the bug does not come to the edge of the object before going the distance specified, then an alternative branch may be chosen. It is supposed that the probing bug will start out inside the object. Additional macros are also available in the system.

ILLUSTRATIONS OF RESULTS ACCOMPLISHED

We shall now describe four applications of the FIDAC system: 1) Analysis of Neuron Dendrites (Fig. 3); 2) Analysis of Schlieren Photographs (Fig. 4); 3) Analysis of Rabbit Brain Cells (Fig. 5); and 4) Analysis of Chromosome Karyograms (Figs. 6–11).

Analysis of Neuron Dendrites

In the young kitten's brain, certain cells in the cortex will appear almost entirely within a single microtome section. The following is a discussion of our computer program used for the analysis of such cells, in order that comparisons can be made among these cells in kittens of different development stages. The BUGSYS language was used. In general, bugs will search the picture to find an object, i.e., a neuron cell, and then will move around the boundary of the object, recognizing the dendrites, as distinguished from the cell body itself, and making measurements on dendrite segment lengths from branch to branch. In particular, four components of such a neuron cell must be recognized before the measurements can be made. These are the cell body, the dendrite-to-cell-body junction, the dendritic branching points, and the ends of the dendrite branches. Figure 3a shows a neuron cell as it appears in the computer's memory; Fig. 3b points out a detail of this picture; and Fig. 3c gives a flow chart of the program for the neuron analysis with the BUGSYS language.

Six subprograms are utilized, as shown in Fig. 3c. The first of these subprograms directs the FIDAC instrument to scan the photomicrographic transparency in the gray mode.

Next a bug, called SRCHXY, is directed to search the picture systematically to find the first object. There are two possible outcomes to this process: either a next object is found; or the bug comes to the lower right-hand corner of the frame, which means there are no more objects in the frame. These are the main functions of the subprogram SUNBOG.

If a next object is found, then the program enters the subprogram CBLOCK. In this subprogram the components of bug SRCHXY are used to initiate a new bug, called BUGM, which together with an auxiliary bug, called BUGS, finds the cell body. There are two possible outcomes to this process: first the object may not be a complete neuron or may be some other fragment, in which case there will be no cell body; here transfer of control will be returned to subprogram SUNBOG, which will continue on to look for another object. The second outcome of CBLOOK is that BUGM and BUGS will actually find a cell body, in which case the subprogram DDLOOK is activated.

In the subprogram DDLOOK, the cell-body boundary is traced until a dendritic junction is found. Again there are two possible outcomes to this subprogram: First, a dendrite may not be found, in which case transfer of control is returned to SUNBOG, which looks for another object. The second outcome is that a dendrite is actually found. In this case BUGM will be residing on one side of the dendrite, cell-body junction. BUGM will then

be used to set up another bug, called BUGT, on the other side of this same junction, and subprogram MAPDD will be activated.

In the subprogram MAPDD, BUGM and BUGT move up the dendrite looking for branch points and end points, and by this means these bugs are able to make the desired measurements on the branching dendrite. When they have completely analyzed all the branches, they will return to the dendrite, cell-body junction where BUGM is again utilized. If more dendrites might possible exist, BUGM will continue to look for these other dendrites, in the subroutine DDOOK. On the other hand, if during the mapping of the present dendrite it has been observed that the entire cell has been analyzed, then BUGM will transfer control to SUNBOG and the search for the next object will be reinitiated.

After all of the objects in a frame have been analyzed, the final-result display subprogram, SC4020, will be activated.

Let us now describe some of the mechanisms by means of which the different subprograms are car-

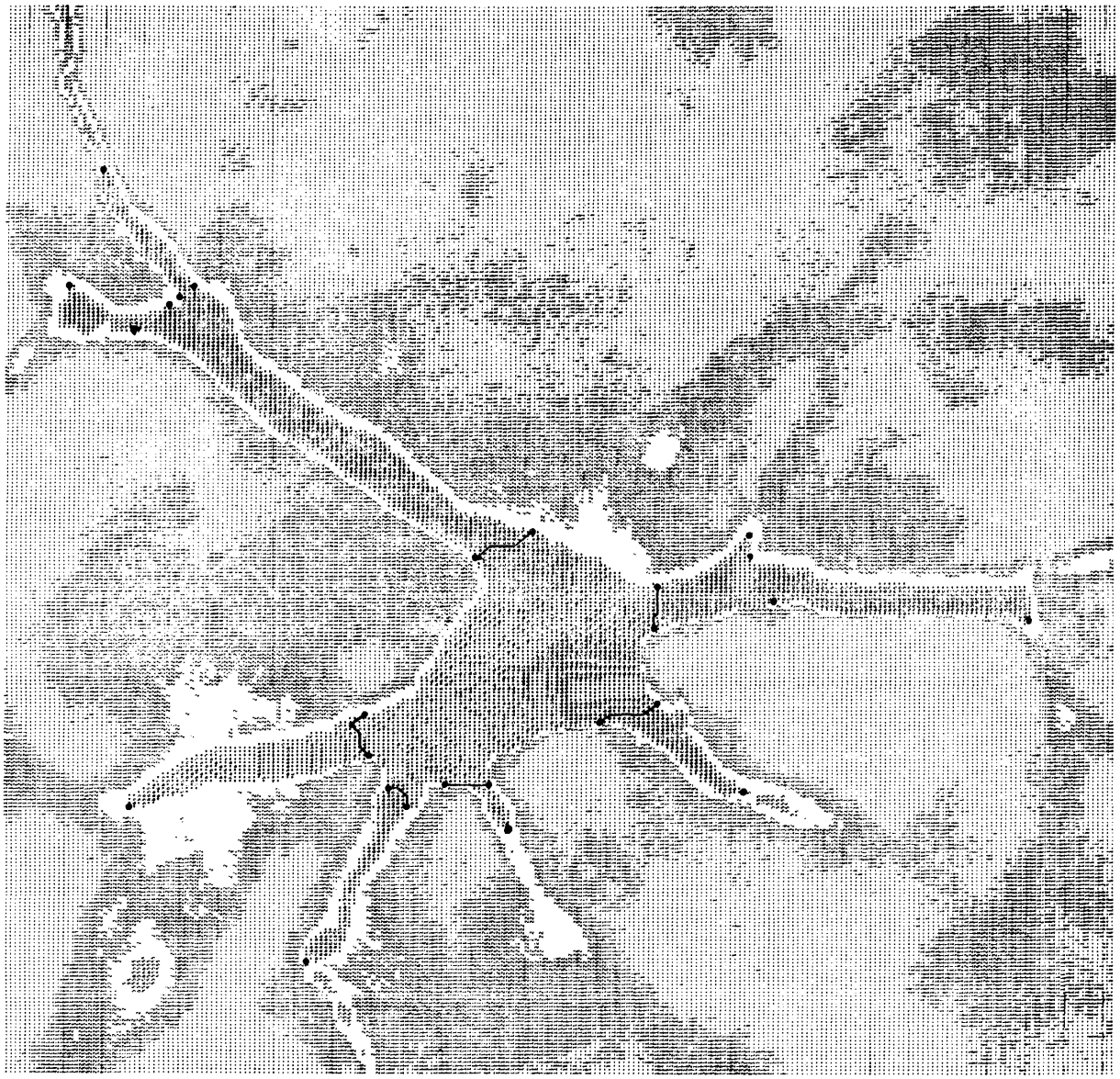


Figure 3a. The large black dots illustrate the computer's determination of the ends of the dendrite arms, of the branch points, and of the cell's body-dendrite junctions. The thin line is drawn at the computer's determination of the actual dendrite, cell-body junction.

ried out. The first step in subprogram CBLOOK is that bug SRCHXY is used to set up BUGM. Now BUGM will move clockwise along the boundary of the object. BUGM will also be used to initiate the subsidiary bug BUGS, which BUGM will drag at a prescribed distance behind it. Together, BUGM and BUGS look for a cell body. The method for finding a cell body is for BUGM to form a *normal probe* perpendicular to their connecting line, directed into the object. The probe is made long enough that it will pass through and beyond the relatively narrow dendritic arm. However, the cell

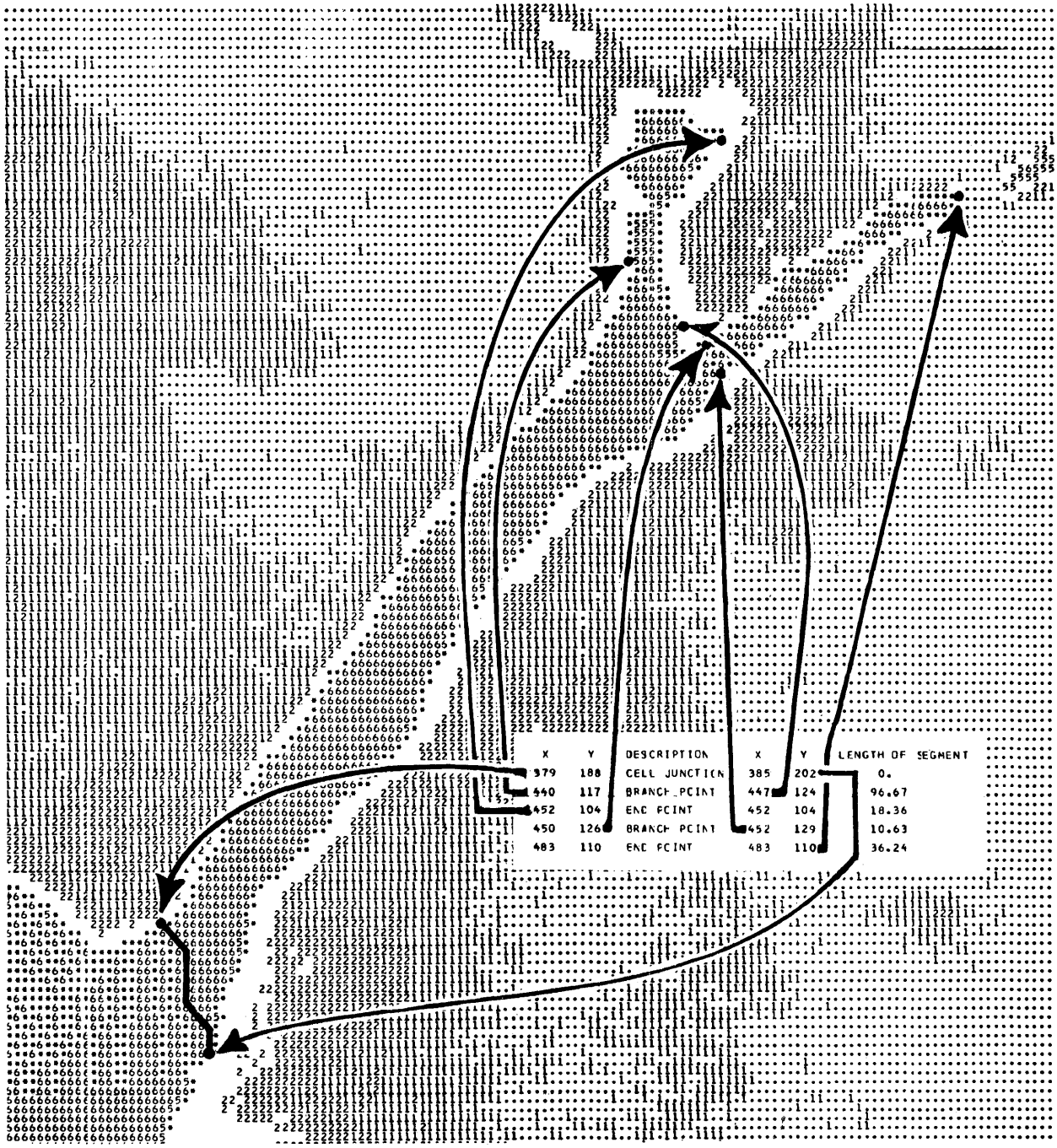


Figure 3b. Details of computer analysis of upper left-hand neuron dendrite showing computered locations of cell junction, branch points, and end points.

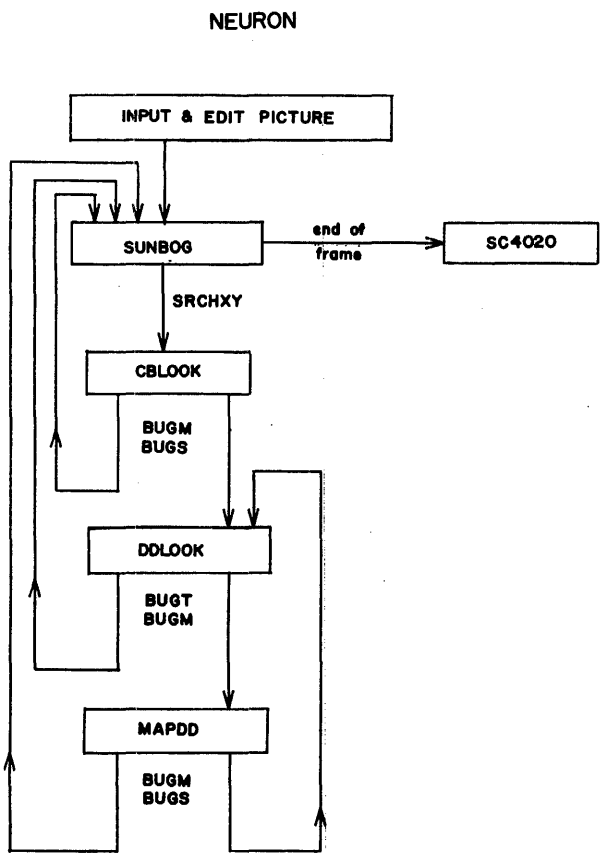


Figure 3c. Flow chart of program for neuron analysis.

body has the distinguishing feature of being thicker than a dendrite. Hence when the end of the probe still remains within the object, it would be reasonable to suppose that a cell body may have been found. A further test is used to confirm the finding of a cell body, since it is possible that a branching point will be wide enough to contain the entire probe. In contrast to such a branching point, the cell body is relatively massive, and hence another criterion is utilized, namely that a reasonably large interior area is found in the region ahead of the probe. Only if such an area is found is the location of a cell body confirmed.

In the subprogram DDLOOK, the boundary of the cell body is searched by BUGM and BUGS. At every point a probe is drawn normal to the line between BUGM and BUGS. When the two bugs begin to climb up a dendrite, the probe goes through the narrow dendrite and comes out the other side. When the width of this narrow portion is less than a predetermined length, then a dendrite is presumed to have been found. When a dendrite is found, BUGT is set up on the other side of the dendrite, cell-body junction, opposite BUGM, and the subprogram MAPDD is initiated. As they bound the cell body looking for dendrites, BUGM and BUGT also observe whether or not either has returned to SRCHXY. If it has, this indicates that the entire object has been bounded, and at the termination of MAPDD the subprogram SUNBOG is entered once more.

The subprogram MAPDD is more intricate than the other programs. Its overall tactic is to have BUGM and BUGT climb out on the dendrite. At the dendrite, cell-body junction, BUGM and BUGT are opposite each other. During the climb they are programmed to remain approximately opposite each other on the dendrite. When a branch point in the dendrite is found, BUGT will be going out on one side of one of the branches while the BUGM will be going out on the other side of the other branch. Thus BUGM and BUGT begin to move apart. The distance between them is observed, and if it exceeds a certain value a branch point is indicated. Confirmation of this branch point is assured by drawing a line, or tangential probe, between BUGM and BUGT. Only when this line leaves and reenters the cell area is the attainment of the branch confirmed. In addition, since this line completely crosses each of the two branches, new bugs can be set up at its intersections with the inner sides of the branch.

Of course there may be many branches to a dendrite. The procedure is to analyze in each case the most counterclockwise branch first. The other branch is stored in a so-called *push-down list* for later analysis. The end of a branch is found when BUGM and BUGT meet at the same point. Then the most recent branch in the push-down list is *popped up* and analyzed. A little reflection will show that all of the branches can be systematically analyzed by this means.

Analysis of Schlieren Photographs

At present, data from ultracentrifugal experiments is usually presented in the form of Schlieren, or interference, images on film or on glass-backed spectroscopic plates. These images are normally measured manually on a microscope having a micrometer stage; then the measurements are processed by certain algorithms to obtain sedimentation coefficients, weight-average molecular weights, or whatever molecular parameters the investigator may be studying. Usually as much time is spent measuring the plates, and as much time again in

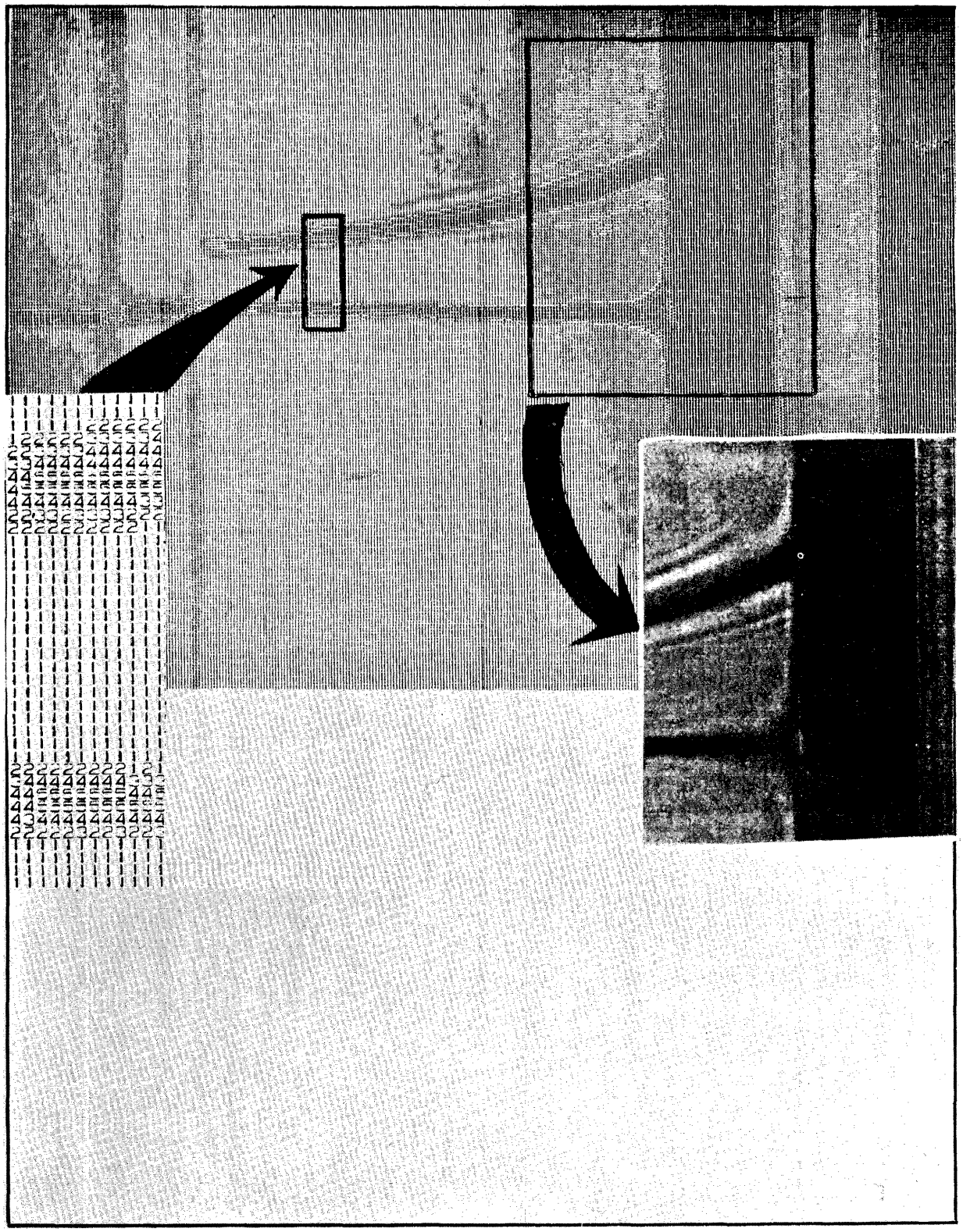


Figure 4a. Picture of a Schlieren photograph as recorded in the computer's memory. In the insert we illustrate an enlarged portion of the figure showing the density levels (where the unit is the zero level). We have also shown an insert of a portion of the original Schlieren photograph.

computation, as was required to perform the experiments. The automatic analysis of such ultracentrifugal Schlieren photographs consists in first reading each picture into the computer by means of the FIDAC instrument (see Fig. 4a), and then utilizing the BUGSYS language to make measurements on the picture. As illustrated by the dots, or "footprints," in Fig. 4b, a bug is first placed to the right, beyond the two lines. It is then moved to the left, searching for the rightmost line, or *function curve*. When this line is found, another bug continues the motion to the left, looking for the second line, or *base axis*. Now we have one bug placed on each line. The bugs are then moved up along their respective lines until the right-hand bug finds the beginning of the function curve. The measurement to be made in this case is the horizontal distance between the two bugs, i.e., Δy. The bugs now move down, one point at a time, making approximately 115 measurements of Δy on the curve. The measurements are made from the "middle" of each line, since the line is really "thick." The STICK statement is used to keep the bug in the middle of the line. After all the measurements are made, the computer programs to determine molecular parameters are executed, to produce the finally desired results. On an IBM 7094, this entire process takes less than one second per Schlieren photograph, with the FIDAC device.

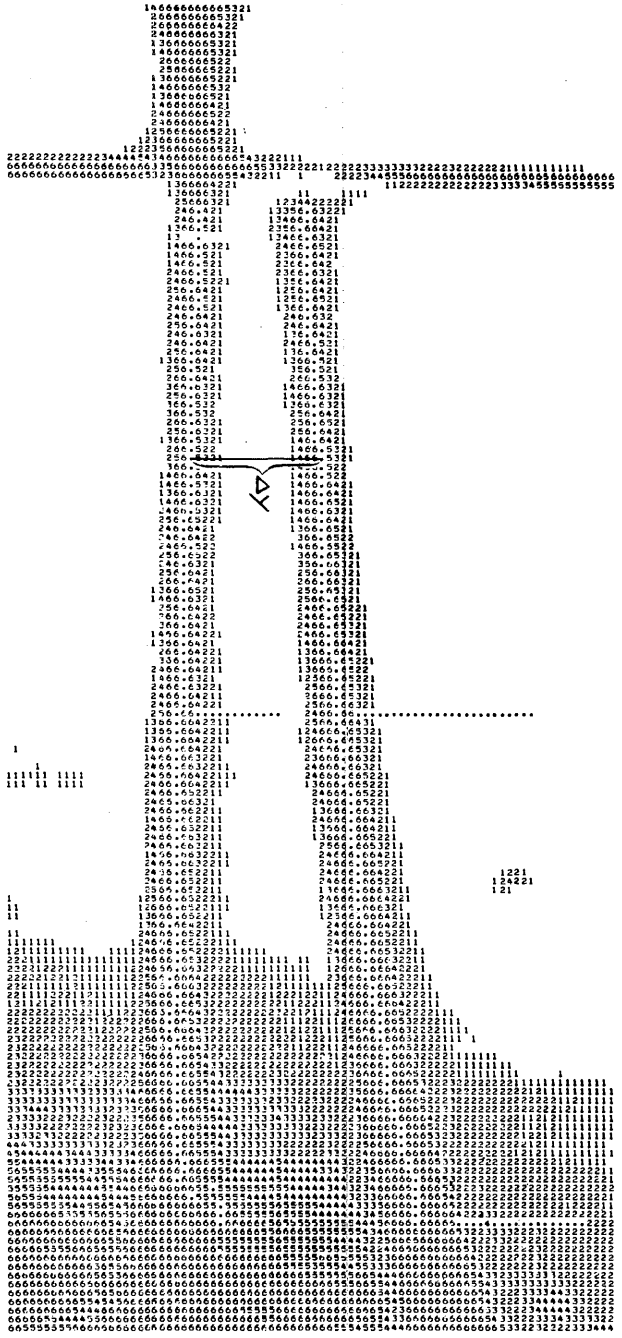


Figure 4b. Computer analysis of Schlieren photographs showing "footprints" of the paths of the bugs.

Analysis of Neuron Cell Bodies

The purpose of this study was to investigate possible morphological changes due to aging in cell populations of neurons from the caudate nucleus of the rabbit. The morphological characteristics measured were 1) the area, 2) the maximum diameter, 3) the maximum perpendicular radii, and 4) eccentricity of the perpendicular radii (see Fig. 5a). These measurements were made for both the nucleus and cytoplasm of each of about 6000 cells, utilizing the FIDAC scanning instrument on an IBM 7094 computer. About 2.1 seconds per cell, or 3½ hours altogether, were required to process the 6000 cells, including the scanning and computer-measurement times.

A problem arose from the fact that in looking through a microscope at a three-dimensional cell only a two-dimensional section is seen at the plane of focus. Therefore the largest projected area of the cell can only be obtained from the examination of a composite of sections seen at many levels of focus. To obtain such a composite, the image from the microscope was projected into a ground-glass plate, and as the plane of focus of the microscope was moved up and down the largest projected area was traced on paper. In this way, drawings were made of each cell in the study, with the cytoplasm in black and the nucleus clear. These drawings were

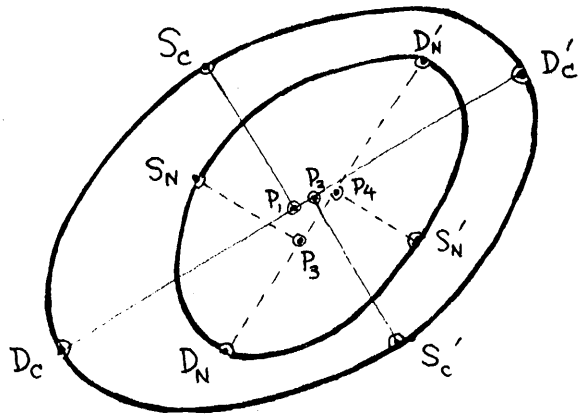


Figure 5a. Diagrammatic picture of a cell indicating the measurements to be made. For the cytoplasm the measurements to be made are the maximum diameter $D_C D'_C$ and the semiminor axes $S_C P_1$ and $S'_C P_2$, as well as the eccentricity $P_1 P_2$ and the area. For the nucleus the measurements to be made are the maximum diameter $D_N D'_N$, the semiminor radii $S_N P_3$ and $S'_N P_4$ and the eccentricity $P_3 P_4$ and the area.

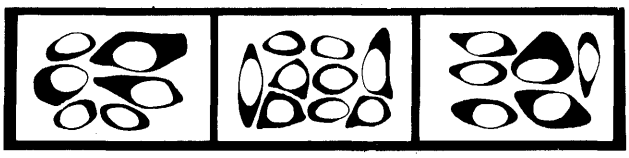


Figure 5b. Actual 35mm frames showing cells.

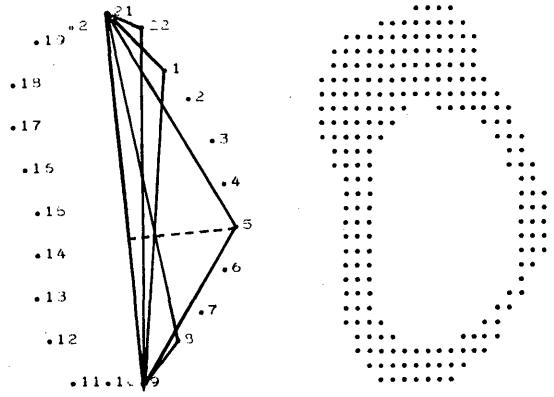


Figure 5c. On the right, a partial-print of an object. On the left, illustration of the selected boundary points on the object and examples of the triangles by means of which the minor semidiameter is determined. This object is shown for illustrative purposes only. In general the brain cells analyzed were much larger, and about 200 boundary points were obtained on the cytoplasm.

then photographed on 35mm film for input through the FIDAC (see Fig. 5b).

For this problem the FIDACSYS programming system was utilized. In the first step, the FIDACSYS system directed the FIDAC to scan a frame and read the picture into the computer's memory. After a frame had been read into the computer, the internal bug scan was initiated in order to locate the first object or cell. When a cell was located, its boundary segments were determined (see Fig. 5c). While tracing out the boundary, the bug also replaces the boundary points with the value "7." In addition, during this process the maximum and minimum x and y extensions of the cell under consideration are also determined.

For the computation of the maximum diameter, the distances between all pairs of boundary points are determined; the points corresponding to the largest distance are the end points of the maximum diameter. For the computation of the maximum perpendicular radii, the areas of all possible triangles constructed on the maximum diameter with an intermediate boundary point as vertex are compared. The largest area on either side will give the two largest perpendicular radii. A simple calculation gives the point of intersection of each maximum perpendicular radius with the maximum diameter, and the eccentricity is thereby determined.

Next an erasing process is initiated; the object is erased from the top down until the nucleus of the cell, which is a clear area, is found. The boundary of the nucleus is then traced by the bug, and a list of boundary points is made. From this list of points, the maximum diameter, perpendicular radii, and eccentricity are again computed.

Analysis of Chromosome Karyograms

As a final example we shall consider the automatic analysis of chromosomes by the FIDACSYS programming system. Recently there has been much active interest in the analysis of chromosomes in the metaphase stage of mitosis, when they appear as structures split longitudinally into rod-shaped *chromatids*, lying side by side and held to one another by a constricted area called the centromere. Certain abnormalities in the number and structure of chromosomes are particularly evident at this stage and can be related to clinical conditions in animals and in men. For example, in man, mongolism and the Klinefelter and Turner syndromes have been correlated with chromosomal aberrations.

The study of chromosomes by manual methods, however, requires a great deal of time—enlarged prints must be made from photomicrographs, and

each chromosome must then be cut out from the print so that it can be aligned with the others for classification into the so-called chromosome karyotype. With the FIDAC system, the time required for analyzing and classifying each chromosome can be radically reduced, to about half a second per chromosome or about 20 seconds for the full complement of human chromosomes. Here we use the computer for investigating large numbers of cells with respect to total chromosome complement counts, to quantitative measurements of individual chromosome arm-length ratios, densities, areas, and other morphological characteristics, and so forth.

By processing large numbers of chromosome sets and statistically analyzing the data, it is possible to give very accurate descriptions of the standard

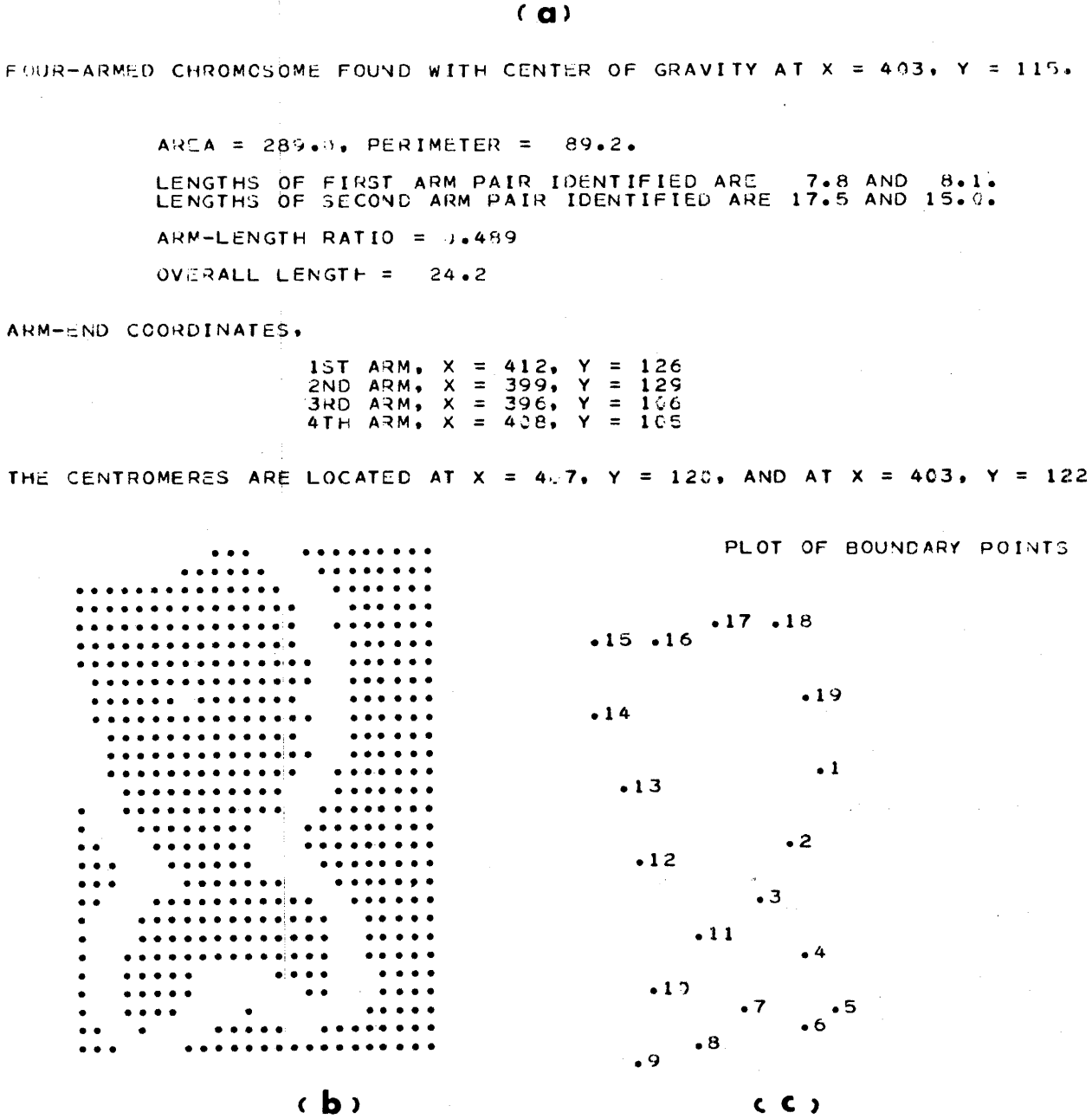


Figure 6. (a) Results of analysis on a chromosome. (b) Subprintout of chromosome being analyzed. (c) Centers of boundary segments.

chromosome complement and of individual chromosome variability for particular species. This statistical technique may be the only way to uncover small variations, which may prove important in relating chromosome karyotypes to diseases.

Figures 6 through 11 illustrate the capabilities of the FIDACSYS chromosome-analysis computer program. After the picture has been placed in the computer's memory, an internal programmed scan is made in the memory for each of the chromosomes in the picture. When a chromosome is found, it is analyzed; the results obtained are as shown, for example, in Fig. 6a. The coordinates of the center of gravity of the chromosome are determined, its area and perimeter are measured, its centromere and arms are identified and the lengths of the arms of each arm pair are measured, the arm-length ratio is computed, and the overall length of the chromosome is determined. Of course, the actual coordinates of the ends of the arms and of the centromeres had to be determined in order that the arm lengths could be properly measured. These coordinates are also given for the chromosome, as illustrated in Fig. 6a.

In Fig. 6b we show a subprintout of the particular chromosome being analyzed. When a chromosome is located during the internal scan in the computer's memory, the boundary of that chromosome is traced. This boundary is then characterized by a sequence of boundary segments. Figure 6c illustrates the actual locations of the centers of the boundary segments that were chosen to characterize the chromosome of the illustration; for this chromosome, there were 19 such segment centers in all.

The next step in the analysis of a chromosome is to characterize the curvatures of the boundary segments. These curvatures, called the *basic parts* of the chromosome, are illustrated in Fig. 7a, in which we show the curvature code corresponding to each boundary segment. The code letters at the end of the alphabet, namely U, V, W, X, Y, and Z, represent concave curvatures in increasing order, whereas the code letters A, B, C, D, E, and F, represent convex curvatures in increasing order. The code letter O represents an essentially straight segment. Next, from these basic parts, the syntax-directed pattern-recognition technique is carried out to build up the so-called *derived parts* of the chromosomes.

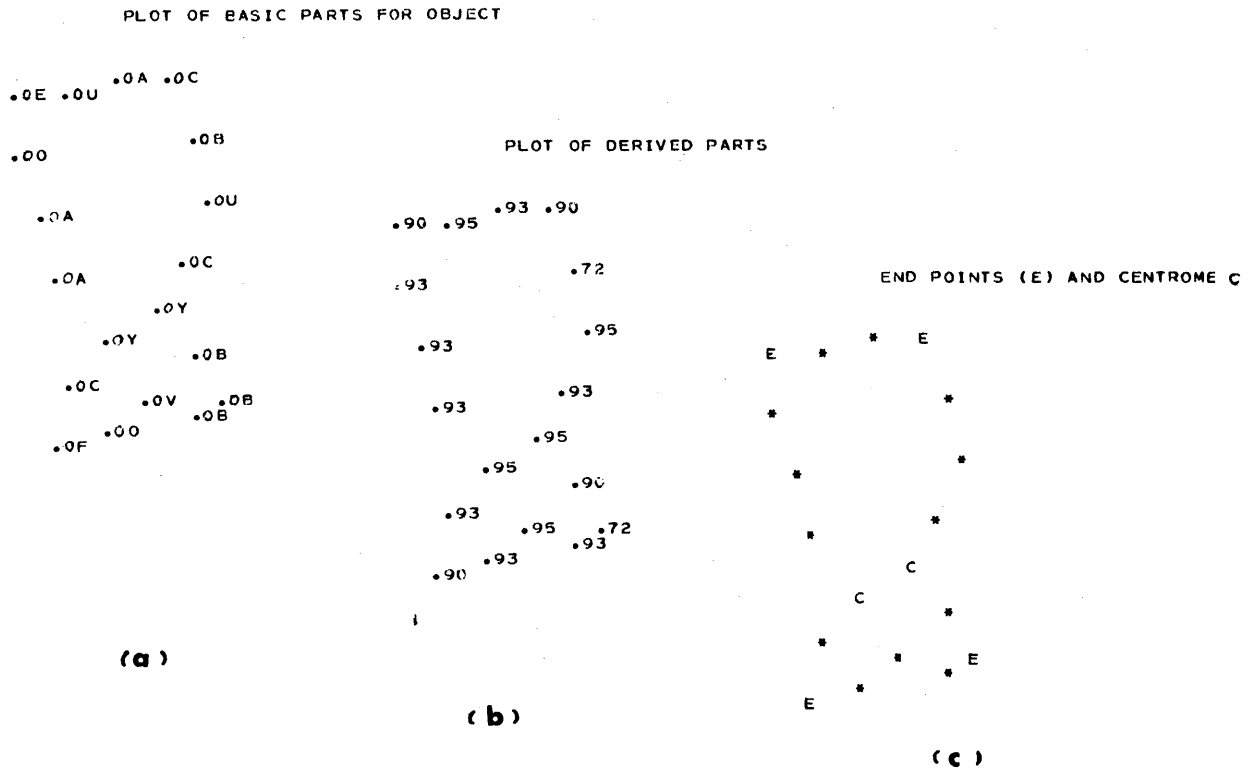


Figure 7. (a) Curvature codes corresponding to each boundary point. (b) Codes for derived parts. (c) Final results showing end points "E," and centromere location "C–C."

CHROMOSOME ANALYSIS SUMMARY

FR NO	CH NO	T	CENTER GRAVITY	PERI METER	OVRALL LENGTH	2 LONG LENGTHS	2 SHORT LENGTHS	LENGTH RATIO	AREA	LONG AREA	SHORT AREA	AREA RATIO
1	1	C	328, 37	100.3	34.7	26.2,26.5	7.9, 8.6	.761	499.	390.	105.	.788
1	2	A	321, 70	183.6	73.3	38.5,37.9	34.9,35.2	.522	1013.	512.	489.	.512
1	3	C	245, 69	89.5	28.8	19.2,19.5	8.6,10.3	.673	434.	303.	128.	.703
1	4	D	416, 87	59.7	16.9	8.4,10.9	8.1,10.3	.511	219.	110.	110.	.501
1	5	C	259, 90	81.2	29.0	24.9,24.7	3.8, 4.5	.856	378.	357.	35.	.910
1	6	C	288,113	68.5	24.4	19.5,19.0	5.3, 5.0	.788	295.	235.	56.	.808
1	7	D	338,134	53.5	16.1	8.6, 8.6	7.3, 7.8	.533	171.	93.	76.	.549
1	8	B	220,156	112.3	40.5	24.7,23.8	16.4,16.1	.598	572.	324.	242.	.572
1	9	D	248,146	64.5	20.1	12.0,10.8	8.9, 8.5	.568	240.	143.	100.	.589
1	10	C	441,151	80.7	28.2	21.0,19.4	8.2, 7.8	.716	397.	293.	97.	.750
1	11	A	427,183	199.7	78.7	42.3,40.9	37.5,36.7	.529	1114.	591.	499.	.542
1	12	D	208,184	49.5	15.6	8.7, 7.5	7.6, 7.4	.520	153.	79.	72.	.521
1	13	B	304,190	117.6	41.0	25.2,21.5	20.0,15.3	.570	531.	295.	237.	.555
1	14	B	356,208	99.2	32.1	18.5,18.7	12.7,14.2	.580	451.	246.	205.	.545
1	15	D	326,215	63.6	19.8	11.9,11.0	9.1, 7.5	.580	219.	128.	87.	.596
1	16	D	265,264	51.7	16.5	8.7, 9.2	7.9, 7.1	.544	154.	81.	69.	.540
1	17	B	214,275	90.9	30.5	19.1,18.7	12.0,11.2	.619	382.	253.	125.	.669
1	18	C	442,274	76.5	24.7	21.0,18.5	5.3, 4.6	.799	351.	277.	64.	.813
1	19	A	250,307	161.0	59.7	32.4,32.0	28.0,26.9	.540	895.	450.	426.	.514
1	20	C	285,305	71.6	23.9	23.2,21.6	1.5, 1.5	.937	301.	297.	7.	.977
1	21	C	366,318	90.9	30.2	22.0,22.4	8.9, 7.1	.735	382.	310.	69.	.818
1	22	A	303,356	203.2	83.0	45.4,46.0	36.5,38.2	.550	1139.	592.	550.	.518
1	23	B	356,394	120.5	39.3	23.5,19.9	20.5,14.7	.552	596.	318.	262.	.548

Figure 8. Printout of final results of completely automatic analysis of full complement of chromosomes of the Chinese-hamster tissue-culture cell of Fig. 9.

This procedure results in the recognition of the ends of the arms and the centromere position. Figure 7b illustrates the codes developed for the derived parts.

Figure 7c illustrates the final results of the recognition method for our illustrative chromosome. Here the letter E has been placed at the points which represent the ends of the arms of the chromosome, and the letter C has been placed at the points that represent the location of its centromere. The measurements of the arm lengths are made from the centromere to the end of the corresponding arm. The area and perimeter of the chromosome have previously been determined by the computer program.

Figure 8 shows a printout from the computer of the completely automatic analysis of the full complement of chromosomes from a Chinese-hamster tissue-culture cell. Note that the computer has automatically 1) counted the chromosomes, and for each chromosome has automatically 2) found the center, 3) evaluated the perimeter, 4) measured the overall length, 5) measured the lengths of the two long arms, 6) measured the lengths of the two short arms, 7) calculated the (arm) length ratio,* 8) evaluated the overall area, 9) measured the area of the long arms, 10) measured the area of the short arms, and 11) calculated the ratio of the arm areas.† Figure 9a shows the photomicrograph of the cell of Fig. 8. Figure 9b was plotted by the computer only from the results of its analysis. This plot shows where the computer 1) located and 2) counted each chromosome (see small numbers), and for each chromosome shows how the computer 3) traced the perimeter (see chromosome outlines), 4) identified the arm end points (see x's), and 5) determined the centromere (see line across chromosome).

In Fig. 10 the computer has plotted the results of the automatic classification of the chromosomes of the cell of Figs. 8 and 9, giving the complete karyogram. The x marks the centromere, and the lines diagramatically illustrate the actual arm lengths; the number below each schematic chromosome relates it to the corresponding chromosome in the Table of Fig. 8 and in the plot of Fig. 9b. In Fig. 11 the computer has plotted a scattergram of the short arm lengths vs the long arm lengths, where the small numbers again relate to the chromosome numbers of Figs. 8 and 9.

*The arm-length ratio is computed as the average of the long arm lengths divided by the sum of the averages of the short and long arm lengths.

†The ratio of the arm areas is computed as the area of the long arms divided by the total area of the chromosome.

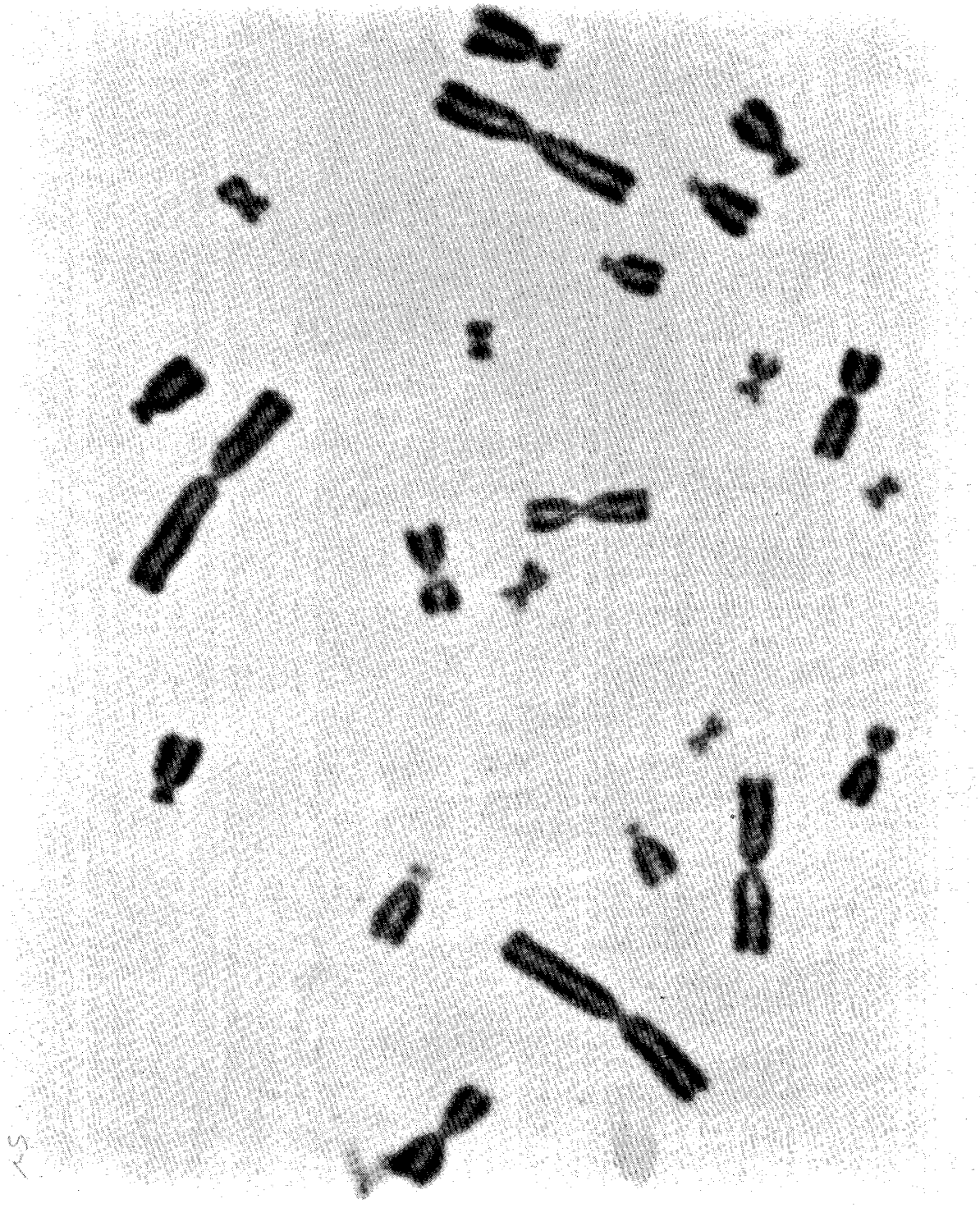


Figure 9a. Photomicrograph of cell of Fig. 8.

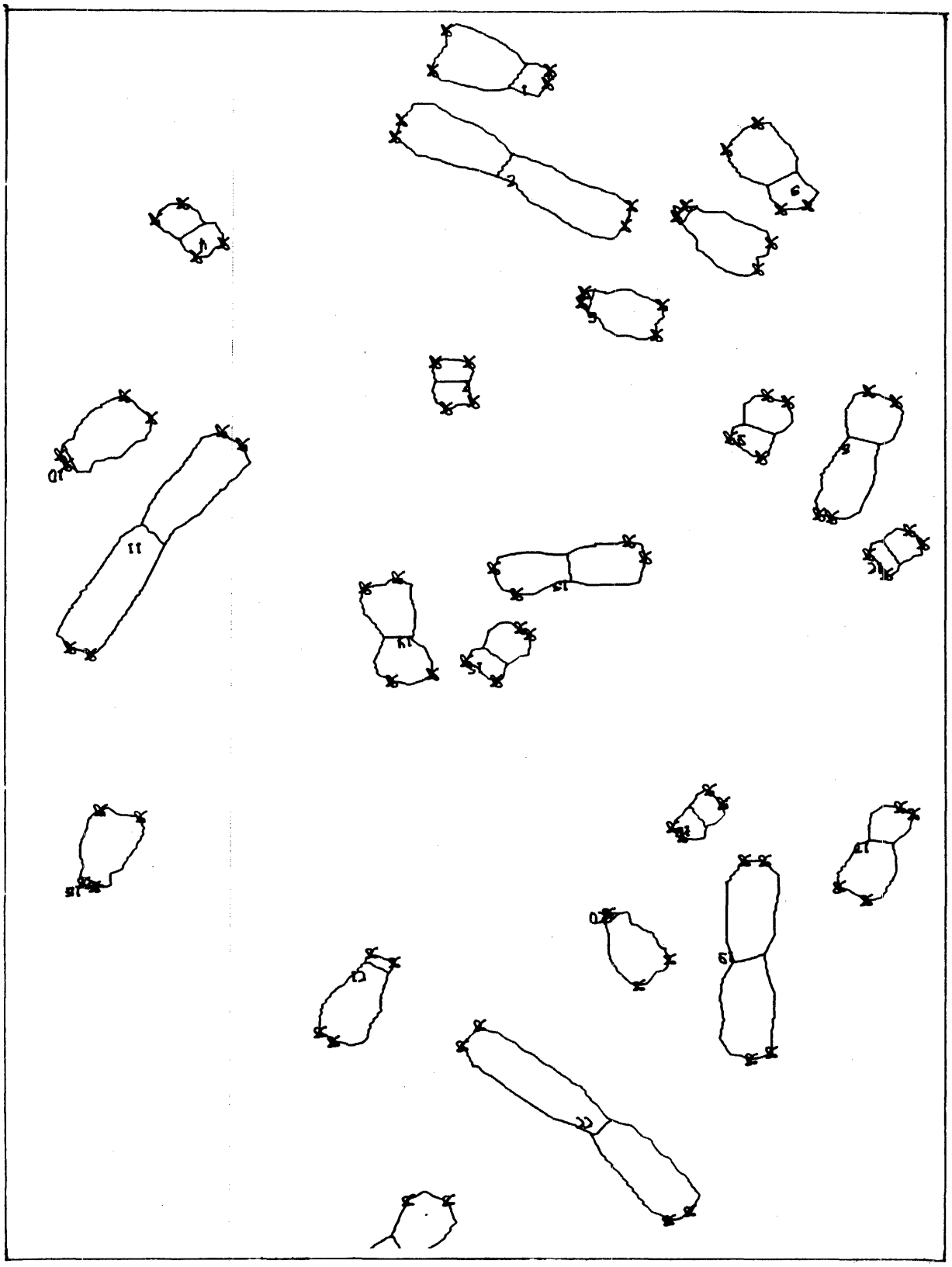


Figure 9b. Plot of computer analysis of cell of Fig. 9a.

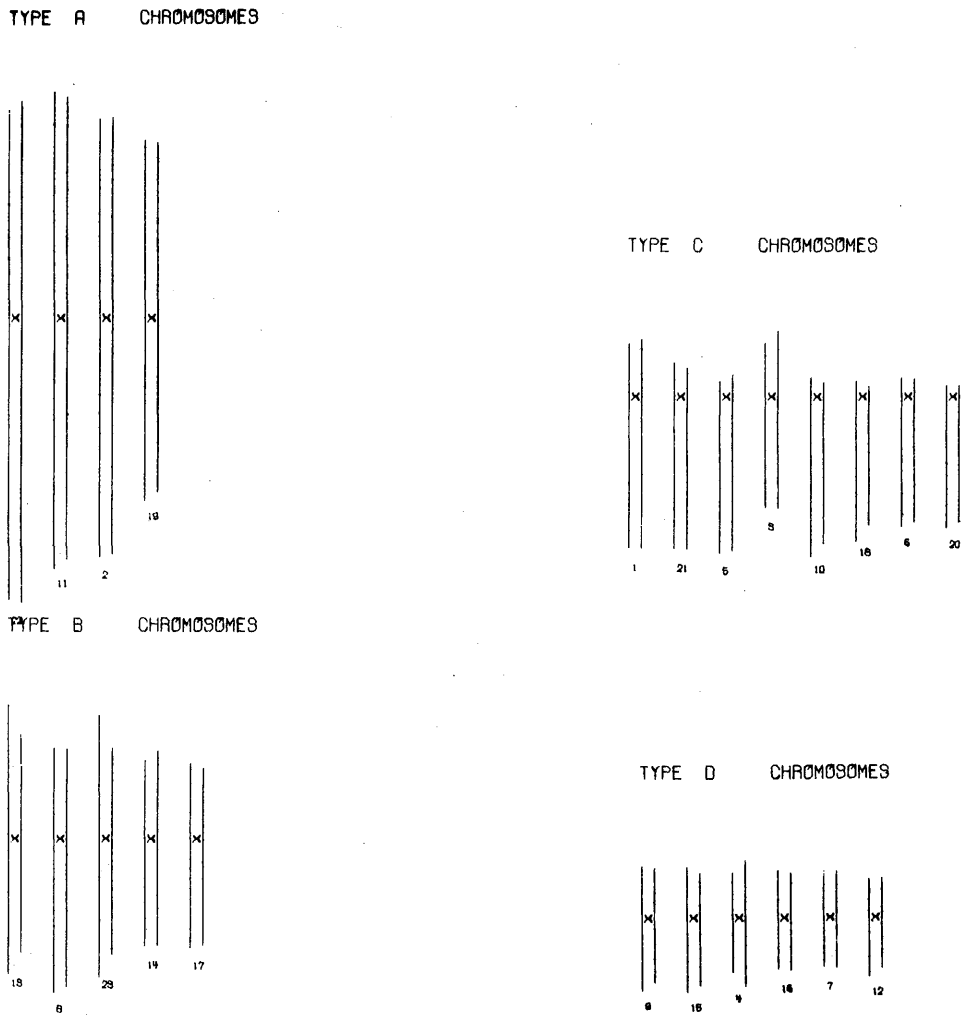


Figure 10. Karyogram plotted by computer.

SUMMARY

In this paper we have presented an introduction to the instrumentation and programming techniques utilized by the authors in biomedical pattern recognition and processing. We have illustrated the techniques by briefly describing four applications already accomplished. Such techniques promise to open up entirely new fields in biomedical research.

ACKNOWLEDGMENTS

We acknowledge the cooperation in this work of Prof. Charles Noback, Dr. Richard Moore, Dr. Alden Dudley, and Prof. Frank Ruddle.

REFERENCES

1. R. S. Ledley, *Use of Computers in Biology and Medicine*, McGraw-Hill Book Co., New York, 1965.

2. Ledley et al, "FIDAC: Film Input to Digital Automatic Computer and Associated Syntax-Directed Pattern-Recognition Programming System," in J. T. Tippet et al (Eds.), *Optical and Electro-Optical Infor. Proc.*, MIT Press, Cambridge, Mass., 1965, Chap. 33, preprint.

3. R. S. Ledley, *Programming and Utilizing Digital Computers*, McGraw-Hill Book Co., New York, 1962, Chap. 8.

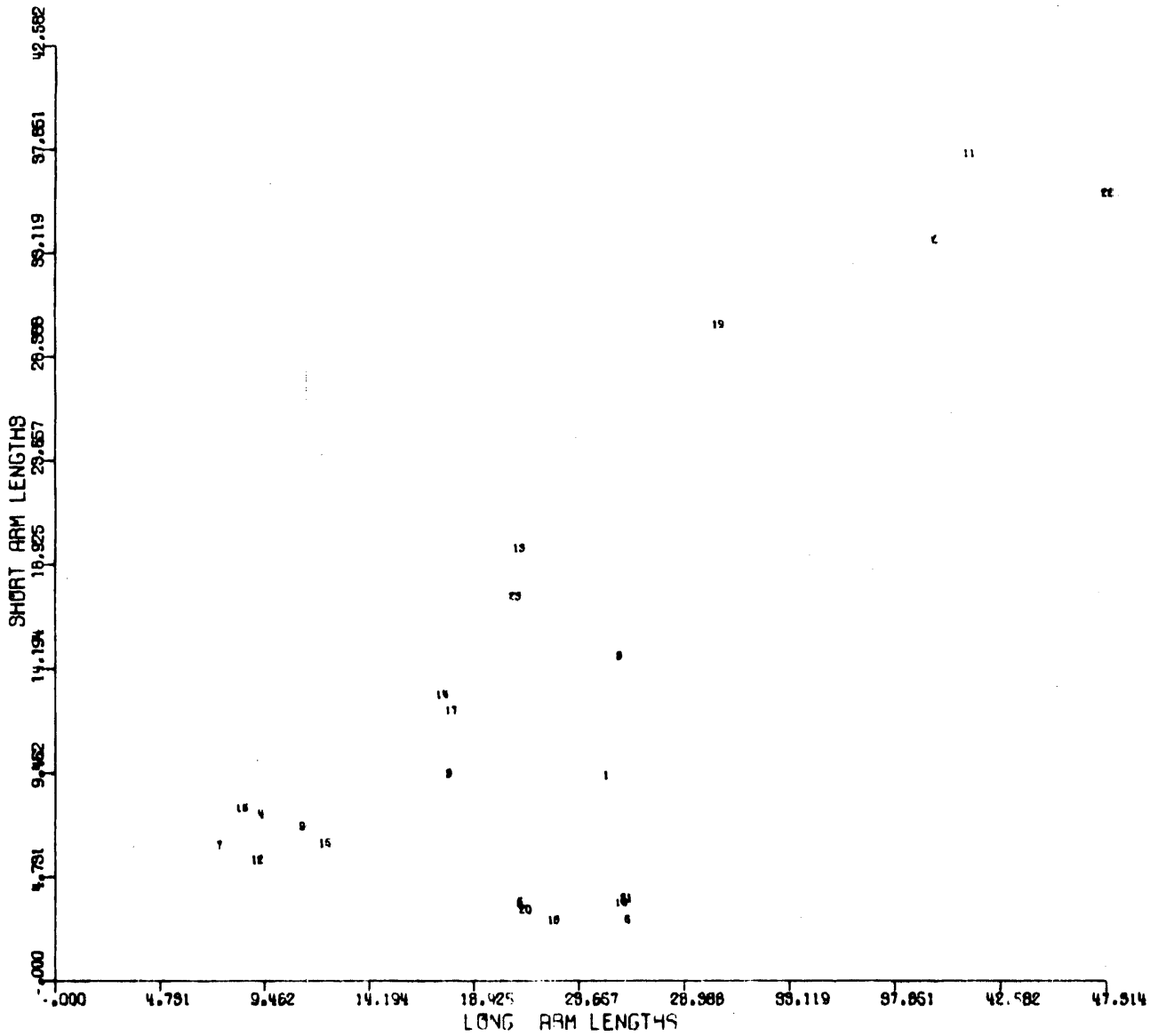


Figure 11. Scattergram plotted by computer of short arm lengths vs long arm lengths.

4. Ledley and J. B. Wilson, "Automatic-Programming-Language Translation Through Syntactical Analysis," *Commun. Assoc. Computing Machinery*, vol. 5, no. 3, pp. 145–155 (Mar. 1962).

5. Ledley, J. D. Jacobsen and M. Belson, "BUGSYS—A Programming Language Not for Debugging," ibid, vol. 9, no. 2 (Feb. 1966). See also K. C. Knowlton, "A Computer Technique for Producing Animated Movies," *AFIPS Conf. Proc.*, vol. 25, pp. 67–87 (Spring 1964).

A CHESS MATING COMBINATIONS PROGRAM*

George W. Baylor and Herbert A. Simon
Carnegie Institute of Technology
Pittsburgh, Pennsylvania

1. INTRODUCTION

The program reported here is not a complete chess player; it does not play games. Rather, it is a chess analyst limited to searching for checkmating combinations in positions containing tactical possibilities. A combination in chess is a series of forcing moves with sacrifice that ends with an objective advantage for the active side.[1] A checkmating combination, then, is a combination in which that objective advantage is checkmate.† Thus the program described here—dubbed MATER—given a position, proceeds by generating that class of forcing moves that put the enemy King in check or threaten mate in one move, and then by analyzing first those moves that appear most promising.

The organization of this paper centers around MATER's ability to analyze chess positions. After a brief look at the program's history in Section 2, the overall organization of the program is presented in Section 3, an organization which is designed to allow flexible movements in an analysis tree of possibilities. Then in Section 4 the "top level" comes under consideration; MATER's heuristics of analysis—the search rules and priorities, the search evaluators—that enable it to find a mate in the maze of possibilities. In Section 5 data on the performance of the program is presented. Finally, the programming language representation of the chessboard and chess pieces and the basic chess capabilities this affords are reported in the Appendix (they are reported in yet finer detail in Baylor[2]).

2. HISTORY OF THE PROGRAM

MATER has led a checkered life. The original mating combinations program, as conceived by Simon and Simon[3] was a hand simulation setting forth a strategy of search. According to hand simulations the program discovered mating combinations in 52 of the 129 positions collected in Fine's[4] chapter on the mating attack. But a hand simulation is not a rigorous model and, as such, is itself sometimes prone to the imprecisions and ambiguities of many verbal theories. Indeed, Myron and May[5] pointed out two such ambiguities in the specifications laid down by Simon and Simon[3].

Newell and Prasad[6] coded an IPL-V program which set up a chessboard, recorded positions, made moves, tested their legality, and performed a few other functions (see Appendix). This they overlaid with the beginnings of a mating program.

*This investigation was supported in part by Research Grant MD-07722-01 from the National Institutes of Health, and by the System Development Corporation while the author was a summer associate there.

†Sometimes the defender is able to avert the checkmate by incurring a heavy loss in material (pieces and/or Pawns). If the attacker's gain in material is indeed an "objective advantage"—the defender being left with no compensatory attacking changes—then such combinations would generally be called mating combinations, even though not ending in mate. The current version of the program confines itself to mating combinations in the narrow sense—those from which there is no escape. Inclusion of the broader class is an obvious extension.

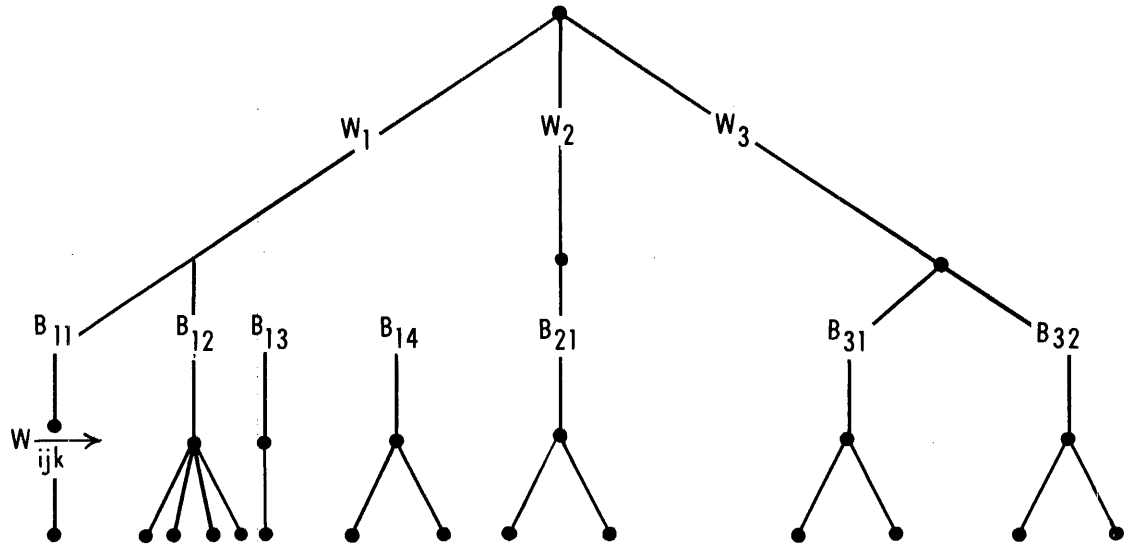


Figure 1. The Analysis Tree
(The W's are White moves, the B's, replies.)

The first version of a mating program described here—MATER I—is the work of G. Baylord,[2] H. Simon and P. Simon,[3] using the Newell-Prasad[6] chessboard. The program was revised during 1964 by Baylor into a second version, MATER II.

3. ORGANIZATION OF THE ANALYSIS TREE

The Problem

As stated in Section 1 the mating program analyzes chess positions. An analysis of a position —as the term is used here*—consists of the set of moves and evaluations made in the course of resolving the choice-of-move problem. Taken together, moves and positions make up a tree of possibilities in which moves operate on positions to produce new positions (see Fig. 1) and on which evaluations of positions and of moves in achieving desired positions can be hung as desired.

The dots or nodes in Fig. 1 denote positions (static states) and the lines between dots denote moves (operators) that transform one position into another.†

How should the analysis of a position be conducted? Figure 2 presents one simple scheme. Would this scheme be workable if it were made operational? For example, let:

1. "X" be defined as checkmate and the program be given the capability of generating moves in its service;
2. the criteria for deciding among moves be specified by certain rules of selection;
3. the program be given the capability of making moves and updating board positions;
4. a test be provided so that the program "knew" if it had achieved "X"; and

5–8. the corresponding provisions be made for "Y", defined as escaping check, and for choosing among replies.

The answer is no, not quite. The scheme lacks a means for recovering from false starts, for retracing its steps when it runs into blind alleys. Indeed, what is lacking can be seen by considering the difference between actually playing a game of chess and analyzing a chess position: the course of analysis is fickle and reversible, whereas in an actual game a move once made cannot be unmade. In other words, the scheme outlined above needs provisions for unmaking moves and for abandoning seemingly unpromising positions as much as it needs the capability of making moves and pursuing promising positions. Ideally, one should like to be able to

*Chess players would probably prefer to define "analysis" as the finished product rather than the process of search, laying stress on the "right" moves and continuations rather than emphasizing how these were arrived at.

†Simon and Newell[7] have often drawn this difference equation analogy to the problem-solving process: given an initial state description and a desired state, the problem is to find a process description that operates on the initial state to produce the desired one. In discussion of the Logic Theorist,[8] for example, the logic expressions correspond to the static states, and the rules of inference, to the operators.

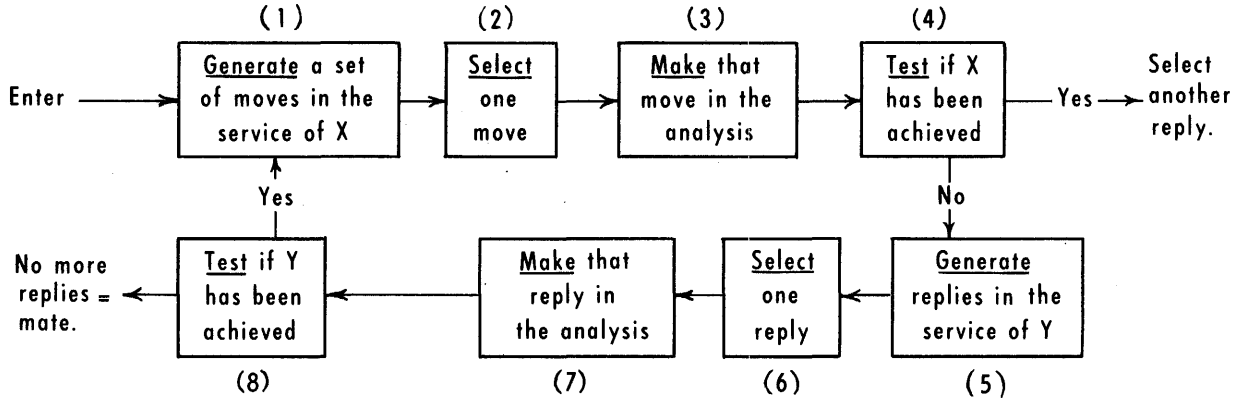


Figure 2. A simple recursive mating scheme.

enter and reenter the move tree at any node (position) at any time and from there to proceed down any branch, old or new. Providing such capabilities for reinstating the right position at the right time is probably the central problem of organization at this level of the program, while making operational and making sense out of steps 1–8 above is probably the central problem at the next higher conceptual level. This section reports on the former problem: implementation of a flexible move tree. Section 4 is devoted to the latter: defining the problem and the heuristics of search.

Building the Tree

The notion of analysis as a tree search is misleading to the extent that it implies that each step consists solely of selecting a move from the many available alternatives. Actually, the process is more one of generating moves as one goes along, of building one's own tree. This is the very distinction Maier[9] (p. 218) has drawn between decision making under conditions of uncertainty* and problem solving: "Decision making implies a given number of alternatives, whereas in problem solving the alternatives must be created. Thus, problem solving involves both choice behavior and the finding or creating of alternatives."

In every chess position, of course, the rules of the game place an upper limit on the number of possibilities that can be created; de Groot[11] found that, averaged across the course of a game, the mean number of legal move possibilities lies somewhere between 30 and 35. This is a full-grown tree; the one the searcher actually builds is much smaller: on the average, four or five branches at the top node and smaller thereafter.

The first question addressed in this section is the technical one: How does one build a tree? How are limbs and nodes—moves and positions, respectively—structured into a tree of possibilities? (See the Appendix for the details of construction of moves and positions.) Second, and pursuing the metaphor, think of the chess player climbing the tree as he builds it. Crawling along a branch in one direction corresponds to making a move in the current position (node) while traversing it in the opposite direction corresponds to unmaking a move and restoring the previous position (node). This ability to back up the tree is what enables a player to abandon unpromising lines of investigation and start afresh. In starting afresh, moreover, the player may either reinvestigate branches he has previously built or build new ones.

Third, as the player builds and climbs he also accrues and retains information. The information garnered en route and the use to which it is put are in large part what *Denk*psychologists have called the development of the problem. That is, the searcher's conception of the problem at any one time consists of the information he has about the problem, how he has evaluated this information, and even how it has shaped his definition of what the problem is (cf Duncker[12] and de Groot[11]). Provisions for gathering information are considered in both this section and the next; the use to which information is put and the matter of problem development are more properly treated in the subsequent sections.

Most of the organizational problems are solved via the description lists of moves. For convenience of reference the entire list of possible attributes a move can take on is set forth in Table 1.

*According to Luce and Raiffa[10] decision making under uncertainty is the condition in which the outcomes of the various known alternatives are unknown.

With respect to the first question—provisions for holding the tree together—a signal cell and attributes A46 and A47 do the job. The signal cell contains the name of the most recent move made on the board—the contemporary—while attributes A46 and A47 are its ancestor and its list of descendants,

Table 1. Move Attributes

A40—From square
A41—To square
A42—Special move (yes or no)
A43—Man removed from castle list
A44—Man captured
A45—Square of captured *en passant* Pawn
A46—Ancestor
A47—List of descendants
A48—Irreversibility of move (reversible; or irreversible)
A49—Value of move (mate, no mate, no value)
A50—Number of descendants
A51—Man moved
A52—Double check (yes or no)
A53—Discovered check (yes or no)
A54—Checking move (yes or no)
A55—Descendant's list of mate threats
A56—Threatened mating square
A58—Mating piece on V(A56) square
A63—Reply NOMV (no move)
A65—Number of checking moves generated to date
A66—Number of replies generated to date
A70—List of King replies
A71—List of capturing replies
A72—List of interposing replies

respectively. The log of the analysis is preserved by defining a dummy move, L31, which has on its description list the list of descendants attribute A47. Thus the course of analysis is linked together as a chain of moves with contemporaries linked by ancestor and descendant relations, as in the example of an analysis tree in Fig. 3.

Second, because of the strong family ties just described, one can eventually crawl one's way down any branch of any node and then back up. That is to say, one can make or unmake any move. Two routines, E65 and E66, make and unmake moves, respectively. The procedure for unmaking a move is exactly the reverse. With the help of routine E11 the position list is restored, that is, the description lists of the pieces and squares affected are restored and the signal cells reset.

At any node a new limb may also be constructed (by routines E51 and E52; see Appendix) simply by specifying the "from square" and the "to square" (and special move status, if any) whereupon the move is added to the list of descendants, V(A47), and assigned an ancestor, V(A46).

Third, information gathered in the search for mate is stored on the description list of the move that gathers it. (See Table 1.) When a move is constructed, its ancestor is always assigned as a value of A46 and the man moved in always assigned as a value of A51. Conditionally, a move is assigned a value of A43 if a Rook or King is removed from the castle list, a value of A44 if a man is captured, a value of A45 if a Pawn is captured *en passant*, a value of A52 if it is a double check, a value of A53 if it is a discovered check, and a value of A54 if it is a checking move at all.

Evaluative information is also gathered: attribute A49 records the win-loss value of a move: mate, no mate, or no value. If a checking move has no descendants, that move mates; consequently, attributes A70, A71, and A72 record the kinds of replies to check. Attribute A47 lists the descendants in toto and the value of A50 is a count of them.

The point here is to illustrate how information is hung on the move tree as it is gathered. How the information is retrieved and utilized is a topic for the next section.

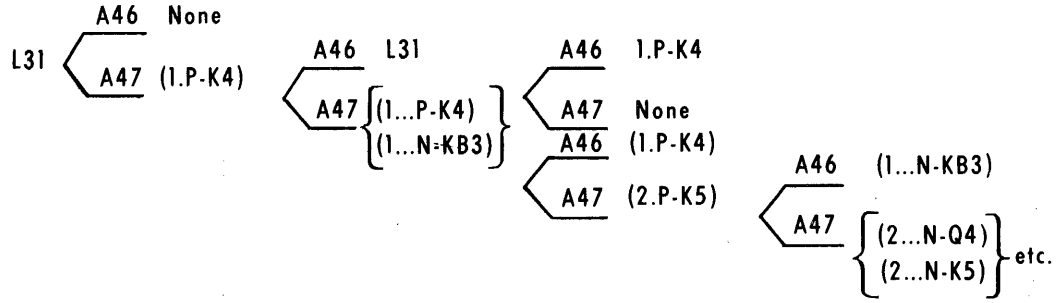


Figure 3. Course of analysis linked as contemporaries, ancestors, and descendants.

4. THE EXECUTIVE AND HEURISTICS OF SEARCH

Introduction

In a given position, what moves should be considered and in what order? Human chess players are known to be highly selective in the moves they look at, a selectivity based on their heuristics of search. Computers must also incorporate such selectivity. What follows then is a discussion of the search heuristics incorporated into the early version of the mating program, some measures on its search behavior, a brief description of the routines that effect the move generation, and, finally, later developments added to a second version of the program, MATER II.

MATER I

Restricting the mobility of the opponent's pieces is a recognized principle of chess strategy. It is particularly important in checkmating combinations since checkmate is defined as an unopposed attack on an enemy King whose mobility has been reduced to zero. Strategically this means the attacker strives to gain control over 1) the square the enemy King occupies, as well as 2) all the squares contiguous to it that do not contain an enemy piece. If just condition (1) obtains, the enemy King is simply in check; if just condition (2) holds, the enemy King is stalemated; while if both (1) and (2) hold, he is checkmated. Viewed in this light, checkmate is a process of acquiring controls, or more and more restricting the enemy King's mobility. This principle is the cornerstone of the mating program.

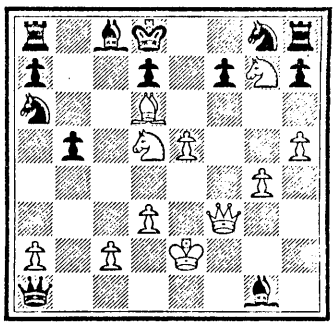
The restriction of mobility principle applies to the generation and selection of moves as well as to decisions about when to abandon search in certain directions. Thus in the mating program: Only checking moves (in MATER I) and moves that threaten mate in one move (in MATER II) are generated for the attacker; the move selected for investigation is the one that most restricts the opponent's mobility; and search is continued down a chosen path only so long as the opponent's mobility is on the decline. This—the rate of growth of the search tree—is an alternative formulation to an evaluation function for terminating search in a particular direction.

Before illustrating the flow of control and the program's executive structure, it is necessary to introduce the notion of a try-list, a notion similar to the "pool of subgoals" in the Logic Theory Machine.[13] Since only one move can be tried out at a time in any particular position, other "eligible" checking moves must wait their turn on a list, L35—the try-list. This list has two noteworthy properties: first of all, it is an ordered list, and second it is independent of a move's level.* Such independence proves powerful in directing search.

The list is ordered by a fewest-replies heuristic: highest priority goes to moves with the fewest number of legal replies, while checking moves with more than four legal replies are discarded entirely. Ties are broken by giving priority to double checks, then to checks that have no capturing responses, then to the order in which the checks were generated. The second property—that checks from all levels are mixed—effects the evaluative principle that search is continued down a particular path only so long as the opponent's mobility is on the decline: when the number of replies at some node in the current line of investigation is equal to or greater than the number of replies at some prior node, the current line is abandoned, the prior node restored, and the alternative that had once been passed over is tried. This nips in the bud unpromising proliferations in the move tree.

In addition to the notions just described—rate of growth serving to terminate search in a particular direction, the set of considerable moves serving to restrict the set of applicable operators in the given position, the try-list ordered by the fewest-replies heuristic serving to stipulate the application of offensive operators—some heuristics of chess strategy serving to stipulate the order of application of defensive operators can also be seen more clearly from the following illustrative position taken from Fine[4] and MATER I's performance on it.†

The following layout is adapted to the simple recursive scheme of Fig. 2.



*The level of a move refers to its depth in the move tree, i.e., how many moves out it is from the starting position.

†They can also be seen more clearly within the picture of heuristic search in general in Newell and Ernst.[14]

(1) Generate checking moves: 1.N-K6ch; B-K7ch; B-B7ch; Q-B6ch.
(2) Select one move for further analysis:
(a)

1.N-K6ch	1.B-K7ch	1.B-B7ch	1.Q-B6ch
1...K-K1	1...NxB	1...NxB	1...NxQ
1...QPxN	1.	1.	1...N-K2
1...BPxN			2.
3.			

(b) Transfer checks to try-list; order them by their "u" values (number of replies):

L35	u
1.B-K7ch	1
1.B-B7ch	1
1.Q-B6ch	2
1.N-K6ch	3.

(c) Select and delete top move from try-list L35: 1.B-K7ch.
(3) Make that move (1.B-K7ch) in the analysis.
(4) Test if checkmate has been achieved: No.
(5) Generate replies to relieve check: 1...NxB.
(6) Select "best" reply*: 1...NxB.
(7) Make that reply (1...NxB) in the analysis.
(8) Test if check has been relieved: Yes.

(1) Generate checking moves: 2.N-K6ch.
(2) Select one move for further analysis.
(a) For each check generate and count replies:

2.N-K6ch
2...K-K1
2...QPxN
2...BPxN
3.

(b) Transfer check to try-list: order them by their "u" values:

L35	u
1.B-B7ch	1
1.Q-B6ch	2
1.N-K6ch	3
2.N-K6ch	3.

(c) Select and delete top move from try-list L35: 1.B-B7ch.
(3) Make that move (1.B-B7ch) in the analysis, restoring the board to the initial position.
(4) Test if checkmate has been achieved: No.
(5) Generate replies that relieve check: 1...NxB.
(6) Select best reply: 1...NxB.
(7) Make that reply (1...NxB) in the analysis.
(8) Test if check has been relieved: Yes.

(1) Generate checking moves: 2.N-K6ch; Q-B6ch.
(2) Select one move for further analysis.
(a) For each check generate and count replies:

2.N-K6ch	2.Q-B6ch
2...K-K1	2...NxQ
2...QPxN	2...N-K2
2...BPxN	2.
2...NxN	
4.	

*"Best" is defined in the text immediately following this example.

(b) Transfer checks to try-list; order them by their "u" values:

L35	u
1.Q-B6ch	2
2.Q-B6ch	2
1.N-K6ch	3
2.N-K6ch	3
2.N-K6ch'4.	

(c) Select and delete top move from try-list: 1.Q-B6ch.
(3) Make that move (1.Q-B6ch) in the analysis, restoring the board to the initial position.
(4) Test if checkmate has been achieved: No.
(5) Generate replies to relieve check: 1...NxQ; N-K2.
(6) Select best reply: 1...NxQ.
(7) Make that reply (1...NxQ) in the analysis.
(8) Test if check has been relieved: Yes.

(1) Generate checking moves: 2.N-K6ch; B-K7ch; B-B7ch.
(2) Select one move for further analysis.
(a) For each check generate and count replies:

2.N-K6ch	2.B-K7ch	2.B-B7ch
2...K-K1	0.	2...NxB
2...QPxN		1.
2...BPxN		
3.		

(b) Transfer checks to try-list; order them by their "u" values:

L35	u
2.B-K7ch	0
2.B-B7ch	1
2.Q-B6ch	2
2.N-K6ch	3
1.N-K6ch	3
2.N-K6ch	3
2.N-K6ch'	4.

(c) Select and delete top move from try-list L35: 2.B-K7ch.
(3) Make that move (2.B-K7ch) in the analysis.
(4) Test if checkmate has been achieved: Yes.
(6′) Select next best reply: 1...N-K2.
(7′) Make that reply (1...N-K2) in the analysis, restoring the board to the appropriate position.
(8′) Test if check has been relieved: Yes.

(1) Generate checking moves: 2.N-K6ch; BxNch; B-B7ch; QxNch.
(2) Select one move for further analysis.
(a) For each check generate and count replies:

2.N-K6ch	2.BxNch	2.B-B7ch	2.QxNch
2...K-K1	0.	2...NxB	0.
2...QPxN		1.	
2...BPxN			
3.			

(b) Transfer checks to try-list; order them by their "u" values:

L35	u
2.BxNch	0
2.QxNch	0
2.B-B7ch′	1
2.B-B7ch	1
2.Q-B6ch	2

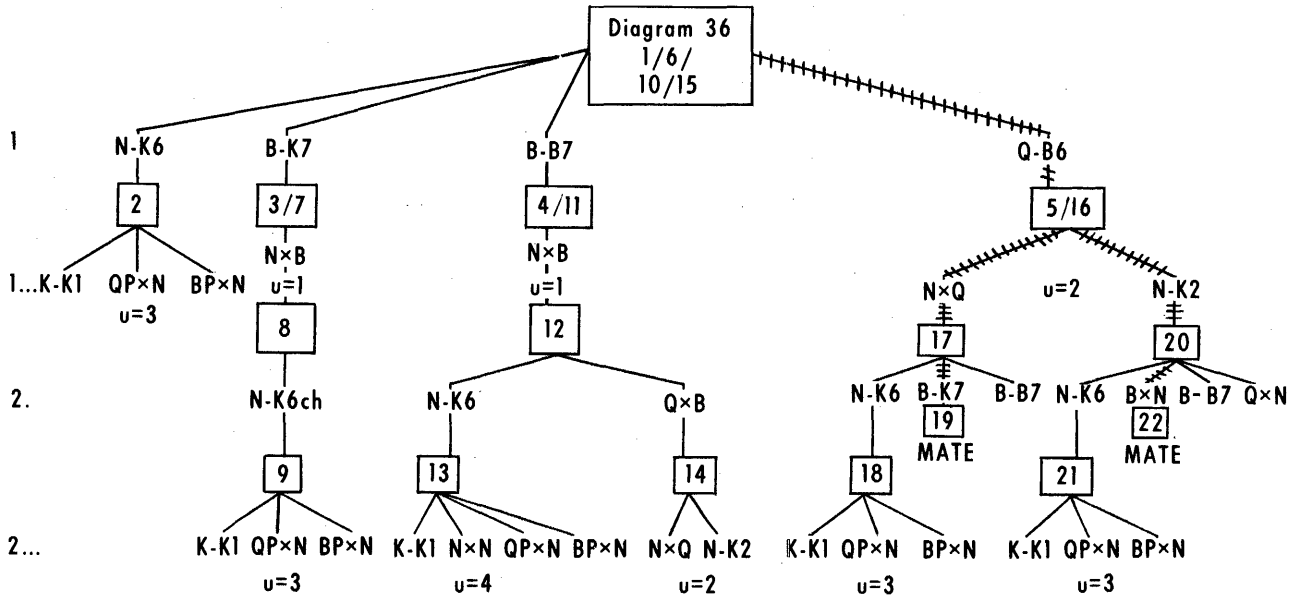


Figure 4. MATER I's analysis tree of Diagram 36, Fine[4].
(The u's are a tally of the number of replies by which the priority of moves on the try-list is established. The square boxes represent positions and the numbers in them trace the course of the investigation—the order in which the positions were taken up. The crosshatched branches trace the mating path.)

2.N-K6ch″3
2.N-K6ch‴3
1.N-K6ch 3
2.N-K6ch 3
2.N-K6ch′ 4.

(c) Select and delete move from try-list L35: 2.BxNch.
(3) Make that move (2.BxNch) in the analysis.
(4) Test if checkmate has been achieved: Yes.
(6′) Select next best reply: None.
(7′) Make that reply (None) in the analysis.
(8′) Test if check has been relieved: No.
Mark MATE and print move-tree (Fig. 4).

This example also illustrates the criteria by which the order of application of defensive moves is accomplished: by "best" reply is meant that reply that seems most likely to give the attacker trouble. Thus the priority of defensive moves Black tries is, first, the capture of the most valuable White pieces by the least valuable Black pieces followed by King moves, interpositions, then order of generation. Again this is an attempt to clip unnecessary proliferations in the move tree: if there is a "killing" reply to a checking move, further analysis of that checking move would seem futile.*

*This is the minimax assumption; namely, that the opponent will make his strongest reply at every opportunity. McCarthy's *killer heuristic* (see Kotok[15]) assumes that a killing reply to one checking move may be a killing reply to other checking moves and thus should be looked at first.

Measures of Search Behavior. Many measures of search behavior can be picked off an analysis tree like MATER I's of Diagram 36 (Fig. 4). For example, the tree can be characterized by counting the number of positions or the number of moves. Simon and Simon[3] call the total number of positions examined the "exploration tree"; in Diagram 36, above, the position count yields an exploration tree of size 16. In general, however, more moves are seen than positions are investigated, which is to say that some moves remain unexplored, such as the replies to 1.N-K6ch in the example above. The count of moves seen—the uninvestigated as well as the investigated ones—will be called the discovery tree; in Diagram 36, above, this move tally yields a discovery tree of size 36 (14 checks and 22 responses). One further refinement can be carved out of the exploration and discovery trees; namely, the "verification tree," which Simon and Simon[3] define as the total number of positions required to prove the combination—the positions resulting from the single best move at each node for the attacker and from every legal move at each node for the defender; respectively, the positive and negative parts of the proof schema.[11] The verification tree "is precisely analogous to the correct path in a maze. It is a tree instead of a single path because all alternatives allowed to the defender must be tested" (Simon and Simon,[3] p. 427). In Diagram 36, above, the

branches of the verification tree are crosshatched, yielding a position count of size 6, or alternatively, a move count of size 5.

These measures do not reveal the time order in which the tree was generated. Human chess players are fickle tree climbers, "progressive deepeners," to use de Groot's[11] term for the phenonenon: "The investigation not only broadens itself progressively by growing new branches, countermoves, or considerable own-moves, but also literally deepens itself: the *same* variant is taken up anew and is calculated further than before" (de Groot,[11] p. 266). In other words, the search strategy is an important structural characteristic of the thought process. In Fig. 4 the order in which positions are taken up is captured by numbering the nodes (positions) in the analysis. These measures will be used for comparative purposes in Section 5.

Routines. How are checking moves and replies actually generated in any given position? There would seem to be two tacks, corresponding to a one-many approach and a many-one approach. In trying to find all the checks in a given position, for example, one could either radiate out from the enemy King and from each square, search for a piece that can get there and give check (the one-many approach), or converge from the squares along the move directions of each attacking piece onto the enemy King's square (the many-one approach). If there are many pieces on the board, the former is the more efficient; if few, the latter.

G1 is a master routine that procures all checks in a given position. It employs the many-one approach, calling subroutine G11 for Queen, Bishop, and Rook checking moves: G12 for Knight and regular Pawn checks; and G13 for double Pawn moves that administer check. Similarly, R21 procures all replies to a given checking move: R11 generates all the King moves that get out of check, R12, all the captures of the checking piece, and R13, all the interpositions. In this way the mating program is able to enumerate all checks and all replies in a particular position.

MATER II

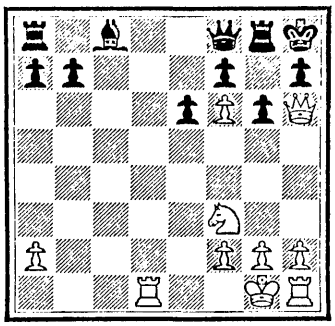
In designing search programs it is useful to distinguish the strategy of search from the information that is gathered during the search. The search strategy tells where to go next, and what information must be kept so that the search can be carried out. It does not tell what other information to obtain while at the various positions, nor what to do with the information after it is obtained. There may be strong interaction between the search itself and the information found, as in the decision to stop searching, but we can often view this as occurring within the confines of a fixed search strategy.

(Newell and Simon,[16] pp. 24–25)

MATER II adds a modification to MATER I's search strategy by bypassing the fixedness in the order of application of operators inherent in the try-list. The new search rule states: in the given position pursue immediately and in depth all checking moves that keep the enemy King stalemated (or nearly so), i.e., moves that can only be answered by captures and/or interpositions or, in the absence of both, by one and only one King move (the "nearly so" condition). In addition to altering the program's search strategy by telling it "where to go next," this procedure also gathers information about the position. In this respect it resembles what de Groot[11] has called a "sample variation," a kind of trial balloon sent up for the express purpose of gathering information to direct subsequent investigation; in this sense it is orientative. Before turning to what information is gathered and how it is used, it should be mentioned that sometimes a sample variation pays off directly—the "sample moves" may be a path to a quick mate.

Specifically, a routine G10 conducts the preliminary search and, if no "easy mate" is found, records the sequence of moves investigated on a list B4. A routine G17 makes use of this information later in drawing up a plan of attack. Just how these routines operate can best be seen by considering in three parts MATER II's analysis for a particular position, Diagram 97 from Fine.[4]

The first part has to do with the preliminary search; the second with the use to which the recorded information is put in drawing up a plan; and the third with the exploration and verification of that plan.



The first 10 moves of the discovery tree are those in Fig. 5a. (Note that both the 1.Q-N7ch and 1.QxRPch sample variations are recorded on list B4.)

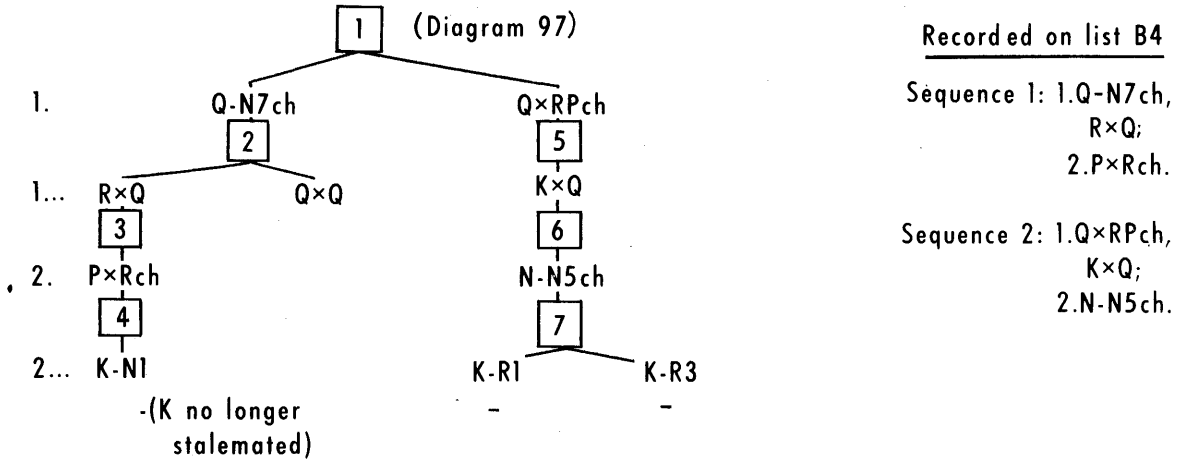


Figure 5A. MATER II's preliminary search in Position 97.

Clearly, the "wishful thinking" goal of the sample variations went unfulfilled; the preliminary excursions did not yield mate. They do yield two sequences of forcing moves, however, that may be useful in constructing a plan of attack. Indeed, the routine G17 searches list B4 for the first move candidates and finds, in this example, 1.PxR and 1.N-N5. The former is rejected as illegal in the current position while the latter is deemed considerable. Note that the set of operators that may be applied in the initial position is expanded in MATER II to include moves other than just checking moves, yet the means by which these are generated continues to ensure a high degree of selectivity.

Routine G17 asks if a proposed move, in this case 1.N-N5, threatens mate in one move. It determines the answer by assuming that Black does nothing on his turn, that is, by playing a "No Move" and then seeing if White can enforce an immediate checkmate. And, indeed, 2.QxRP is mate. In other words, White leaves the actual problem space to seek a mate in a simplified planning space (see Newell, Shaw and Simon[8]) and, in fact, the second part of the move tree is given over to solving the problem in the planning space (Fig. 5b).*

Finally, the third stage is devoted to testing the soundness of the plan; that is, suppose Black tries to avert 1.N-N5 and 2.QxRPmate. Can he? It happens that in this position he cannot so that the exploration tree and the verification tree are identical in this stage of the analysis. (The rather lengthy third stage is omitted here.)

MATER II contains one other highly selective mechanism for finding moves that threaten mate in one. Controls exerted over the enemy King's square and the squares in this immediate vicinity are built into a list structure called the King's Sector. For a given square five kinds of control have been defined:

1. No control—the enemy King can move to the given square.
2. Attacking control—the attacker can move to or capture on the given square.
3. Occupation control—one of the attacker's pieces occupies the given square.
4. Block control—one of the defender's pieces occupies the given square.
5. X-ray control—the attacker can unmask an attacker control by removing

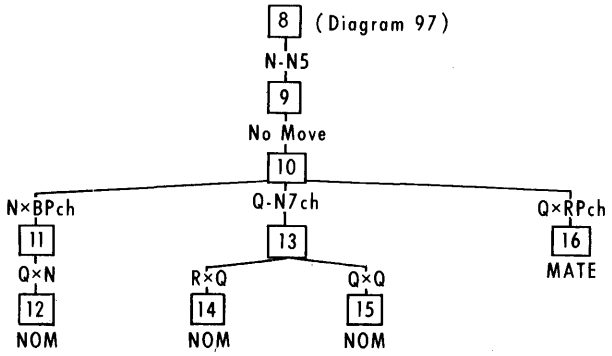


Figure 5B. MATER II's plan formation in Position 97.

*A hybrid version of MATER I and II would first have reinvestigated 1.Q-N7ch and 1.QxRPch, invoking the fewest-replies heuristic, transferring these two checks to the try-list, and elaborating them, before even considering moves which threaten mate in one. Unfortunately the statistics gathered on various versions of the program are too incomplete to say which search strategy is superior across positions, if in fact a correct strategy can be determined independent of position.

one of his own pieces (corresponding to a "discovery" in chess jargon) or he could unmask an attacking control but for an enemy interposer (corresponding to a "pin").

The King's Sector, L40, is constructed by the four routines E91-E94. The complete structure of L40 and the information contained therein can best be seen in Fig. 6, the King's Sector for Diagram 70 from Fine.[4] Attribute Y3 has data term XO as its value, a tally of the number of uncontrolled squares in the Sector (see Fig. 6).

How is all this information retrieved and used by the program? First, a routine G14 tries to generate mate-threatening moves by converting an X-ray control into a second attacking control. For example, in Diagram 70, routine G14, seizing on the White Bishop's X-ray control of KN8, proposes 1.BxBP but then rejects the move because 2.B-N8 does not administer check, let along mate. Second, a routine G19, given one attacking control, tries to add a second. Routine G19, seeing an attacking control over KN7 in position 70, proposes to add another with the moves 1.Q-KB6, 1.Q-N6, and 1.Q-R6. Since all three produce mate if Black does nothing, all three are accepted as considerable moves in the plan.

In summary, MATER II contains several mechanisms for generating a selective set of considerable moves. Incorporating MATER I's ability to generate all checking moves and all replies in a given position, MATER II goes on to generate mate-threatening moves based either on their earlier appearance in forced sequences of checking moves or on the function they serve in controlling key squares around the enemy King. Moreover, MATER II has a set of routines, R15, R16, R22, for generating defensive replies to a threatened mate.

MATER II also contains three principal mechanisms in its search strategy specifying the order in which moves are to be considered. Defensively, in reply to checking moves, captures are preferred to King moves, which, in turn, are preferred to interpositions; while in reply to one-move mate threats, captures are preferred to moves that defend the mating square as well as to interpositions and King runs. Offensively, search is directed by pursuing particular moves in depth so long as the enemy King remains very highly constricted, and then later by pursuing the move that leaves the opponent with the fewest replies. Each of these search evaluators rests on a single criterion: sometimes a line of search is terminated because the defender is left with King moves in reply (nodes 4 and 7 in the move tree of position 97); sometimes a move is rejected because it does not produce immediate mate (nodes 11 and 12 in position 97); sometimes a move just never gets off the waiting list (node 2 in position 36); and

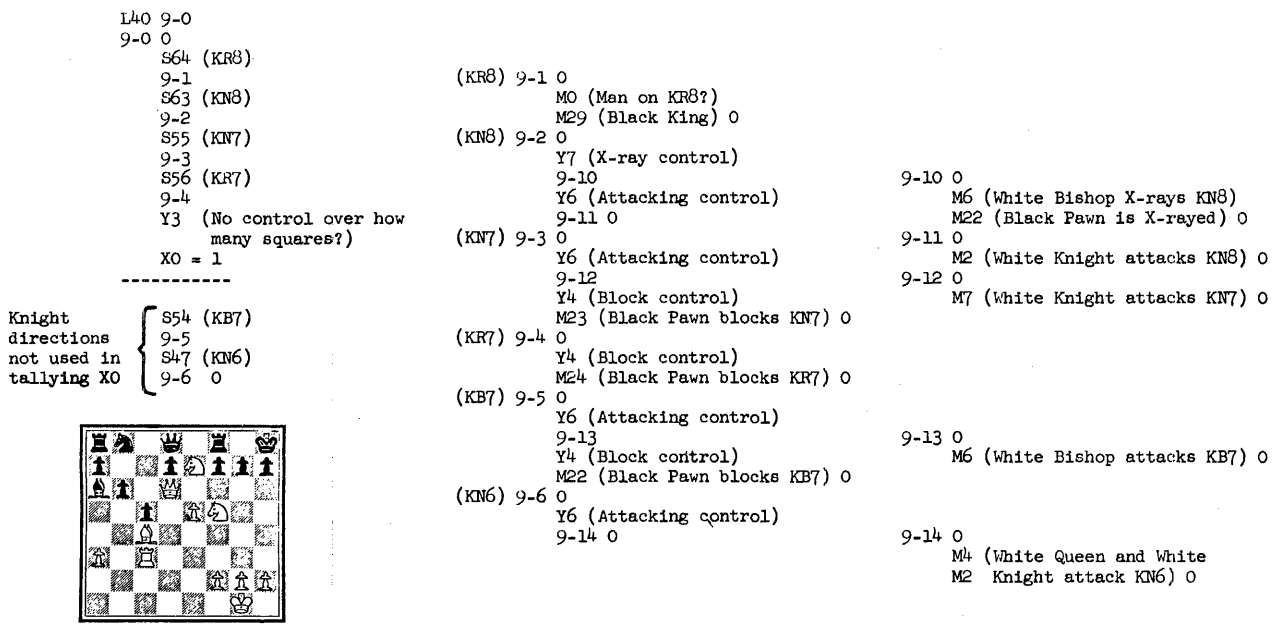


Figure 6. IPL-V List Structure of King's Sector Controls (arranged in attribute-value pairs) of R. Fine's[14] Diagram 70, from a game Alekhine-Supico, 1942.

checking moves with more than four legal replies are always rejected out of hand. Indeed, it is the thesis here that these kinds of criteria, criteria based on features of the task area, are what regulate chess players' choice-of-move decisions and form a good alternative representation to complicated weighting functions of the sort employed by Samuel[17] in checkers and Bernstein and Roberts[18] in chess. Even though mating combinations are the only facet of the game in which the final evaluators, MATE and NOMATE, are well defined*—a degree of certainty attained nowhere else in the game—the search-directing decisions intermediate to the final choice of move must all be made on far less than certain criteria, just like the rest of the game.

Except for the sample variations recorded on the list B4, the information-gathering mechanisms rely on the description list of the move that gathered it, including the final evaluation, MATE or NOMATE, which are propagated back up the tree by the minimax inference procedure in an attempt to demonstrate the proof of a combination.

5. INTERPRETATION AND RESULTS

Introduction

With respect to the verification of simulation models in general, and problem-solving models in particular, two criteria for assessment seem to have emerged clearly: an achievement criterion and a process criterion. That is, can the model solve the class of problems it was designed to handle, and are its mechanisms for doing so equivalent to, or even comparable to, a human problem solver's? The answer to the first question is relatively straightforward; not so to the second, however, since the requirements for equivalence or comparability of process are themselves open to question. In the present report, we shall not consider questions of human simulation but will confine ourselves to a discussion of the achievement of the programs.

MATER I solves combinations which consist of uninterrupted series of checking moves, given that the defender at no node in the verification tree has more than four legal replies; MATER II solves combinations that begin either with checks or with one-move mate threats and checking moves thereafter. This limitation on the class of moves the program can see restricts severely the class of combinations on which the model can be tested. Nevertheless, the program has been tested on material taken from Fine's[4] chapter on the mating attack. Solutions to one class of positions in the chapter call for an uninterrupted series of checking moves ending in mate (51/129 positions). Another class of positions is solved with one-move mate threats and checking moves thereafter (5/129 positions). In the residual class, mate can either be averted through a sacrifice of material or the mate is not "forced," as the term was defined in Section 1 (73/129 positions).

MATER I's Achievement

MATER I found solutions to 43 of the 51 mating positions. The machine missed one combination entirely by failing to move a Pawn that gave a discovered check and exhausted available space before finding the other seven.* Table 2 breaks these 43 positions down according to certain structural measures of search behavior: the depth of search to mate (D), the mean size of the verification tree necessary to prove the combination ($\overline{VT}$ measured in moves), and the mean size of the discovery tree generated in searching for mate ($\overline{DT}$ measured in moves). These latter two measures were defined in Section 4.

Table 2. MATER I's Performance on 43 Positions from Fine[4]

N (positions)	D	$\overline{VT}$	$\overline{DT}$	$\overline{VT}/D$	$\overline{DT}/D$	$\overline{VT}/\overline{DT}$
15	2	3.5	15.5	1.8	7.8	4.4
11	3	5.4	24.6	1.8	8.2	4.6
14	4	9.9	61.5	2.5	15.4	6.2
2	5	11.0	56.0	2.2	11.2	5.1
1	8	17.0	108.0	2.1	13.5	6.4

Simon and Simon[3] (p. 428) suggest depth, number of positions in the exploration tree, and number of positions in the verification tree as measures of the difficulty of combinations. They remark that the four positions in their sample "are not ordered in the same way with respect to the different measures of difficulty." Using depth, number of moves in the discovery tree ($\overline{DT}$), and number of moves in the verification tree ($\overline{VT}$) as equivalent measures, the

*"Well defined" is used in the sense that there exists a satisfactory test that enables the player to recognize the solution to his problem.[19]

*In particular, MATER I failed to see 2.P-B6ch in position 148. The seven positions that exhausted available space did so because the fewest replies heuristic failed to discriminate among alternative checking moves: among six alternative discovered checks in positions 41 and 100; among a large number of initial checks available in positions 109 and 140; and among a large number of checks in depth involved in a King hunt on the open board in positions 111, 130, and 157.

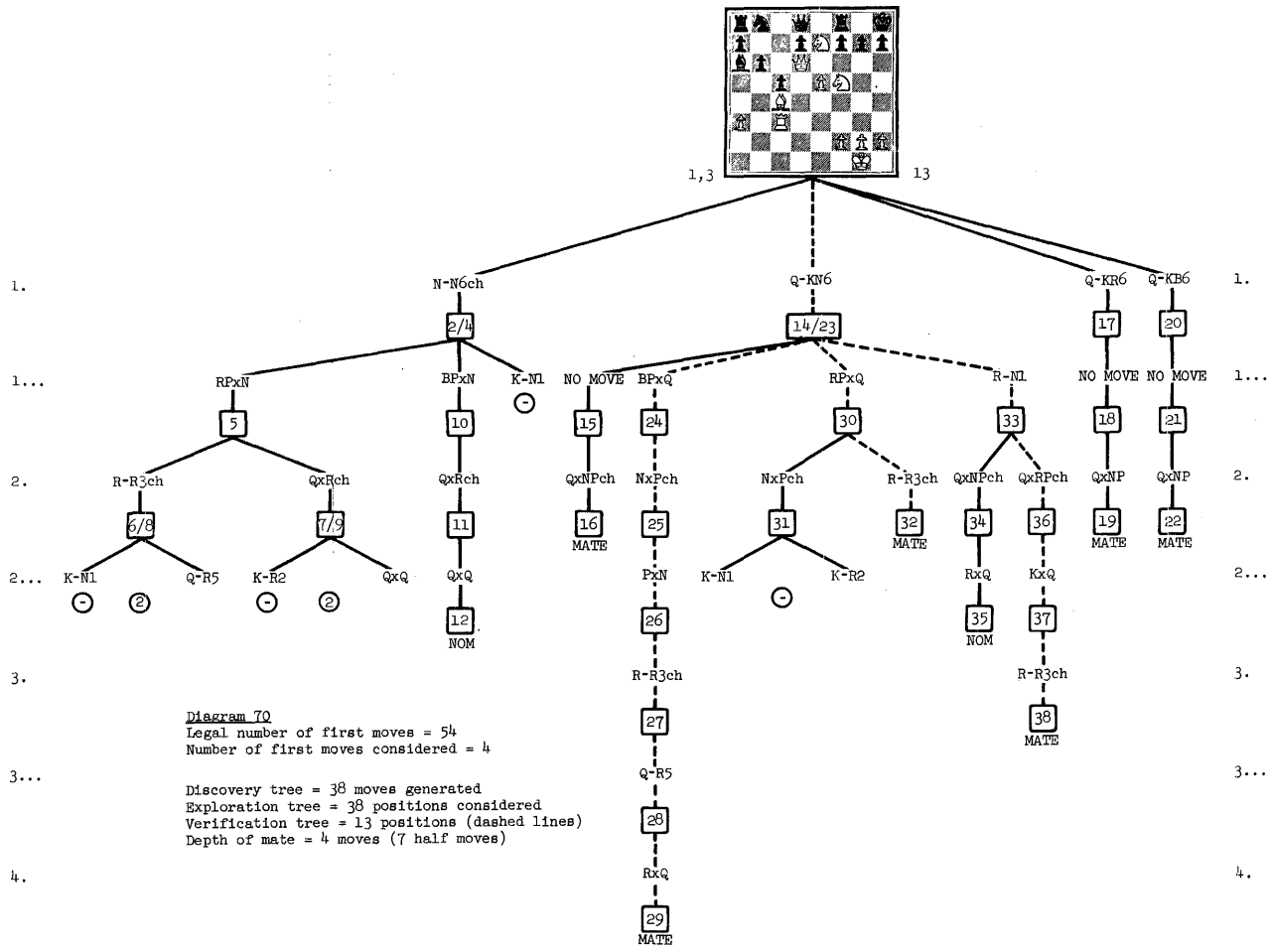


Figure 7. MATER II's Analysis Tree and Order of Search on Diagram 70, from Fine[4].

data of Table 2 do show, with but one exception, the same ordinal relationship across measures, at least when averaged over the 43 positions.

Between depth and the size of the verification tree ($\overline{VT}/D$) there is a fairly close correlation—around two moves in the verification tree per move in depth. This confirms the Simon and Simon[3] observation: the tree varies linearly, not exponentially, with depth, and it probably is this property that makes deep analysis possible in combinations. Neither $\overline{DT}/D$ nor $\overline{VT}/\overline{DT}$ shows any consistent relationship.

The only roughly constant ratio, $\overline{VT}/D$, has more to do with the characteristics of mating positions than with characteristics of the mating program. The only measure on the program's search behavior is $\overline{DT}$ and there seems to be no consistent relationship between it and the two measures on the combinations ($\overline{DT}/D$ and $\overline{VT}/\overline{DT}$).

MATER II's Achievement

MATER II has been tested on all five positions from Fine[4] that necessitated an initial threat of mate in one and checks thereafter. It solved three directly, the other two, because of a change in computer facilities, by hand simulation. The search tree for position 97 has already been described. In position 107 the initial Queen sacrifice as well as an unexpected Bishop sacrifice were easily spotted in the service of mate. The analysis tree of position 70 is given in Fig. 7. Note that the correct move and theme in position 70 derive from the celebrated game of Marshall's for which spectators showered the chessboard with gold coins!* Of the two hand

*The program also finds the correct sequence of moves from the immortal Lewitzky-Marshall game, played in Breslau in 1912 (Diagram 69 in Fine[4]); it is excluded from the count here since Lewitzky, had he not chosen to resign, could have averted mate at the cost of a piece.

simulated runs position 95 required but the simple addition of a second control on the KN7 square via 1.Q-R6, while position 113 required the move 1.R-R5, which had been discovered in one of the exploratory sample variants.

Conclusion

In conclusion, MATER's power stems from its ability to generate a small selective set of moves that merit investigation. Since most of the earlier chess programs (see the review in Newell, Shaw and Simon[20]; and Kotok[15]) spent their analysis time processing the wrong moves, it would seem that MATER II's two major mechanisms for generating relevant moves—its reliance on the sample variations and on the control of key squares—warrant further research. MATER II's major weakness, on the other hand, lies in its poorly organized search strategy for using its selectivity at all points in the analysis process.

On the horizon, proposals have been made for strengthening the program's perceptual capabilities as well as altering its search strategy.

Appendix

THE BASIC REPRESENTATION

Statement of the Problem

In this Appendix we describe the basic representation implemented by Newell and Prasad.[6] Two interrelated questions guided the choice of representation:

1. What are the necessary components of a chess representation?
2. How should this information be organized?

In response to (1): The program should be able to "see" the same things a human sees when he looks at a chessboard. Thus the program requires an internal representation of the squares and pieces on a chessboard and the relations among them, and a set of processes that can pick off and make use of these relations as needed. The former requirement is called "setting up the chessboard," the latter "move-making and board processing capabilities."

In response to (2): The game of chess provides an inhomogeneous collection of information out of which moves must be forged. Thus there must be enough variety in the representation to discriminate all the different kinds of moves; given that, the information should be stored in such a way that little space is allotted to moves that seldom occur (such as Pawn promotions, castling, etc.), and the dependence and division of information between routines and data should remain flexible and open to change and never solidify into a resistant collection of conventions (Newell[21]). List processing languages are specifically designed to cope with such problems.

A chessboard is made up of squares, which lodge pieces, which make moves from one square to another. Objects in chess, like the 64 squares and 32 men, can be represented as symbols on lists, and moves can be represented as names of description lists with certain prescribed associations (such as the square *from* which a piece comes, the square *to* which it moves, and the kind of move in question). A chess position, moreover, can be fully described as a list of pieces and squares and a chess game as a list of moves that originate from a standardized initial position and terminate in a well-defined checkmate position.

Setting up the Chessboard

A chessboard is made up of eight ranks and eight files which rule off 64 squares. The sequence of symbols S1...S64 is used to denote these 64 squares.

In the data section of the program there are a list of ranks, L1, containing members R1 through R8, and a list of files, L2, containing members F1 through F8. Each rank is itself the name of a list containing eight member squares; e.g., R1 contains S1, S2, ..., S8.

In the routines section of the program a superroutine, E1, sets up a chessboard; it calls nine routines E2-E7, E9, E10, and E12, which do the work. Routine E2 builds the eight file lists, F1 through F8, out of the rank lists, R1 through R8. Then routine E12 takes each of the 64 squares and assigns it rank and file (x,y) coordinates, which are later used to compute another set of relations among squares.

Squares. For each square on a chessboard it is essential to know: 1) the name of its occupant, if there is one, and 2) the name of all its neighboring squares in the chess-legal directions. The first desideratum is effected by defining an attribute MO, "Man on Square?" on the description list of every square and assigning as its value the name of the piece occupying it—if there is one.

The extensive network of relations among squares, constituting all legal move directions in chess, is captured by defining 16 directions on the chess-

board, beginning with D1 for the forward direction and continuing clockwise to D16 for forward and left (e.g., the Knight's move KN1 to KB3).

Thus the even numbers (D2, D4, . . . ,D16) define the eight possible Knight move directions; half the odd numbers (D1, D5, D9, D13) define rank (horizontal) and file (vertical) directions; the other half (D3, D7, D11, D15), diagonal directions.

Routines E3, E4, E5, and E8 use lists L1, L2, L5, and L6 to build the network of relations among squares by assigning to each of the 64 squares all surrounding squares as values of each of the 16 directions that obtain. In the initial position, for example, the list structure of the square S8—White's KR1 in standard American chess notation—would look like this:

Name of list	*Attribute*	*Value*
Square KR1 (S8)	Man on Square (M0)?	White King Rook (M8)
	Square to the North (D1)?	Square KR2 (S16)
	Square to the West (D13)?	Square KN1 (S7)
	Square to the WNW (D14)?	Square KB2 (S14)
	Square to the NW (D15)?	Square KN2 (S15)
	Square to the NNW (D16)?	Square KN3 (S23).

Note that in such a list structure representation only those five of the 16 legal directions that are needed are defined; space in memory is not consumed by providing the "information" that the other 11 directions have *no* values, as it would seem to be in a matrix representation of this kind of data. (Cf the argument in Newell,[21] pp. 411–412, for a fuller statement of this view.)

Men. The 32 men take and retain their designations from their placement in the initial position; they are denoted by the sequence of symbols M1 . . . M32.

For each piece, it is essential to know:

1. the square he occupies (attribute S0),
2. his type (attribute A1),
3. his color or side (attribute A2),
4. his legally permissible move directions (attribute A20), and
5. his legally permissible capture directions (attribute A21).

The first of these is effected by defining an attribute SO, "Square on?" on the description list of every piece and assigning as its value the name of the square the piece currently occupies. The other four attributes assume the complete range of values in accord with the rules of chess.

In the data structure there are 11 lists that group the chess men by types or otherwise useful categories: White Pawns; Black Pawns; Bishops; Rooks; Knights; Queens; Kings; White Rooks, Bishops and Queen; Black Rooks, Bishops, and Queen; White Knights; and Black Knights.

For each side, moreover, there is a list giving the type of each man on that side, and another list giving the move directions of each type of man.

Routines E6 and E7 assign to each man his type, color, move directions, and capture directions. In the initial position, for instance, the list structure of M8 would look like this:

Name of list	*Attribute*	*Value*
White King Rook (M8)	Man on what square (SO)?	Square KR1 (S8)
	Type of man (A1)?	a Rook
	Color of man (A2)?	White (K10)
	Move directions (A20)?	a list of directions D1, D5,D9,D13
	Capture directions (A21)?	a list of directions D1,D5,D9,D13.

Positions. A chess position can be described fully in terms of squares and pieces. Since MATER is supposed to find a checkmate in any given position, obviously some representation is necessary for encoding a particular position. This representation is a describable list called the position list, L10. Its main list consists simply of the name of each man present on the board and the name of the square each man occupies, arranged in attribute-value pairs. Its description list contains a set of special attributes pertinent to the characterization of the position; in particular, S65, the "Whose move is it?" attribute that flip-flops between K10 (White on move) and K11 (Black on move); S66, the name of the castle list that contains the Kings and Rooks still "eligible" for castling; S67, the signal cell that gets set when an *en passant* capture is in the offing; and S69, the name of the most recent move made on the board.

Routine E10 takes as input any position list—either the initial position or the mating position, which is read in from cards by routine E90—and

converts it into a set of associations between pieces and squares such that every piece has an attribute SO ("Square on?") with the square that piece occupies as value, and every square has an attribute MO ("Man on?") with the name of the chess piece —if there is one—occupying that square as its value.

This ends what might be called the "static" perceptual relations on the chessboard. What follows is a bundle of basic routines that attempts to provide "dynamic" perceptual relations to the program.

Move-Making Capabilities

Moves are the operators that transform one chess position into another. What are the common properties of chess moves? Each involves a piece, or sometimes two, going from one square to another. If the "to square" is already occupied, the move is called a capture. If the "to square" is on the eighth rank and the piece a Pawn, it is called a promotion. But in all cases the common "from-to" property holds.

This permits a move to be represented as the name of a description list containing a "from square" (as the value of an attribute A40) and a "to square" (as the value of an attribute A41). This information is sufficient to specify most moves. There is a special class of moves, however, which, while adhering to this "from-to" pattern, introduce some idiosyncratic properties of pieces. Each of the following five members in this class is assigned a different value to the special move attribute A42: King's side castling, Queen's side castling, a double Pawn move, an *en passant* response thereto, and a Pawn promotion.

Five steps are required to make a move: first, the move must be constructed; second, it must be tested for legality; third, for repetition of position; fourth, it must actually be made on the board; and fifth, it should be printed.

Routines E51 and E52 construct regular moves and special moves, respectively. Both create a new cell or symbol, which becomes the name of the move. Both take as input the square from which the piece is to move and the square to which it is to move and assign these as values of attributes A40 and A41, respectively. For the special move routine, E52, the type of move must also be specified as input; it is assigned as the value of attribute A42. The name of the man moved is also received as the value of another attribute A51, for reasons that will appear under step 3 below. The move 1.P-K4, for example, would be represented as follows (where a_1 and a_2 are internal cell names):

Name of list	*Attribute*	*Value*
a_1	From Square (A40)?	Square K2 (S13)
	To Square (A41)?	Square K4 (S29)
	Man moved (A51)?	White King Pawn (M13).

Similarly, the special move P-K8 = Q would be represented in the following format:

Name of list	*Attribute*	*Value*
a_2	From Square (A40)?	Square K7 (S53)
	To Square (A41)?	Square K8 (S61)
	Special Move (A42)?	Promotion to Queen
	Man moved (A51)?	White King Pawn (M13).

Second, a routine E18 checks to insure that a newly constructed move is legal. The routine tests, for example, whether a Bishop is moving through a Pawn, whether a Rook is making a Bishop move, whether a player is castling through check, and the like. The output is a simple "+" ("yes, the move is legal") or "−" ("no, the move is illegal").

Third, according to the laws of chess a threefold repetition of position constitutes a draw, and, according to the laws of computers, a loop. Before a move is executed, therefore, a routine E55 tests if the move under consideration has been played before in this same position. The position could not have occurred before if the move is irreversible, that is, if once the move is made on the board no subsequent set of legal moves can ever regain the exact same position. Captures, Pawn moves, and castling are all irreversible. Thus when a capturing move is constructed (step 1), it is given an attribute A44 with the man captured as its value. When a castling move is constructed, its status as a special move is recorded as the value of attribute A42. And for a Pawn move, a record is kept via the man moved attribute, A51. Routine E56, called by E55, tests for any of these three conditions to declare a move irreversible. If none of them obtains, E55 must take some further comparisons between the A40 and A41 values of the proposed move and earlier moves.

Fourth, a routine E65 makes a regular move on the board; routines E71–E75 and E81–E85 execute the special moves. A move is made by updating the position list, which is done in two steps: first, with the assistance of routine E11, the description lists of the pieces and squares affected by the move are updated. Second, the signal cells—S65, S66, S67, and S69—affected by the move are reset.

Since different routines are needed to make each of the five special moves, these routines are simply associated with their respective special move values in the data section of the program.

Fifth, there is a print routine, E16, which prints out the name of the move, the "from square," the "to square," and the man captured, if any.

Additional Board Processing Capabilities

In addition to the move-making capabilities just described, there is a second group of routines intended to provide the machine with some more of the perceptual capabilities a human possesses; these are the board processing routines which provide answers to questions asked of the board. Routine E13 finds the direction, if one exists, between two given squares. Routine E14 tests to see if there is a piece between two squares in a given direction, and E24, if there is one and only one piece between two squares in a given direction. E15 tests if a piece is under attack, and routine E26 asks specifically if that piece is the enemy King. Routine E33 tests whether a given square is under attack, while E34 builds up the list of men of a particular color attacking a given square. Routine E36 tests if a given square is defended. These are some of the more important "building block" routines used in constructing the move tree.

Processing Speed

It might be supposed that since the program was written in an interpretive language, IPL-V, without any attempt to provide a special machine-language representation for primitive board manipulations, that it would be a very slow player. This has not proved to be the case—a tribute to the advantages of selectivity over machine brute force. The most difficult mates, requiring the examination of about 100 positions, were achieved in about 10 minutes on a CDC G-20—which would be equivalent to about three minutes for the IPL-V system on the IBM 7090. An excellent human player might be expected to take ten minutes or more to discover and verify the mate in a position of this difficulty.

ACKNOWLEDGMENTS

We would like to acknowledge the assistance of G. A. Forehand, B. F. Green, A. Newell, M. R. Quillian, and R. F. Simmons.

REFERENCES

1. M. M. Botvinnik, *One Hundred Selected Games* (translated by S. Garry), Dover Press, New York, 1960.

2. G. W. Baylor, "Report on a Mating Combinations Program," SDC Paper, No. SP-2150, System Development Corporation, Santa Monica, Calif. (1965).

3. H. A. Simon and P. A. Simon, "Trial and Error Search in Solving Difficult Problems: Evidence from the Game of Chess," *Behavioral Science*, vol. 7, pp. 425–429 (1962).

4. R. Fine, *The Middle Game in Chess*, David McKay, New York, 1952.

5. S. M. Myron and W. H. May, "A Note on Serendipity, Aesthetics, and Problem Solving," *Behavioral Science*, vol. 8, pp. 242–243 (1963).

6. A. Newell and N. S. Prasad, "IPL-V Chess Position Program," unpublished working paper, Carnegie Institute of Technology (1963).

7. H. A. Simon and A. Newell, "Models: Their Uses and Limitations," *The State of the Social Sciences*, L. D. White, ed., University of Chicago Press, Chicago, 1956, pp. 66–83.

8. A. Newell, J. C. Shaw, and H. A. Simon, "The Processes of Creative Thinking," *Contemporary Approaches to Creative Thinking*, H. E. Gruber, G. Terrell and M. Wertheimer, eds., Atherton Press, New York, 1962, pp. 63–119.

9. N. R. F. Maier, "Screening Solutions to Upgrade Quality: A New Approach to Problem Solving Under Conditions of Uncertainty," *Journal of Psychology*, vol. 49, pp. 217–231 (1960).

10. D. Luce and H. Raiffa, *Games and Decisions*, John Wiley & Sons, New York, 1957.

11. A. D. de Groot, *Thought and Choice in Chess* (rev. translation), Mouton and Company, The Hague, 1965.

12. K. Duncker, "On Problem Solving," (translated by L. S. Lees from the 1935 original), *Psychological Monographs*, vol. 58, no. 270 (1945).

13. A. Newell, J. C. Shaw and H. A. Simon, "Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics," *Computers and Thought*, E. A. Feigenbaum and J. Feldman, eds., McGraw-Hill, New York, 1963, pp. 109–133.

14. A. Newell and G. Ernst, "The Search for Generality," *Proceedings of IFIPS Congress 65*, Spartan Books, Washington, D.C., 1965, vol. 1, pp. 17–24.

15. A. Kotok, "A Chess Playing Program for the IBM 7090," unpublished bachelor's thesis, Massachusetts Institute of Technology, 1962.

16. A. Newell and H. A. Simon, "An Example of Human Chess Play in the Light of Chess Playing Programs," *Progress in Biocybernetics*, N. Weiner and J. P. Schade, eds., Elsevier, Amsterdam, 1965, vol. 2, pp. 19–75.

17. A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *Computers and Thought*, E. A. Feigenbaum and J. Feldman, eds., McGraw-Hill, New York, 1963, pp. 71–105.

18. A. Bernstein and M. DeV. Roberts, "Computer vs. Chess Player," *Scientific American*, vol. 198, pp. 96–105 (1958).

19. J. McCarthy, "The Inversion of Functions Defined by Turing Machines," *Automata Studies*, P. E. Shannon and J. McCarthy, eds., Princeton University Press, Princeton, N.J., 1956, pp. 177–181.

20. A. Newell, J. C. Shaw, and H. A. Simon, "Chess-playing Programs and the Problem of Complexity," *Computers and Thought*, E. A. Feigenbaum and J. Feldman, eds., McGraw-Hill, New York, 1963, pp. 39–70.

21. A. Newell, "Some Problems of Basic Organization in Problem-Solving Programs," *Self-Organizing Systems*, M. Yovitts, G. T. Jacobi and G. D. Goldstein, eds., Spartan Books, New York, 1962, pp. 393–423.

MULTIDIMENSIONAL CORRELATION LATTICES AS AN AID TO THREE-DIMENSIONAL PATTERN RECONSTRUCTION*

Samuel J. Penny and James H. Burkhard
Lawrence Radiation Laboratory, University of California
Berkeley, California

INTRODUCTION

Spatial reconstruction of a three-dimensional object from a set of stereo photographs must begin with the matching of the topological characteristics in one view with those in the other views. A human is usually good at this form of pattern recognition, but he is too slow for some applications. For high-speed processing of bubble chamber photographs it has been necessary to design a computer code that will match the images seen in the various views.

If we have n stereo views of an object, where each view contains images of the m characteristics to be correlated, a total of m^n combinations are possible when one image is used from each view. These combinations are known as "n-tuplets." There is then some set containing m of the n-tuplets that describes the true matching of the images. With the physical limitation that an image in one view should not match more than one image in any other view, this solution set must be chosen from a total of $(m!)^{n-1}$ possible sets. This is in the class of combinatorial problems that involve the construction of an ordered set $S = \{s_1, \ldots, s_m\}$ where the s_i are elements of a finite set U and the elements of the set S must be chosen subject to certain restrictions.

One approach for finding the correct solution set would be to compute the likelihood for each possible set of m n-tuplets, and then choose that set with the maximum likelihood. This approach can require a prohibitive amount of computation and is unreliable when there are large errors in the data. An alternative approach would be to use some type of elimination process, but even this can be expensive unless a technique is used to simplify the bookkeeping and to provide a rapid correlation among the various steps in the process. The "correlation lattice" and methods for scanning it for solution sets (by means of an iterative scheme of backtrack programming[1,2]) supply the simplicity and speed to make this approach economical on a computer.

THE THREE-DIMENSIONAL LATTICE

At the Lawrence Radiation Laboratory in Berkeley, much of the high-energy-physics research deals with nuclear-particle events occurring in a bubble chamber. Three stereo views of the event are photographed, and points along the track images are measured. The images must then be matched so that the spatial trajectories of the particles involved in the interaction can be mathematically reconstructed. Using the parameters of the trajectories, we do a kinematic analysis of the interaction to produce the information required by the physicist for his experiments.

Figure 1 is a photograph and sketches of a simple nuclear event that occurred in the bubble chamber.

*This work was performed under the auspices of the U.S. Atomic Energy Commission.

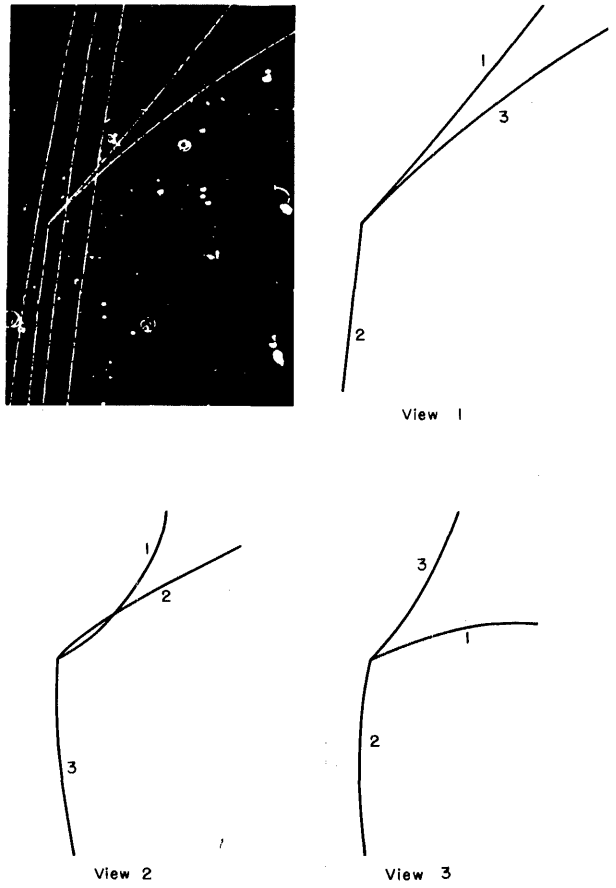


Figure 1. A photograph and sketches of a nuclear-particle interaction.

As charged nuclear particles move through the liquid in the chamber, they leave small tracks of bubbles. A nuclear event occurs when an elementary particle interacts with an atomic nucleus in the liquid and produces a new group of particles moving in different directions.

Numbers have been assigned to the track images in the different views in the figure. The set of image-number triplets that corresponds to the correct track-image matching is

$$S = \{(1,1,3),\ (2,3,2),\ (3,2,1)\} \qquad (1)$$

where, for example, the triplet (1,1,3) means that track image 1 in view 1, track image 1 in view 2, and track image 3 in view 3 are images of the same physical track.

Producing the Lattice

The computer's procedure for matching the track images begins with the construction of the correlation lattice, E. Each dimension of this lattice corresponds to a set of track images from one of the stereo views, and the indices along a dimension are the identifying numbers of the track images in that view. The value, one or zero, of an element e_{ijk} in E will indicate the possibility or impossibility, respectively, of matching image i in view 1 with image j in view 2 with image k in view 3. Since all image combinations must initially be assumed to be possible, the procedure begins with all lattice elements equal to one. Figure 2 shows the initial lattice for our sample event.

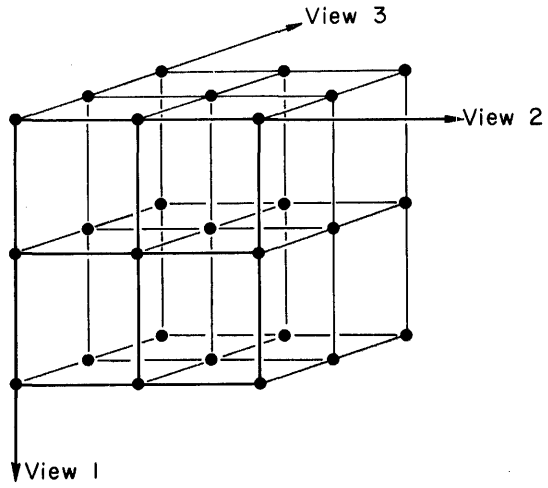


Figure 2. The initial correlation lattice. (In Figs. 2–6, the solid disk ● represents a value one; an open disk ○ represents zero.)

Often the correlation lattice will not be cubic, as it was for our sample event. It is possible that spurious images may be seen or real images missed in some views. With the bubble chamber film, it may be difficult to dissociate the background tracks from the interesting interaction in some view, thus producing spurious images in that view. However, the measuring devices seldom miss track images. The strategy for dealing with the noncubic lattice for track matching is then rather simple: if L is the number of images the code "expects" to match, then every view must have at least L measured track images, and the solution set must contain L triplets. Other applications may require other strategies.

The remainder of the matching procedure is an iterative process of applying successively more difficult tests of physical consistency to the image combinations in an attempt to eliminate matching possibilities from the correlation lattice, and, after each test sequence, scanning the lattice to determine the number of possible solutions sets that remain.

If there is no such set, or only one, the procedure ends. If there is more than one set, the lattice is still ambiguous, and it is necessary to go into the next test sequence.

Testing Image Combinations

A test may be concerned with an image from each of the three views (a triplet test) or it may compare images from only two views (a pair test). The pair test is very effective in the early stages of the procedure since, if an image pair is shown to be inconsistent, any triplet involving that pair must also be inconsistent. A single test can therefore eliminate many elements at a time.

The tests may be expressed as binary-valued functions. A pair test is written as the binary function $F_{P,Q}(p,q)$ of the pair of image indices p and q from views P and Q. When the image pair is not consistent with physical constraints, the value of the function is zero; otherwise its value is one. A similar definition is made of the triplet test $F_{1,2,3}(i,j,k)$.

For efficiency, we do not apply a test to any image combination that has already been eliminated by a previous test sequence. The determination of triplets to be tested with a triplet test is straightforward: A triplet is tested only if its corresponding element in E has the value of one. For the set of pair tests $(F_{1,2}; F_{2,3}; F_{3,1})$, one view at a time must be systematically eliminated from the image combinations by forming the "projections" of E along each of its three dimensions, one at a time. This produces the three two-dimensional-projection lattices ${}_1E$, ${}_2E$, and ${}_3E$, where the elements of these lattices are given by

$$ {}_1E = ({}_1e_{jk}) = \left(\overset{n_1}{\underset{i=1}{.\text{OR}.}} e_{ijk} \right) \tag{2a} $$

$$ {}_2E = ({}_2e_{ij}) = \left(\overset{n_2}{\underset{j=1}{.\text{OR}.}} e_{ijk} \right) \tag{2b} $$

$$ {}_3E = ({}_3e_{ij}) = \left(\overset{n_3}{\underset{k=1}{.\text{OR}.}} e_{ijk} \right) \tag{2c} $$

.OR. is the logical sum operator, i.e.,

OR	0	1
0	0	1
1	1	1

The image pairs to be tested are determined by scanning the three projection lattices for elements with a value of one and testing only the corresponding pairs. The projections of the initial lattice shown in Fig. 2 will obviously produce projection lattices that have elements all equal to one.

Recording Test Results

Now that the image combinations to be tested are determined, it is now necessary to record the results of the test sequence in the correlation lattice E. For a triplet test this simply involves setting the lattice element corresponding to the triplet to zero if the value of the test function is zero. The procedure for recording the results of a pair-test sequence is to record the value of the test function in the two-dimensional-projection lattices that indicated the image pairs to be tested. After all image pairs have been tested, the resulting projection lattices are used to "mask" the correlation lattice E. This masking operation is written as

$$ E' = {}_kE.\text{AND}.E \tag{3} $$

or expressed at the element level for the view $\frac{2}{3}$ projection as

$$ e'_{ijk} = ({}_1e_{jk}).\text{AND}.(e_{ijk}) \tag{4} $$

for all i, j, and k. Here, .AND. is the logical product operator

AND	0	1
0	0	0
1	0	1

The masking is done for all three projections, and the final lattice E' replaces E as the correlation lattice.

Returning to our sample event, suppose that the first test sequence was a pair test and eliminated five image pairs: (1,2) and (2,2) were inconsistent in views 1 and 2, and (1,1), (2,1), and (2,3) were inconsistent in views 1 and 3. No pairs could be eliminated from views 2 and 3. Figure 3 shows the projection lattices containing the test results ready to be masked into the correlation lattice E.

Scanning for Solutions

After masking into the correlation lattice, we now scan the three-dimensional lattice for possible solutions. A solution in the lattice is defined as a set of lattice elements that all have a value of one and are chosen in such a way that no image in any view is used more than once. The length L of the solution

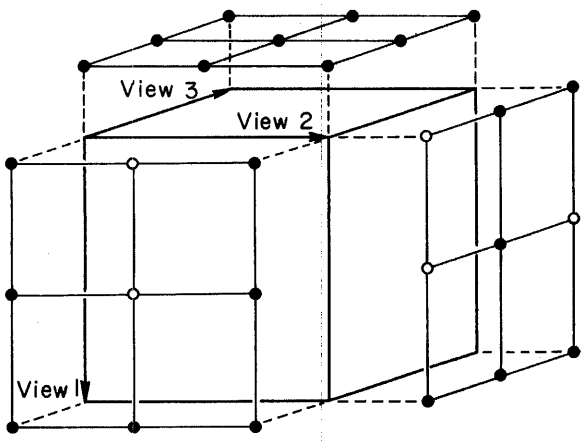


Figure 3. The projection lattices containing the test results ready to be masked into the correlation lattice.

set is defined as the number of elements in the solution set; in our sample event, the length is three.

Figure 4 shows the results of the masking operation on our sample event. The dashed lines represent the two possible solution sets that will be found by the scanning procedure.

The search for solutions in the three-dimensional lattice is based on the following proposition: If there exists a solution set consisting of L triplets—$S_L(E)$—in the three-dimensional lattice E, then at least one two-dimensional-solution set consisting of L doublets—$S_L({}_kE)$—must also exist in each of the projection lattices of E; i.e., a solution in E will project as a solution in any projection of E. A corollary to this proposition is that if no solution set of length L is found in some projection of E, then no solution set of length L exists in E.

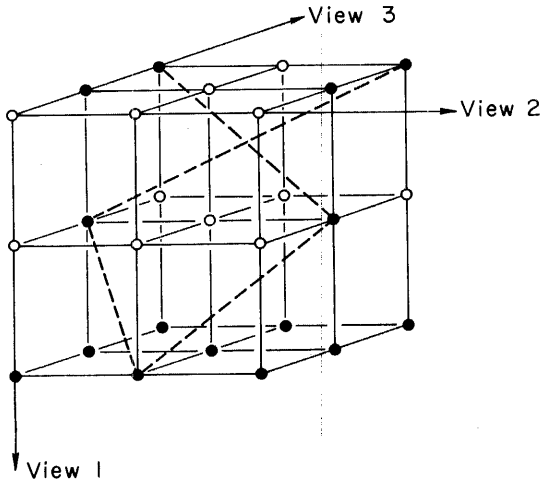


Figure 4. The correlation lattice after making. Dashed lines show the two remaining solutions.

The scanning procedure is a four-step process. In the first step the lattice is reprojected along some axis to produce ${}_kE$ (${}_3E$ in our case). In the second step this two-dimensional lattice is scanned for two-dimensional solutions of the required length by the technique of backtrack programming. When a solution is found, the index pairs forming the solution set are used to construct a two-dimensional "post-projection" lattice P (step three) as described later. In step four this lattice is then scanned for solutions in order to determine the third index of the solution triplets. This type of scan fixes the indices of the triplets one view at a time—i.e., i is fixed for view 1, then corresponding j's are chosen from view 2, and this information is used to pick out the corresponding k's from view 3. These steps are illustrated in Fig. 5.

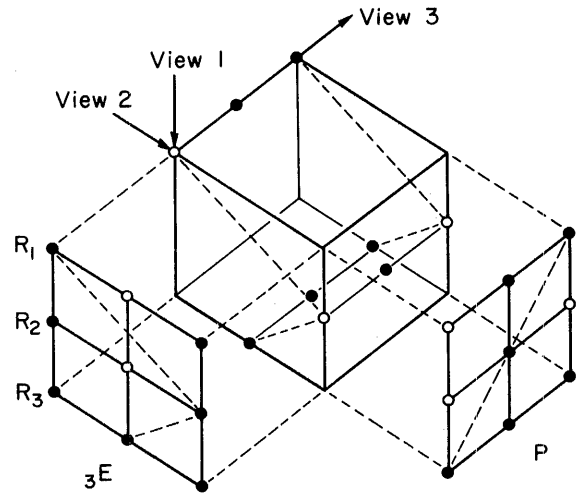


Figure 5. The lattice and projections used in the scanning procedure. The solution in ${}_3E$ defines posts in E to be used as rows in P.

As the three-dimensional solutions are found, they are used to construct a new three-dimensional lattice, E_S, that contains only elements known to be a part of some solution. This lattice will be used later to discard unused elements on the basis of logical inconsistency. (If it is found that an element is not now a part of some solution in E, it can never be a part of any solution, and it can then therefore be ignored in any further processing.)

The first step—to produce ${}_3E$—is the projection procedure of Eq. (2c). By the proposition stated above, any solution in E will project as a solution in ${}_3E$.

The second step—to scan ${}_3E$ for a two-dimensional solution—is done with backtrack program-

ming[1] in the following way. Suppose that we are dealing with a projection lattice of n rows and m columns and are searching for solutions of length L. Row i of the lattice is used to form the set $R_i = \{r_{ij}\}$, where the first m members of the set have values of zero or one, equal to the corresponding elements of the row, and the $(m+1)$th member of the set is always zero. (This last member represents a dummy track used to flag the row as not being used in a solution set.) Using these sets to form the Cartesian product space $R_1 \times R_2 \times \cdots \times R_n$, we search for a vector $(r_{1j_1}, r_{2j_2}, \ldots, r_{nj_n})$ in the space that satisfies the criterion function

$$f(r) = \sum_{i=1}^{n} r_{ij_i} = L \tag{5}$$

and that meets the constraint that no two j_i indices can have the same value unless that value is $m+1$. (This constraint allows any column of the projection lattice to be used only once.)

The concept of backtracking is to construct the vector one component at a time, with modified criterion functions used at each step to determine if the line of pursuit still has a chance of success. When it is found that a partially constructed vector is doomed to failure, the last component is discarded, and the program backtracks to resume construction from the preceding elements.

The modified criterion function used in the scan of a projection lattice at the kth row is

$$f_k(r) = \sum_{i=1}^{k} r_{ij_i} \geq L - (n-k) \tag{6}$$

i.e., there must be enough rows left for the scan to reach the final value of L. When n and L are equal (as in our sample event), this function reduces to

$$f_k(r) = k \tag{7}$$

In Fig. 5, the backtrack program begins by selecting r_{11} from set R_1 and testing it with the modified criterion function $f_1(r)$. In this case

$$f_1(r_{11}) = 1$$

and the criterion is met. Having found the first component of our sample vector, $(r_{11}, -, -)$, the program now begins a search in R_2. The first choice is r_{21} and by applying Eq. (7) we see that

$$f_2(r_{11}, r_{21}) = 2$$

However, the constraint that a column may be used only once eliminates this path from consideration. The component r_{21} is discarded, and we move to r_{22}. Here again we meet failure because

$$f_2(r_{11}, r_{22}) < 2$$

Continuing, we find that r_{23} meets the requirements, and the sample vector is extended to $(r_{11}, r_{23}, -)$.

The search now goes to R_3. As seen in Fig. 5, r_{32} is the only member meeting the constraints, and so completes the vector (r_{11}, r_{23}, r_{32}). The solution set we have found in ${}_3E$ is then

$$S({}_3E) = \{(1,1), (2,3), (3,2)\} \tag{8}$$

At this point in the scan procedure, the position of the scan in ${}_3E$ is saved, and step three is entered.

The third step is to construct the post projection lattice, P, defined by the solution set just found in ${}_3E$ [Eq. (8)]. The index pairs in this set prescribe the posts (parallel to view 3) in the correlation lattice E that are to be used as rows in P. This projection is shown on the left of Fig. 5.

The fourth step—to fix the third index of the solution triplets—is done by a scan of P in the same manner as the scan at ${}_3E$. As shown in the figure, the solution vector (r_{13}, r_{22}, r_{31}) is the only one possible, and a solution set has been found in the correlation lattice E—

$$S(E) = \{(1,1,2), (2,3,2), (3,2,1)\} \tag{9}$$

Elements corresponding to this set are made equal to 1 in the "solution lattice" E_S.

After the solution is found in P, the scan of P will continue in an effort to find any more that might be present. If other solutions had been found, they also would have been entered into E_S.

When the scan for solution sets in the post projection has been exhausted, the program returns to step two to continue the scan of ${}_3E$ from the point where the last solution set was found. If another solution set is found, steps three and four are again used to determine if the solution in ${}_3E$ is the projection of a solution in the three-dimensional lattice. These procedures continue until the scan of ${}_3E$ is exhausted, at which time the solution lattice, E_S, will contain the elements of all solutions found in the scan. There is a second possible solution for our sample event, with the resulting solution lattice shown in Fig. 6.

The solution lattice is used as a replacement for the original correlation lattice in order to eliminate all elements that do not contribute to some solution set. This lattice is now checked for elements common to all solutions in the lattice. These triplets are

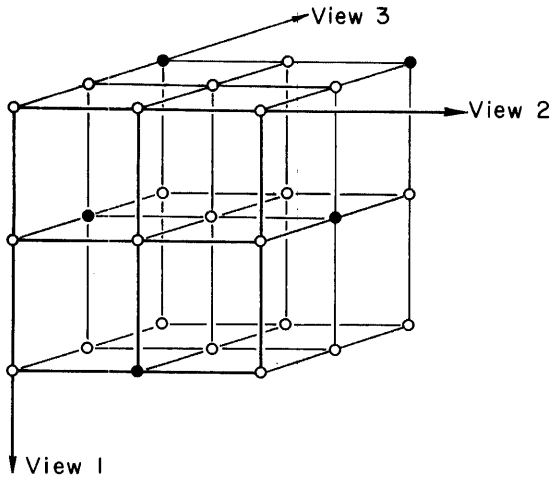


Figure 6. The solution lattice E_S showing the two remaining solutions.

fully determined and need no further testing. Their elements are set equal to zero, their indices are saved to become a part of the final solution, and the required solution length in E is reduced accordingly. In Fig. 6 we see that the triplet (3,2,1) is of this type and can be eliminated from the lattice. This leaves only four elements to be tested by the second test sequence, thus illustrating the power of what seemed to be a rather weak first test that could eliminate only 5 image pairs out of the 27 tested.

THE MULTIDIMENSIONAL CORRELATION LATTICE

The properties and procedures described thus far for the correlation lattice can be extended to the case of n stereo views requiring an n-dimensional correlation lattice. If m_i topological characteristics are seen in view i, the lattice will have dimensions of m_1 by m_2 by ... by m_n. Solutions in this lattice will be made of L n-tuplets chosen with the same constraints as those used in the three-view case.

In this general case, the tests of image combinations can compare any number of views from 2 to n, and the concept of the pair and triplet test is extended to the more general k-tuplet test, a binary function of image indices from k views.

It is possible to form various "orders" of projections of an n-dimensional lattice. The first-order projection, ${}_kE$, that we used in the three-dimensional case still eliminates only one view, but it produces an $(n-1)$-dimensional projection lattice. The elements of the ${}_1E$ projection are now written as

$$ {}_1e_{j\ldots k} = \overset{m_1}{\underset{i=1}{.OR.}}\ e_{ij\ldots k} \tag{10}$$

The second-order projection, ${}_{jk}E$, eliminates two of the views and is formed by projection from the appropriate first-order projection. In this procedure, the projection operation is commutative, i.e.,

$$ {}_{jk}E = {}_j({}_kE) = {}_k({}_jE) \tag{11}$$

In general, it is then possible to discuss the kth-order projection in which k of the views have been eliminated by successive projections.

An $(n-k)$-tuplet test will use the kth-order projection lattices to determine which image combinations it is to test and to record the results. However, the masking of the projections back into the lattice may be done in different ways. Storage requirements may prevent the keeping of all intervening projections to allow successively higher order masking back onto the n-dimensional lattice. It may be more efficient to record the test results directly into the lattice E; i.e., if the value of the test function is zero, then all elements in E that use the same indices used by the test function are set to zero.

The scanning procedure for the n-dimensional lattice is simply a continuation of the three-dimensional scan. A solution set in the two-dimensional projection lattice, ${}_{3\ldots n}E$, defines the posts in the three-dimensional lattice, ${}_{4\ldots n}E$, to be used in the construction of the post-projection lattice. A solution set in this lattice then defines posts in the four-dimensional lattice, and a second two-dimensional post projection is constructed. This continues as far as the selection of posts from the n-dimensional lattice E in order to determine the final index of the n-tuplets forming the solution set.

SUMMARY

The correlation lattice and the iterative backtrack scheme of scanning for solution sets of matching triplets were used as the basis for a code written in the FORTRAN IV language to match track images of nuclear-particle events. This code has been running on the IBM 7094 for over a year, and timing studies show that it can match images of events with seven images in each view in less than one half second, with most of that time devoted to computation for the various image-combination tests.

As developed, the program is divided into two logically separate sections (test and logic) with a

simple interface. The tests are represented as FORTRAN logical functions which have values ".TRUE." or ".FALSE." corresponding to the possibility or impossibility, respectively, of matching the image combination being tested. This division allows the tests to be developed independently of the logic section of the code and permits easy development of new programs using the same solution-finding technique.

The literature offers several examples of the use of backtrack programming.[1-3] As Golomb and Baumert so candidly state in their excellent summary of the technique,[1] "Backtrack has been independently 'discovered' and applied by many people." We regret that we are a member of those ranks. However, the notion of using backtrack in an iterative sense as we have done seems to be new. We feel that backtrack in any form may offer other people a very useful tool and hope that wider publication of the method will result in fewer independent discoveries.

ACKNOWLEDGMENTS

We thank Dr. Frank Solmitz and Dr. Margaret Alston for their suggestions and encouragement during the development of the concepts and methods of the correlation lattice.

REFERENCES

1. S. W. Golomb and L. D. Baumert, "Backtrack Programming," *Journal of the Association for Computing Machinery,* vol. 12, no. 4, pp. 516–524 (Oct. 1965).

2. R. J. Walker, "An Enumerative Technique for a Class of Combinatorial Problems," *Proceedings of the Tenth Symposium in Applied Mathematics of the American Mathematical Society*, vol. 10, American Mathematical Society, Providence, R.I., 1960, pp. 91–94.

3. J. D. Swift, "Isomorph Rejection in Exhaustive Search Techniques," ibid, pp. 195–200.

A PATTERN RECOGNITION TECHNIQUE AND ITS APPLICATION TO HIGH-RESOLUTION IMAGERY

R. D. Joseph and S. S. Viglione
Astropower Laboratory, Missile & Space Systems Division, Douglas Aircraft Company, Inc.
Newport Beach, California

SCOPE OF THE PROGRAM

The purpose of this project was to extend the study of the feasibility of automatic TIROS photograph analysis. A specific objective was to arrive at the design specification for a feasibility model of an automatic vortex recognition system.

The study had four main phases: 1) the development of logical design techniques applicable to the analysis of TIROS photographs, 2) experimental investigations of selected design parameters on a simplified problem, 3) the simulated design of a vortex recognition system using actual TIROS photographs, and 4) the consideration of suitable hardware for implementing the system.

In the study of the design approach, a variation of the discriminant analysis-iterative design technique[1] was developed. A description of this technique is given in the next section along with a discussion on discriminant analysis, in which alternative pattern differences are assumed. Only one of these, one in which differences in covariance matrices of the pattern distributions are exploited, is suitable for the cloud pattern analysis. This discriminant analysis yields a quadratic discriminant surface and requires complex hardware for mechanization of the resulting first layer logic units. To simplify the implementation of the system, an approximation to the quadratic unit was developed. The approximate unit is derived from a principal axis solution, and substitutes a pair of parallel hyperplanes for the quadratic switching surface of the more complex unit. Methods have been developed to make the system invariant to changes in the brightness and contrast of the input patterns. This invariance is considerably more effective than a simple normalization of the input pattern, as it is achieved by making each logic unit invariant to such changes. The iterative design process itself is a means for assigning output weights to the logic units, and for emphasizing the difficult patterns. In common with the popular error correction methods, iterative design will find a solution whenever it is possible to assign these output weights to give perfect performance on the sample patterns. Unlike error correction, iterative design provides an "optimum" set of weights when no solution exists, and maximizes the switching surface to pattern distances when one does exist.

A portion of the experimental program was performed on a simplified problem using low-resolution, hand-printed alphabetic characters. These studies were used to investigate selected aspects of the design technique, rather than its application. The simplified problem permitted a very much more extensive investigation than would be possible on the cloud patterns. The average computer simulation time (IBM 7094) to design a recognition network for cloud patterns was five hours—for the alphabetic characters, five minutes. Two sets of

alphabetic patterns were obtained, each consisting of 20 examples of 12 letters. One of these sets was used to design recognition networks, the other to test them. The topics studied include the cost of providing contrast and brightness invariance and the cost of the parallel hyperplane approximation.

Certain topics, namely those which are particular to TIROS photograph analysis, or the application of the design technique to cloud pattern analysis, are not suitable for the sample problem investigation. Studies were therefore performed on actual TIROS cloud photographs. In one study, the resolution of negatives taken from the TV display were modified by a screen process. Prints of these modified resolution frames were examined by a large number of people to determine the resolution requirements of untrained personnel for vortex recognition. The remaining studies were performed on patterns stored as digitized video signals. The study of primary interest was whether or not a vortex recognition network could be designed that would give good generalization results, i.e., a network capable of recognizing vortices in cloud photographs not included in the training or learning sample. Pattern sets of 1000 patterns and 200 patterns were utilized, the former for network design and the latter for generalization testing. Other topics studied were the effect of changing the resolution of the design set of patterns, the effect of changing the rate of network design, the effect of merging several network designs, the effect of limiting the input fields of the logic units, and the effect of varying a parameter which controls the proportion of patterns which turn the logic units on.

THE APPROACH INVESTIGATED

The Design Technique

The design technique for designing networks of the general perceptron structure (Fig. 1) is based on a loss function. To each pattern in a sample of classified patterns a loss is assigned. The particular form of the loss function employed assigns the loss L_i to the ith sample pattern

$$L_i = \exp\{\delta_i(\theta - \beta_i)\} \tag{1}$$

where θ is the threshold of the response unit, δ_i is plus or minus one, depending on the desired classification of the ith pattern, and β_i is the value of the discriminant function for the ith pattern.

$$\beta_i = \sum_j w_j \alpha_j^i \tag{2}$$

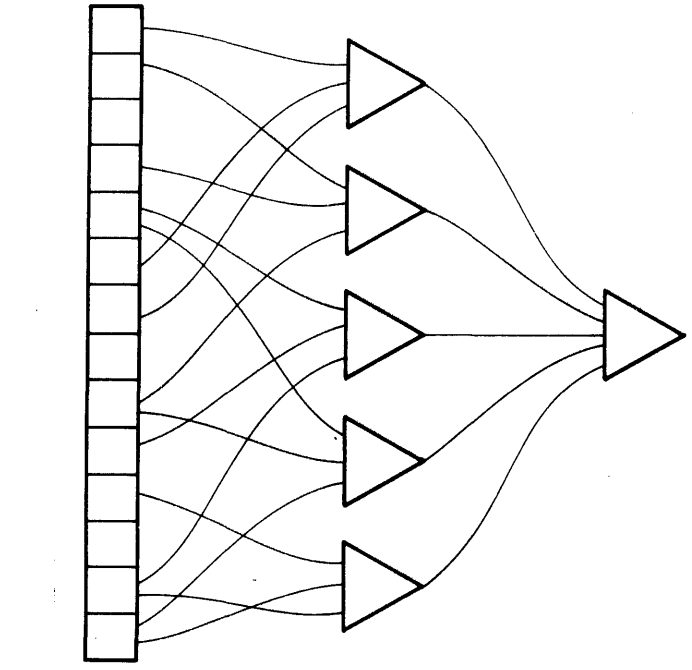


Figure 1. Decision network structure.

where w_j is a weight assigned to the jth logic unit, α_j^i is one if the jth logic unit is activated by the ith pattern and zero otherwise. The system loss is defined as the sum of all of the sample pattern losses.

Iterative Design. In the work of Highleyman[2] and Bledsoe,* the weights, w_j, and the threshold, θ, are established by minimizing the loss function. "Discriminant-analyses/iterative-design" shares the usage with the earlier techniques. The current approach extends these techniques, utilizing the loss function for two additional purposes.

The recognition logic is designed sequentially. Logic units are designed using discriminant analysis. As logic units are added to the design, the pattern losses indicate which of the sample patterns require the most attention. In acquiring the data required to generate additional logic units, the loss assigned to a pattern is used to weight the contribution of that pattern to the group statistics. This insures that the logic units thus derived are suitable for the part of the recognition problem as yet unsolved.

Logic units are generated by selecting a subspace of the original signal space, and performing a discriminant analysis in that subspace. Subspaces are used since it is usually not practical to perform the discriminant analysis in spaces with high dimensionality, nor is the implementation in analog hardware of the resulting logic unit feasible. No technique has yet been found for optimally selecting subspaces, so that an element of randomness is used.

Randomly chosen subspaces are not all of equal utility, and often a particular subspace will prove to be of no value at all. For this reason, selection is used. Each time the recognition network is to be expanded by the addition of logic units, more units are generated than are to be added. Only the best of these units are selected. The selection criterion is—which unit would result in the greatest decrease in the system loss.

The particular form of the system loss function does not permit its optimization with respect to all of the weights, w_j, simultaneously. Its optimization with respect to one weight and the threshold simultaneously, however, is readily accomplished. A relaxation process is used, cycling through all of the logic unit output weights and optimizing with respect to each weight individually. It has been shown[3] that this iterative process will lead to a desirable set of weights if the process is continued in depth. However, as new units are added to the network, different weightings for the units become desirable, so that it seems uneconomical to iterate the weight adjustment process too long on a partially designed network. The compromise chosen is to iterate the process a given number of cycles each time the network is expanded. Part of the investigation was concerned with the effects of the number of iterative cycles used.

Discriminant Analysis. Discriminant analyses are used to generate a logic unit, given a subspace. To perform the discriminant analysis, assumptions concerning the distributions of patterns in the subspace must be made. These assumptions, in general, are not justified. To some extent these assumptions are important; in other cases, they are not.

In the cases under investigation, two pattern classes are considered. The two pattern classes are assumed to have multivariate normal distributions. The statistical data which must be extracted from the patterns is thus limited to the first two moments but this does not seem to be a serious shortcoming for this problem. Experience with discriminant analysis with a normality assumption has shown it to work well even when the distributions are not normal, provided, of course, that the pattern classes may be separated on the first two moments of their distributions. Further, the selection and iterative design procedures evaluate and weight logic units according to their actual performance on the sample patterns rather than the performance predicted for them by the discriminant analysis.

The assumptions which are made concerning the mean vectors and the covariance matrices of the two distributions seem more important. These assumptions control the nature of the differences between the distributions which are to be exploited. They also control the nature of the resulting logic units.

An assumption, which results in easily implemented logic units, is that the covariance matrices are equal. The pattern class distributions differ only in their mean vectors. Under this assumption the optimum decision boundary for a logic unit is a hyperplane—one which bisects the line segment joining the mean points of the two pattern classes. The estimate of the common covariance matrix is used to establish the orientation of this hyperplane. The resulting logic unit is the linear input threshold unit almost universally used in perceptron work. In the following material, this technique is designated "Oriented Hyperplane." Figure 2a illustrates the

*Personal communication.

oriented hyperplane in two dimensions (in this case the hyperplane is a line).

The calculation can be simplified by ignoring the orientation of the hyperplane. The logic unit boundary is the hyperplane which bisects the line segment joining the class means and which is perpendicular to that line segment. This approach eliminated the need for estimating and inverting the covariance matrix. The same logic unit would result if it were assumed that the common covariance matrix was a multiple of the identity matrix. This technique is designated "Perpendicular Bisector" and is illustrated in Figure 2b.

The assumption of the preceding techniques—that the pattern class distributions differ in their mean vectors—does not fit the cloud pattern recognition problem. It can be shown that if patterns are to be recognized in all rotations and translations, then the mean vector for a pattern class will have all components equal. These components will equal the average intensity of all patterns of that class. It would appear that the mean vectors should be assumed equal, and the common mean vector should have equal components. The pattern classes are differentiated by their covariance matrices.

If it is assumed the covariance matrices are different, then the decision boundary becomes a quadratic surface. The assumption concerning the mean vectors does not alter the general form of the solution, just its computation. The technique which assumes the mean vectors equal, and the common mean vector to have equal components is called "Quadratic Surface." It is illustrated in Fig. 2c.

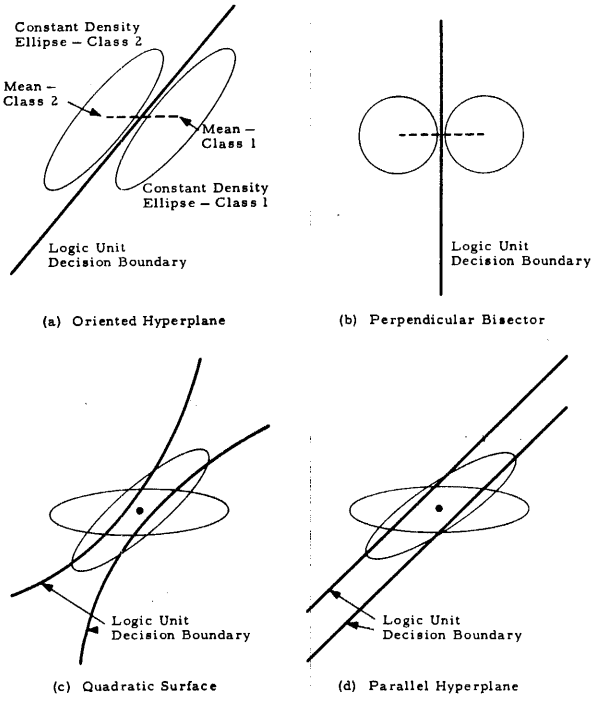


Figure 2. Discriminant functions.

Quadratic units are difficult to implement in hardware, and require large amounts of memory in a digital simulation. An approximation is possible in which the principle axis of differentiation is computed. A pair of parallel hyperplanes perpendicular to this axis is substituted for the quadratic surface. The resulting logic unit is easily implemented. This technique is designated "Parallel Hyperplane" and is illustrated in Fig. 2d.

Details of the computations for these four techniques are given in Ref. 3. The techniques are summarized in Table 1.

Table 1. Technique Comparison

Technique	Difference Exploited	Remarks
Oriented Hyperplane	Mean vector	Linear input threshold unit
Perpendicular Bisector	Mean vector	Same—easier computation
Quadratic Surface	Covariance matrix	Quadratic input threshold unit
Parallel Hyperplanes	Covariance matrix	Linear input–double threshold unit

Gray Scale Invariance. The gray scale of TIROS frames vary from picture to picture, and the darker shades on one picture may actually be lighter than the brighter areas of another frame. Indeed, such variation can occur in different areas within a single frame. It seems desirable that an invariance to such changes in the gray scale be built into the logic of a decision network, rather than obtained by providing a multiplicity of hardware to operate at different gray scale levels. A single normalization of the gray scale of a frame does not provide the degree of invariance of the techniques used in this study.

For each of the techniques previously described a version which was invariant under linear transformations of the gray scale was devised (that is, when the intensity x is replaced by $a(x + b)$). This was accomplished by determining the restrictions placed on the decision boundary of a logic unit by the requirement that its switching surface be such that the unit's state is unaffected by changes of picture brightness or contrast. Next, it was determined what conditions were required in the discriminant

analysis to assure that the restrictions on the decision boundary would be met. Finally, the brightness and contrast of the sample pictures were modified to yield the desired conditions.

As an example, the Perpendicular Bisector (which produces linear logic unit boundaries) will be considered. It is easily shown that for these units to have the desired invariance, the threshold must be zero, and the weights of the input connections must sum to zero. To insure that the weights sum to zero, the weight vector is the difference between the mean vectors for the two pattern classes within the subspace, each mean vector was required to have the appropriate coordinates summed to zero. Thus, in the computation for each unit, for each sample pattern, a value of "b" is computed which will produce the desired zero sum, and the pattern components reduced accordingly. Similarly, a zero threshold occurs when the mean vectors within the subspace have equal length. This in turn is accomplished by scaling each pattern so that the (zero sum) means have equal length. For the generation of each unit, one scale factor "a" is derived and is applied to all patterns of the negative class. For this technique (Perpendicular Bisector), it is not necessary to treat individual patterns, as the mean vectors themselves may be reduced and scaled to the same end. The consideration above shows, however, that this reduction and scaling is legitimate under the assumption that all patterns which differ only in brightness and contrast are equivalent.

SAMPLE PROBLEM INVESTIGATION

A study of a simplified problem—alphabetic character recognition—permitted a much broader investigation, as the computer time required for a single network design averaged 1/60 that required with the actual TIROS frames. The investigation was for the most part restricted to problems concerning the design technique itself, rather than its application to a specific problem. Eight variations of the basic design technique were studied, these differing in the means for generating property filters, and four variations on these to produce units invariant to changes in the gray scale. Two of the basic methods exploit differences in the means of the distributions of measurement vectors for the pattern classes, one method representing a computational simplification of the other. The other two basic methods exploit differences in the covariance matrices of the distributions, one method permitting much simpler implementation than the other. Several "open" perceptron techniques and a correlation technique were studied to provide a basis for evaluation.

All eight of the techniques were very effective in producing small networks giving perfect performance on the sample patterns. In general, 10 to 15 property filters were required for one binary decision on the 240 patterns.* This rate of design is too fast, and emphasizes the weakness in using only performance on a limited sample of patterns (the 240 samples represented 12 different letters) as the design criterion. Any false correlations which are found may solve part of the problem of classifying the *sample* patterns, but do not contribute to the solution of the real problem. In part, these false correlations may be detected by comparing the error rates achieved on the sample patterns used in the design with the error rates on an independent sample. When these false correlations are prevalent, one should increase the sample size, or make stronger use of distributional assumptions. For partial relief, several steps to slow down the design rate have been considered. A slower rate allows more opportunity for useful properties to be found before portions of the problem are considered solved on the basis of extraneous correlations. The presence of the false correlations is also indicated in a comparison of the results with the property filter generation methods based on differences in the means of the distributions, and those based on differences in the covariance matrices. Although little difference is shown in the rate of design, the ability of the networks to generalize to the independent sample is considerably different. As shown in Fig. 3, there is considerable variability in the design, due to randomness in the selection of subspaces for the logic units. This variability tends to mask differences which occur in the experiments performed. At least three networks were designed with each method, and the results averaged to reduce the variability.

In part, the variability is due to the high selection ratio (see Fig. 4) used, the ratio of the number of property filters generated to the number incorporated in the network. Since the subspaces are generated randomly, the evaluation of a unit to be gen-

* Figure 3 is a plot of system loss vs number of logic units for a number of network designs for each technique. The curves for each technique represent a different randomly selected starting point.

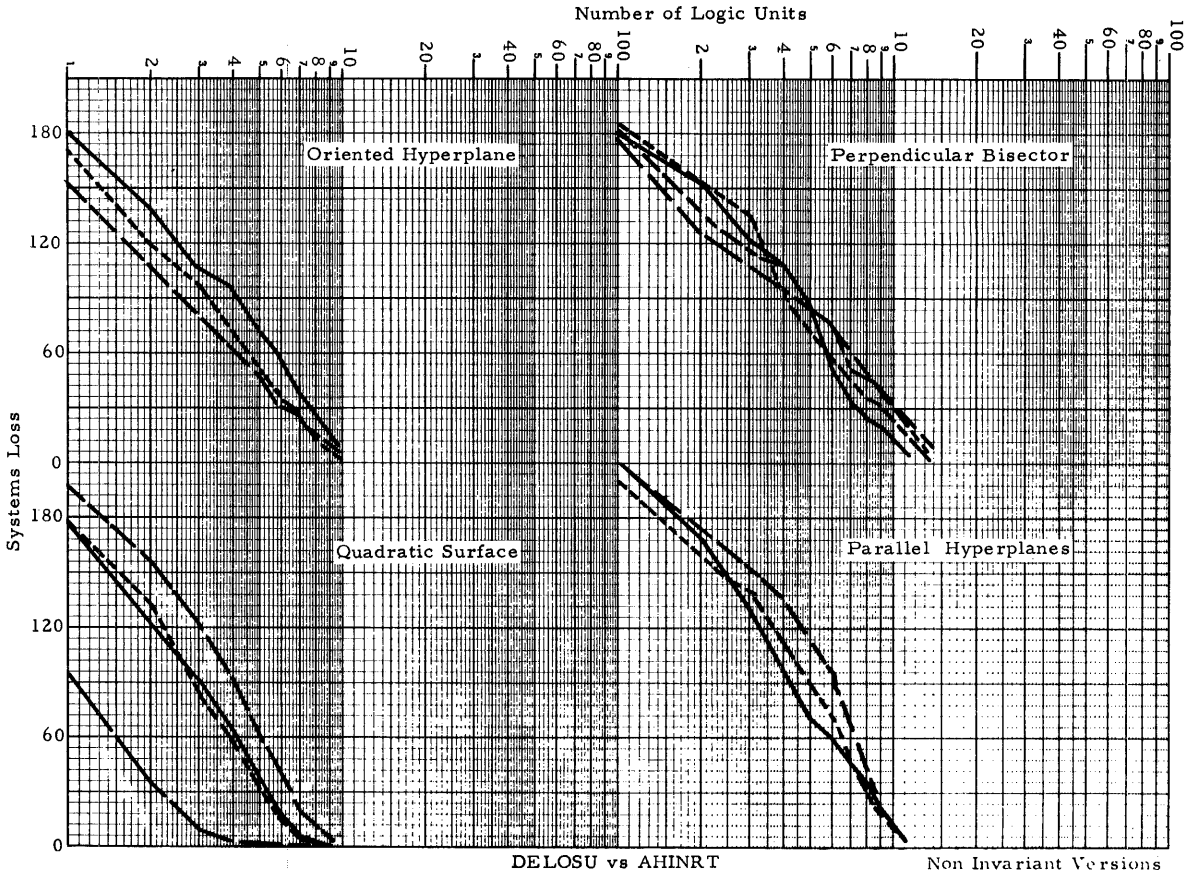


Figure 3. Effects of randomness—system loss (curves are for individual networks).

erated may be regarded as a random variable. With a high selection ratio, only the tail of its distribution is being used, a situation which induces variability. High selection ratios may be undesirable since units which are selected are usually of high utility for the classification of the sample patterns. In a complex problem, however, it is unlikely that a single property filter could solve a large portion of the real problem. High ratios could thus put a premium on the extraction of extraneous correlations.

Despite the difficulties discussed above, the design techniques worked comparatively well on the sample problem. The final network designs work considerably better, both on the sample patterns used in the design process and the independent sample, than did networks three to five times as large, designed using two "open" perceptron techniques. The utility of the current technique appears to be in the design of a set of property filters. In one experiment, the linear decision function of a network designed with the current technique was reassigned using the perceptron techniques. The resulting network classified the independent sample about as well as did the original network. A correlation method was also used. The entire set of sample patterns was stored, and unknown patterns classified by finding the highest correlation coefficient (the second highest when the sample deck was used for testing). This method showed the least effect of spurious features correlating highly with a pattern class. The error rates on the design sample and the independent sample were virtually identical. The generalization error rates were the lowest achieved. The best networks designed with the current technique came very close to the generalization error rates of the correlation scheme.

The number of freely variable weights in a network appears to be a critical quantity. Within the range investigated, the effect of the number of input connections per property filter may be accounted for by this quantity. For the linear input property filters, the number of free weights varies linearly with the number of connections. Thus a network of seven logic units, each with seven input connections

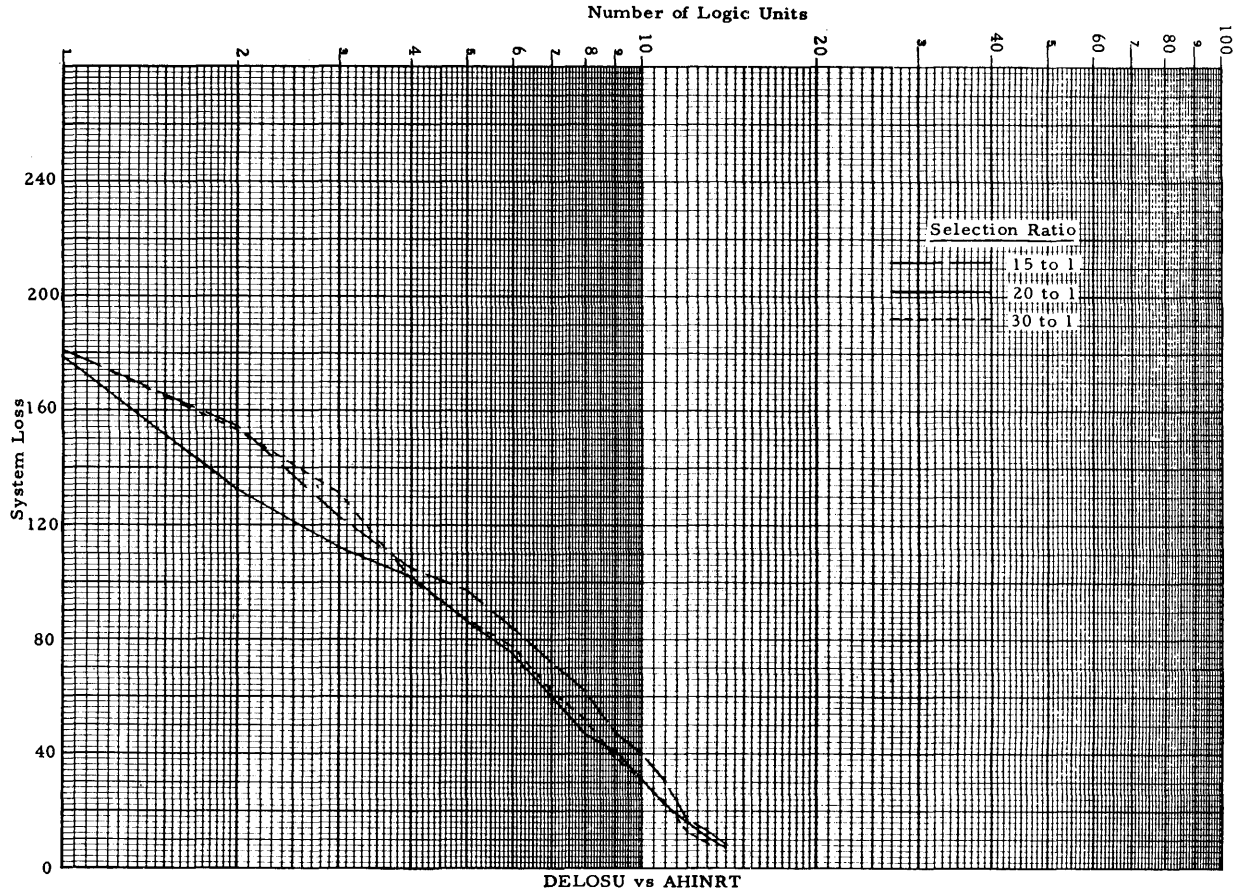


Figure 4. Effects of selection ratio—system loss.

would work about as well as one with eight units each with six connections. For the quadratic input units, the number of free weights varies quadratically with the number of input connections (Fig. 5).

The cost of making the property filters invariant to linear changes in the gray scale may also be accounted for by the number of free weights. It can be shown that the constraints which produce the invariance reduce the number of free weights by two per property filter for the linear input units, and by the number of input connections plus one for the quadratic input units. In both cases, the loss is equivalent to the loss of one connection plus one weight. The experimental results confirm these predictions. The value of the gray scale invariance is strikingly demonstrated by the generalization error rates for samples in which the gray scale has been varied both linearly (as per the invariance) and nonlinearly (see Fig. 6).

The cost of using the simplified techniques appears to be an increase of 30% in the number of property filters needed when differences in the mean vectors are being exploited, and an increase of 50–75% when covariance matrix differences are being used. In the latter case, the simplified units have less than half as many free weights, so that the total network has fewer free weights in the simplified case (see Fig. 7).

CLOUD PATTERN INVESTIGATION

Four files of digitized TIROS patterns were generated. The first and third files are frames containing vortex patterns, the frames in the second and fourth files contain nonvortex cloud cover. The first two files contain 500 patterns each, and were used in network design; the last two files of 100 patterns each were used for testing the networks. The 1200 patterns are derived from 223 actual TIROS frames, by considering different translations and rotations of these pictures. Great difficulties were encountered in obtaining and verifying these patterns. The

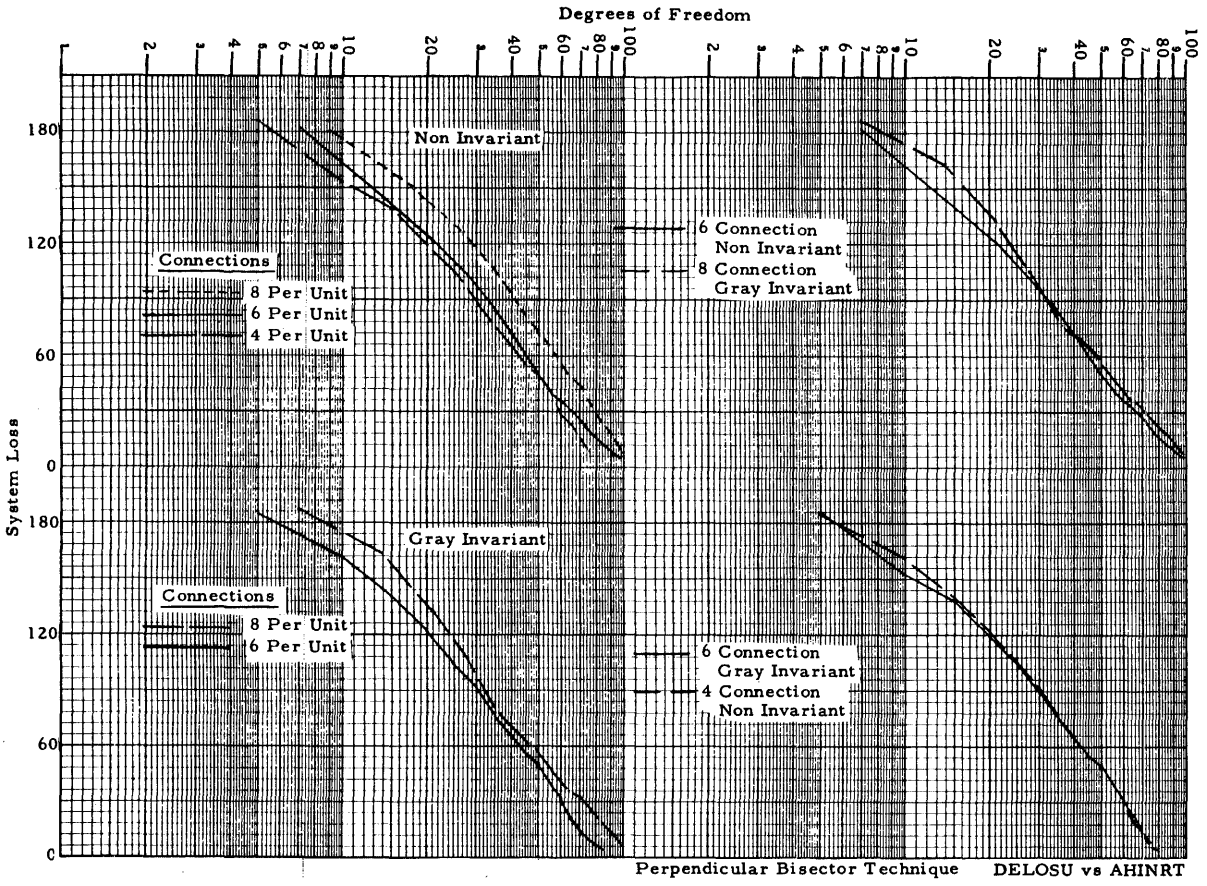


Figure 5. Degrees of freedom comparisons—system loss.

TIROS frames were edited to a 180-by-180 size, and the gray scale reduced to four bits. Examples of these patterns are shown in Fig. 8. The editing of 9/16 of the original frames was accomplished to remove as much of the horizon as possible.

Seven complete recognition networks were designed, and four fractional networks were designed. A total of 48 hours of IBM 7094 time was used to design these networks and two hours were used to test their generality. These figures are dwarfed by the amount of time used in program debugging.

The partial network designs were obtained to adjust two parameters of the design technique. One of these parameters controls the expected fraction of nonvortex patterns which turn each property filter on, the other parameter controls the rate of design and the stability of the weight adjusting procedure. Three partial networks were available for selection of each parameter. Instability did not occur in these networks (although it had occurred with different values in the debugging runs). Values which gave the fastest design rates were selected for these parameters. In retrospect, this criterion does not seem optimum. The parameter value selected by this method gives an expected activity rate for nonvortex patterns of one-half of the property filters.

Approximately 250 property filters are required to provide perfect separation of the 1000 patterns in the design files. The actual figures range from 220 to 290. Designs with more than 250 units occurred only when constraints were placed on the property filters. Steps to terminate the design process are taken when perfect performance on the design patterns is obtained. This termination procedure is such that it produces designs with 10 property filters more than required for the zero error rate. These units and the additional adjustments made to the output weights appear to reduce performance levels rather than improve them. The error rates on the design sample drop rapidly until they reach 1%, at which point a sharp break is noted in the rate of reduction. Approximately 180 property filters are required to achieve the 1% error rate (see Fig. 9).

Two generalization error rates were computed,

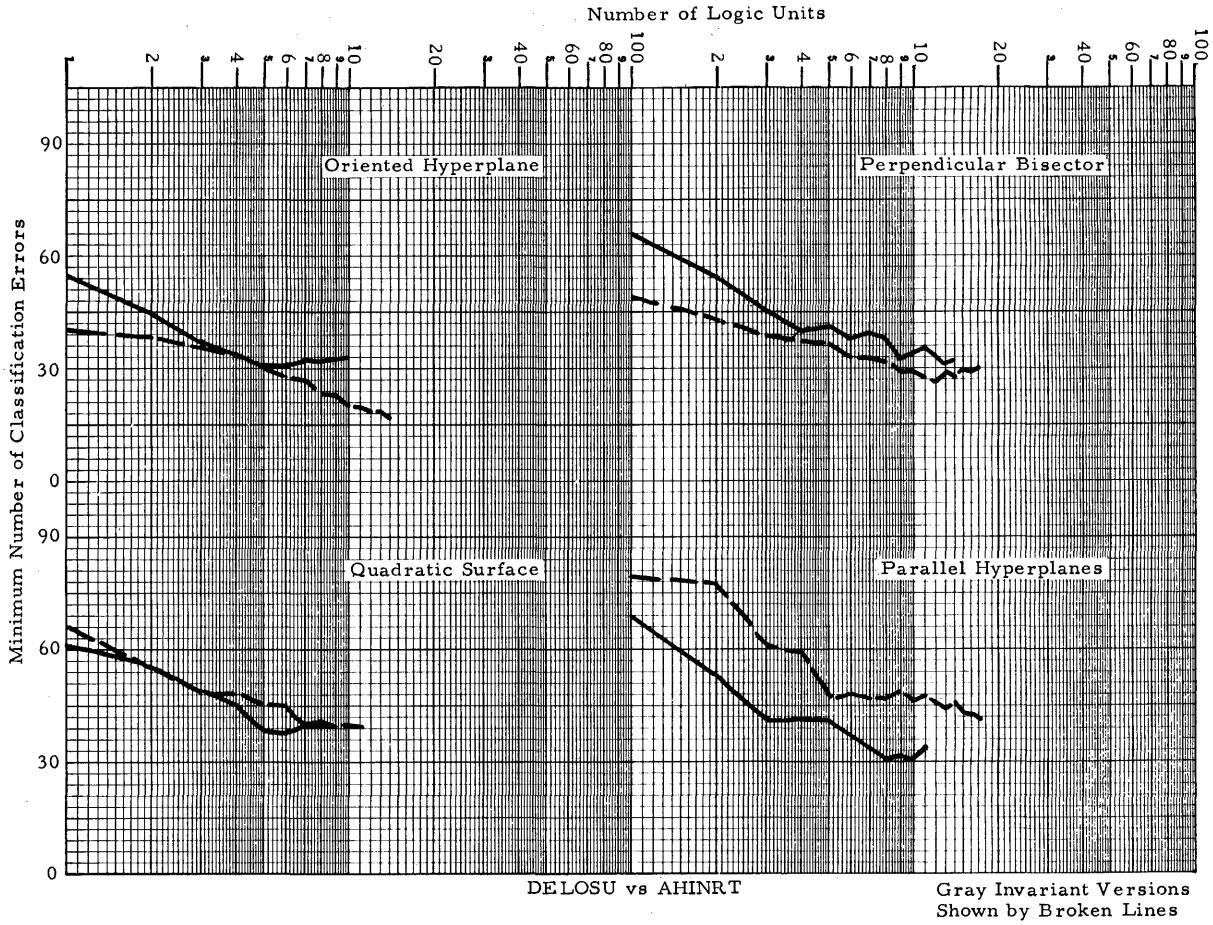


Figure 6. Effects of gray scale patterns–Deck II, VAR

corresponding to different thresholds for the decision element. One of these thresholds gives a minimum rate on the generalization patterns. These error rates are computed for each increment of 10 property filters to the network. The error curves generally show two significant local minima. Usually these minima occur near the beginning of the design and in mid-design. Occasionally, the minimum near the beginning is replaced by one near the end of the design. The mid-design minimum occurs when the error rate on the design patterns is about 5 or 6%. The mid-design minimum is usually the larger of the two (see Fig. 10).

The first three networks were designed under standard conditions. Different random numbers provide different input connection combinations and hence different networks. These networks provide the best generalization performances of the seven networks, giving error rates which are not particularly low. Network 1 has 250 property filters. The first minimum gives a 36.5% generalization error rate with both thresholds—the lowest rate observed. The mid-design minimum gives the rates of 37.5 and 39.5%. The highest error rate is 46%. Network 2 has 250 property filters, early error rates of 38 and 43.5%, and mid-design minima of 41 and 42.5%. The maximum error rate is 48%. Network 3, with 220 property filters, exhibits only mid-design minima which are 39 and 44%. At one point in the design an error rate of 51% is observed. Thus each network is capable of making less than two errors in each five decisions at some point in its design. Although these performances are not high, the networks do achieve better than chance performance quite consistently.

When the first three networks are considered in combination, the generalization error rates are not better than those for the best single network. The error rate curves are smoother, however. Failure to improve the single network performance is in-

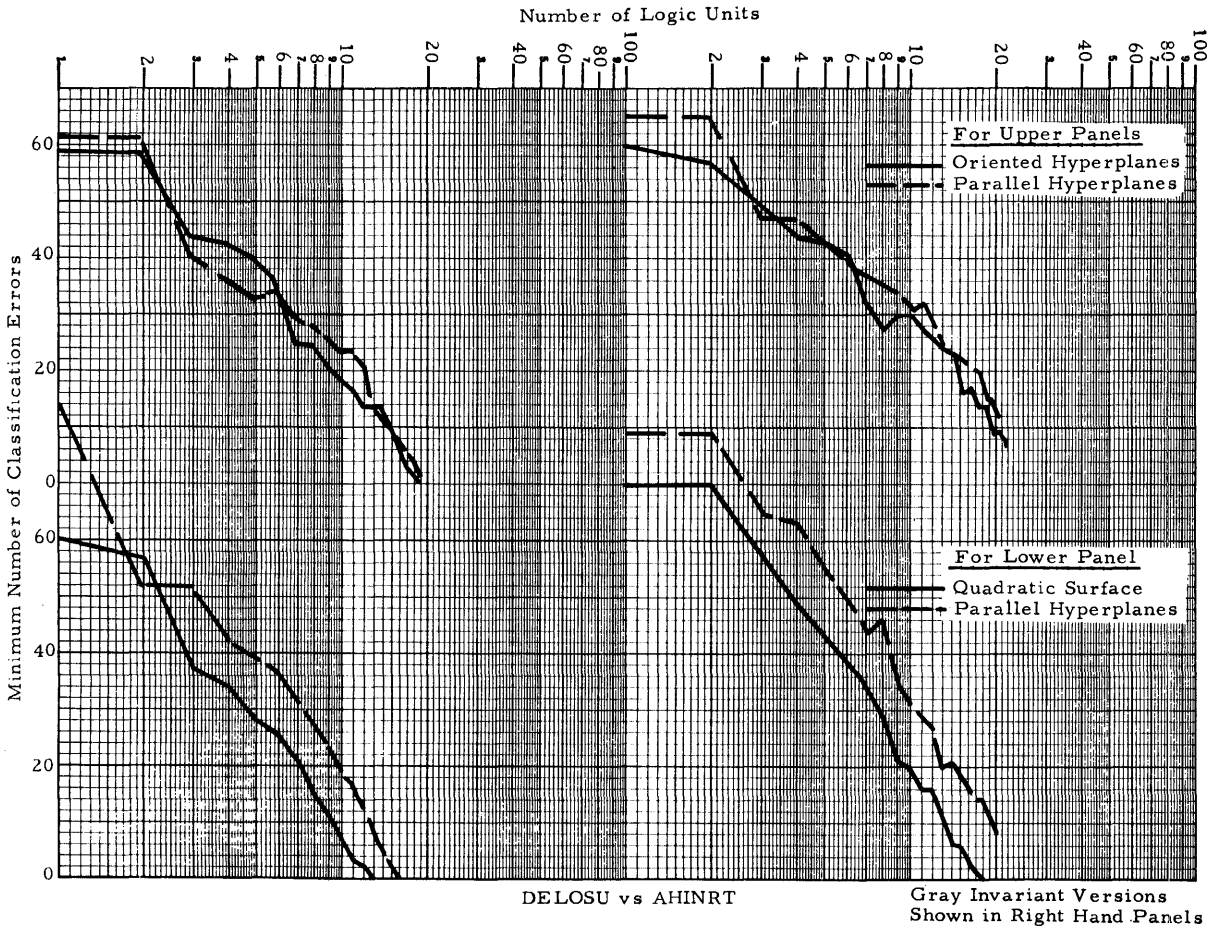
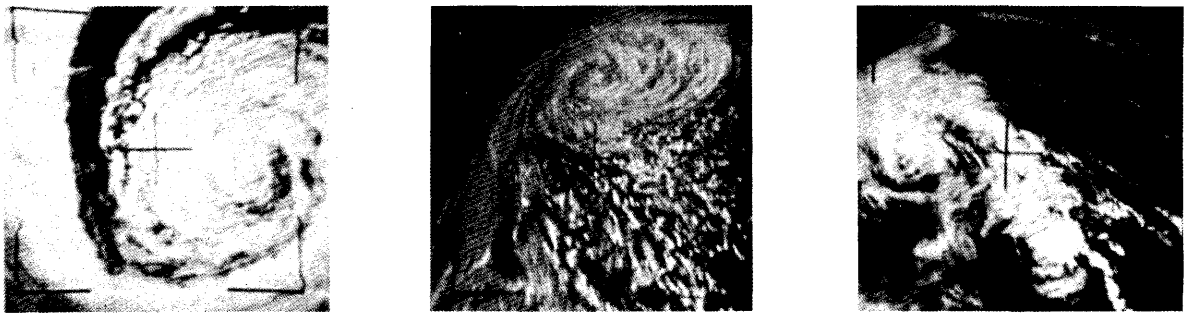


Figure 7. Technique complexity comparison—learning rate.

dicative of a nonrepresentative design sample rather than too fast a design rate.

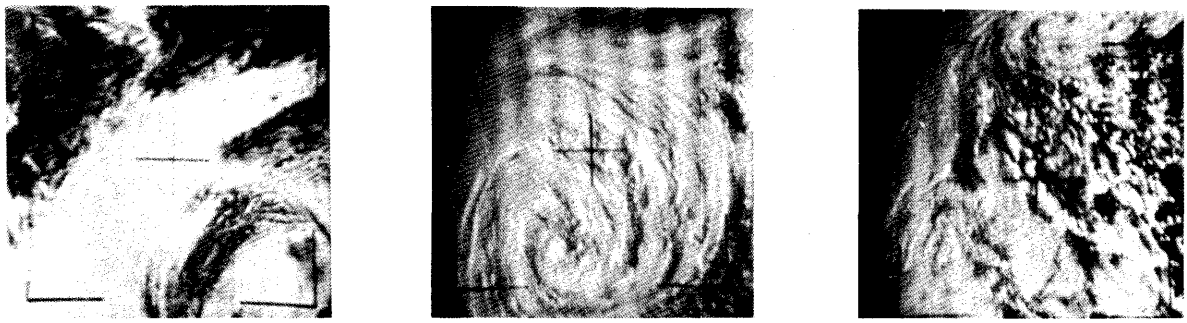
Networks 4 and 5 were designed on reduced resolution design patterns, using 90-by-90 and 60-by-60 rasters, respectively. The generalization performances were tested on full resolution patterns, however. Network 4, (see Fig. 11) with 240 property filters has early minima of 41 and 42%, and mid-design minima of 43.5 and 46.5%, these latter occurring at different points in the design. Performance no better than chance is observed 12 times in the design. Network 5 has 250 units and shows pronounced minima only near the end of the design. These minima are 41 and 41.5%. Performance no better than chance is observed 10 times in the design. These figures represent significant deteriorations from the first three networks. Combined with optical studies, which indicated that full resolution is necessary for many patterns, this result indicates that the property filters in the first three networks, at least in part, are dealing with the proper level of detail.

Networks 6 and 7 were designed with restricted area subfields for the property filters, No. 6 with 90-by-90 subfields, No. 7 with 45-by-45 subfields. They required 280 and 290 property filters. The rate of design was similar to the other networks until a 1 or 2% error rate on the design patterns was achieved. Unlike the first four networks, the generalization error rate did not increase during the remaining network design. The minimum error rates for network 6 (see Fig. 12) are 41.5 and 44.5%. Performance is no better than chance seven times. For Network 7, the minimum error rates are 43.5 and 45%. The design fails to better chance performance 19 times. These results are significantly poorer than those with the first three networks. The conclusion is that local properties are not desirable for this problem when 10 input connections are used for each property filter.



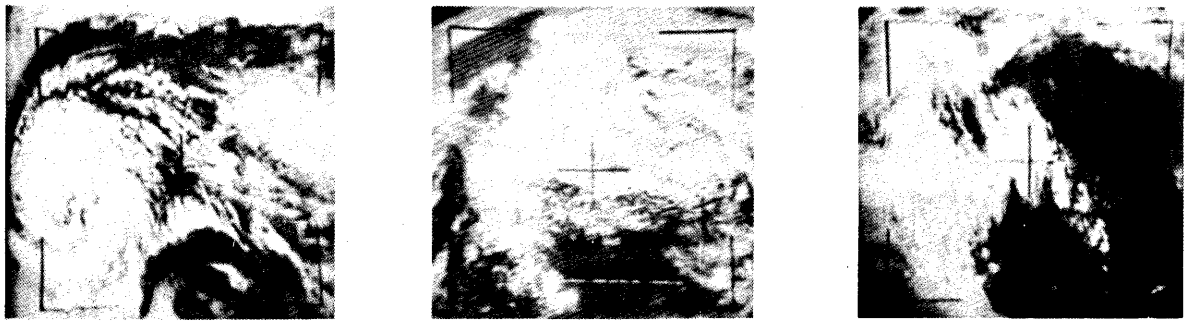
(49) (50) (36)

Variations in Vortex Size



(22) (28) (31)

Variations in Picture Contrast



(45) (18) (55)

Differences in Pattern Definition

Figure 8. Examples of cloud patterns.

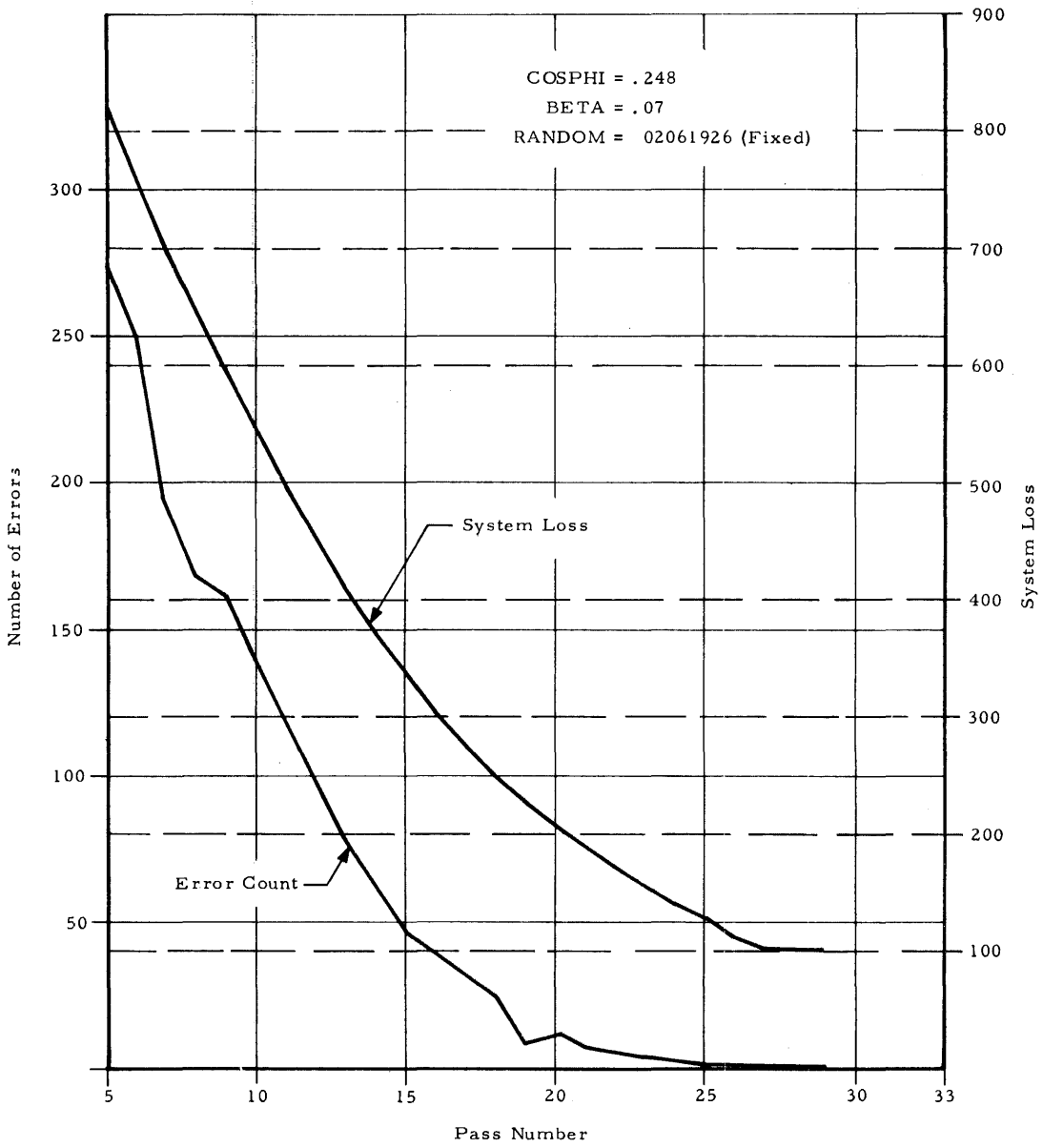


Figure 9. Machine design #1.

Optical studies were performed on TIROS frames. The resolution of 60 frames was altered to several levels using a screening process (see Fig. 13). The prints obtained were then examined by 15 to 20 people, in order of increasing resolution. The resolution level at which their decisions became consistent with their final judgment was noted. For a remarkable number of frames, the panel disagreed in their final judgment. When the resolution was too low, the frame was almost always judged nonvortex. Consequently, when the final judgment was nonvortex, all preceding judgments were also. About half of the vortex decisions were reached at the highest resolution level (240 lines per frame). The other half were made at the coarsest resolution levels (about 70 lines per frame). Very few final judgments were made at intermediate levels.

SUMMARY AND CONCLUSIONS

An extensive study of a particular design technique for pattern recognition, and particularly its application to TIROS photographs, has been conducted. Many aspects of the technique which were

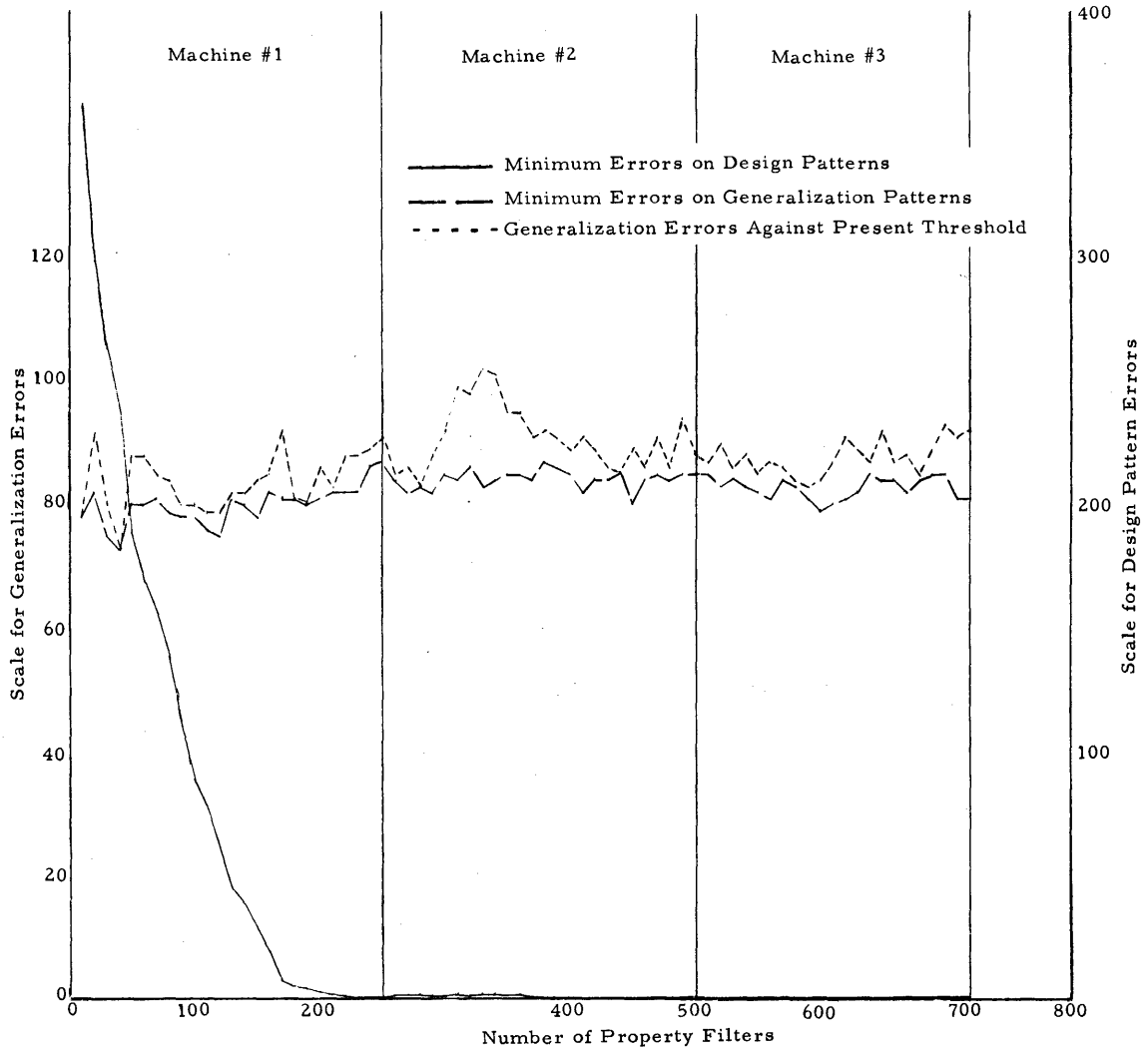


Figure 10. Network performance—machine #1 plus #2 plus #3.

not specific to this application were investigated in a series of experiments on alphabetic characters. Other experiments were performed on data from actual TIROS frames in an attempt to separate frames containing vortices from those which do not.

Automatic design techniques for pattern recognition networks may be given varying degrees of coupling to a sample of patterns. At one extreme, the technique may be tightly coupled in that the only design criterion is the correct classification of the design sample of patterns. The drawback to this extreme is that even if the sample of patterns is very representative of the actual distributions, the finiteness of the sample can lead to networks giving poor generalization results. A nonrepresentative sample almost insures poor generalization. The advantages are 1) that there is no need to make distributional assumptions; and 2) that since the goal is a high level of performance on all patterns, and since it is unlikely that the generalization performance will exceed the performance on the design sample, perfect or near perfect performance on the design sample is a good starting point. At the other extreme are the techniques with minimal coupling to the design sample. A design is accomplished by making strong assumptions concerning the actual distributions of patterns, using the sample to estimate a limited number of parameters of these distributions. The disadvantages are that if the distributional assumptions are incorrect, then poor performance is obtained. Even if the assumptions are nearly correct, the technique usually results in networks which

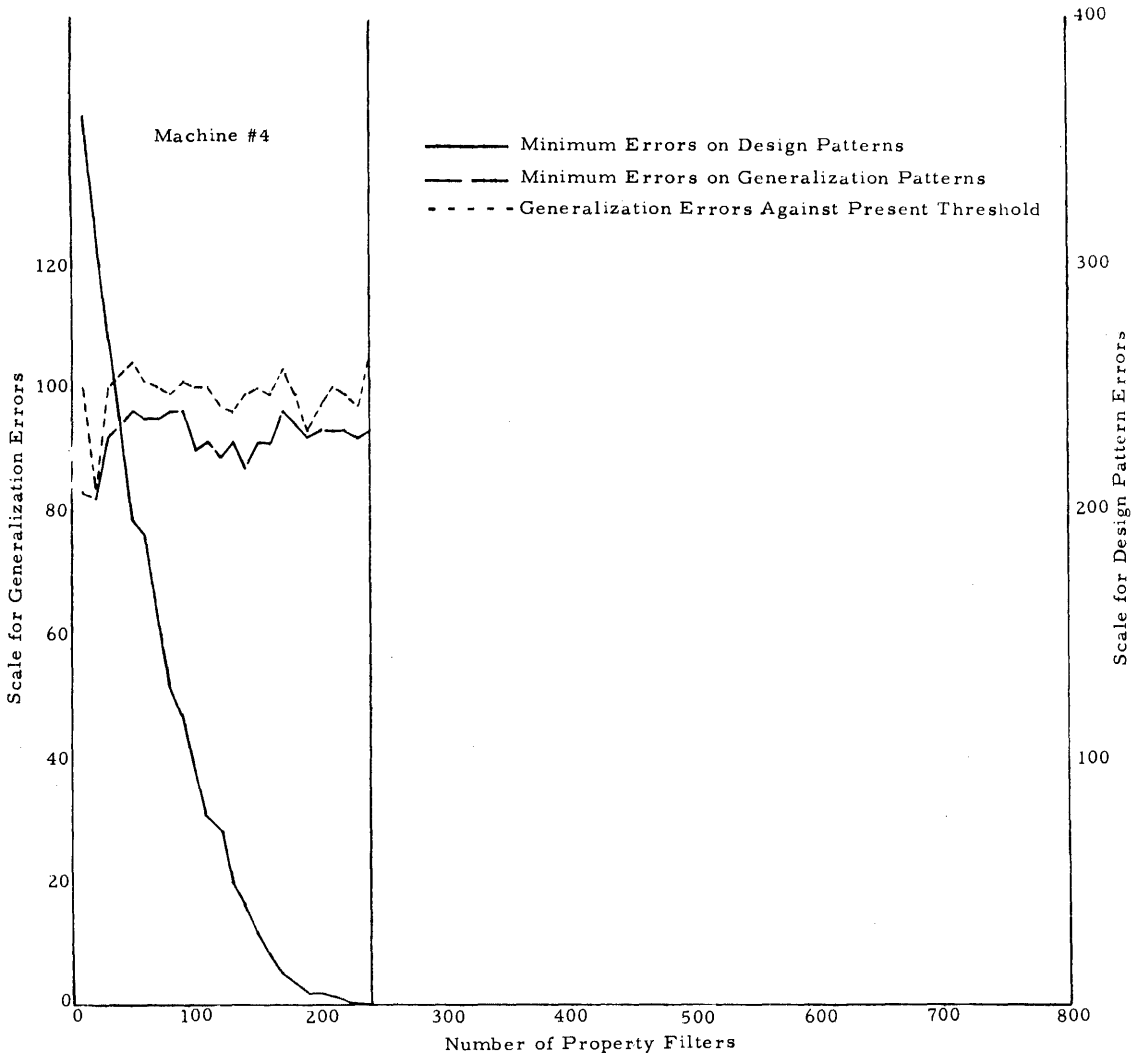


Figure 11. Network performance—machine #4.

classify a majority of the patterns correctly, but leave a substantial minority incorrect. Increasing the sample size does not help usually, for although the less common patterns are better represented, only a few average values are extracted from the sample. The advantage is that the number of sample patterns required by the technique is minimized, since the technique is not nearly as sensitive to noise and false clues in the sample patterns.

The present technique was selected to provide a compromise between these extremes. Initially, the design of the property filters is very loosely coupled in that populations of them are designed statistically under very strong distributional assumptions. The selection of property filters and the design of a linear discriminant for the decision element are tightly coupled to the design sample. This is done in terms of a loss function which reflects how well each sample pattern is classified. The losses are used as weightings in the statistical averages used to design property filters so that as the design process proceeds, the design of these filters becomes more tightly coupled to the sample patterns. The intent was to derive properties in a loosely coupled fashion to avoid the sensitivity to sample representativeness of the tightly coupled systems. The purpose of increasing the extent of this coupling was to avoid the common failing of loosely coupled systems—that of solving the same portion of the problem over and over, while ignoring the harder parts of the problem.

The effects of this increase in coupling are evident in the experimental programs, in which the generali-

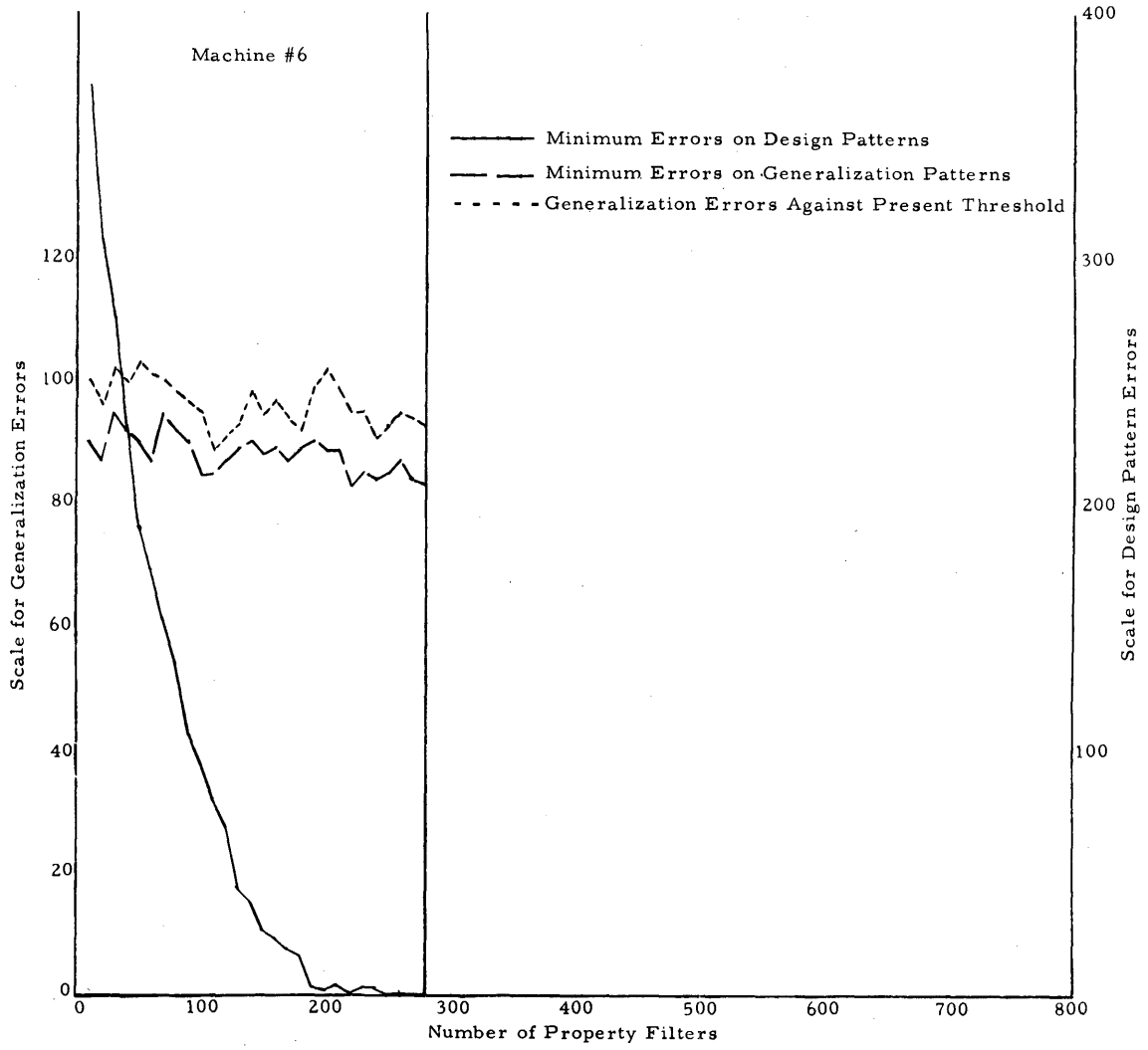


Figure 12. Network performance—machine #6.

zation performance nearly always does not improve, and often deteriorates as the last few errors in the design phase are eliminated. Overall, the design technique appears to be tightly coupled, having never indicated an inability to provide perfect performance on the sample patterns with a relatively small network.

Techniques were devised to provide property filters which were invariant to linear changes in the gray scale. Experiments on the alphabetic characters indicated that the increase in network complexity required for perfect performance on the sample patterns was nominal, although for the type of property filter selected, the gray invariance was costly in the hardware implementation.

Consideration of the TIROS frame analysis indicates that the property filters must be designed by a covariance analysis—that is, it is combinations of intensity values rather than individual intensities which are important. The alphabetic characters seem more susceptible to analysis of the distribution means, due to centering of the patterns. In the experiments on alphabetic characters both mean analysis and covariance analysis were used. On the TIROS frames only covariance analysis was used.

Normal populations which differ in their covariance matrices give rise to quadratic input property filters. These are very expensive in analog hardware, and require very large amounts of memory in digital implementations. A parallel hyperplane approximation was developed. Experiments on the alphabetic characters indicate that with this approx-

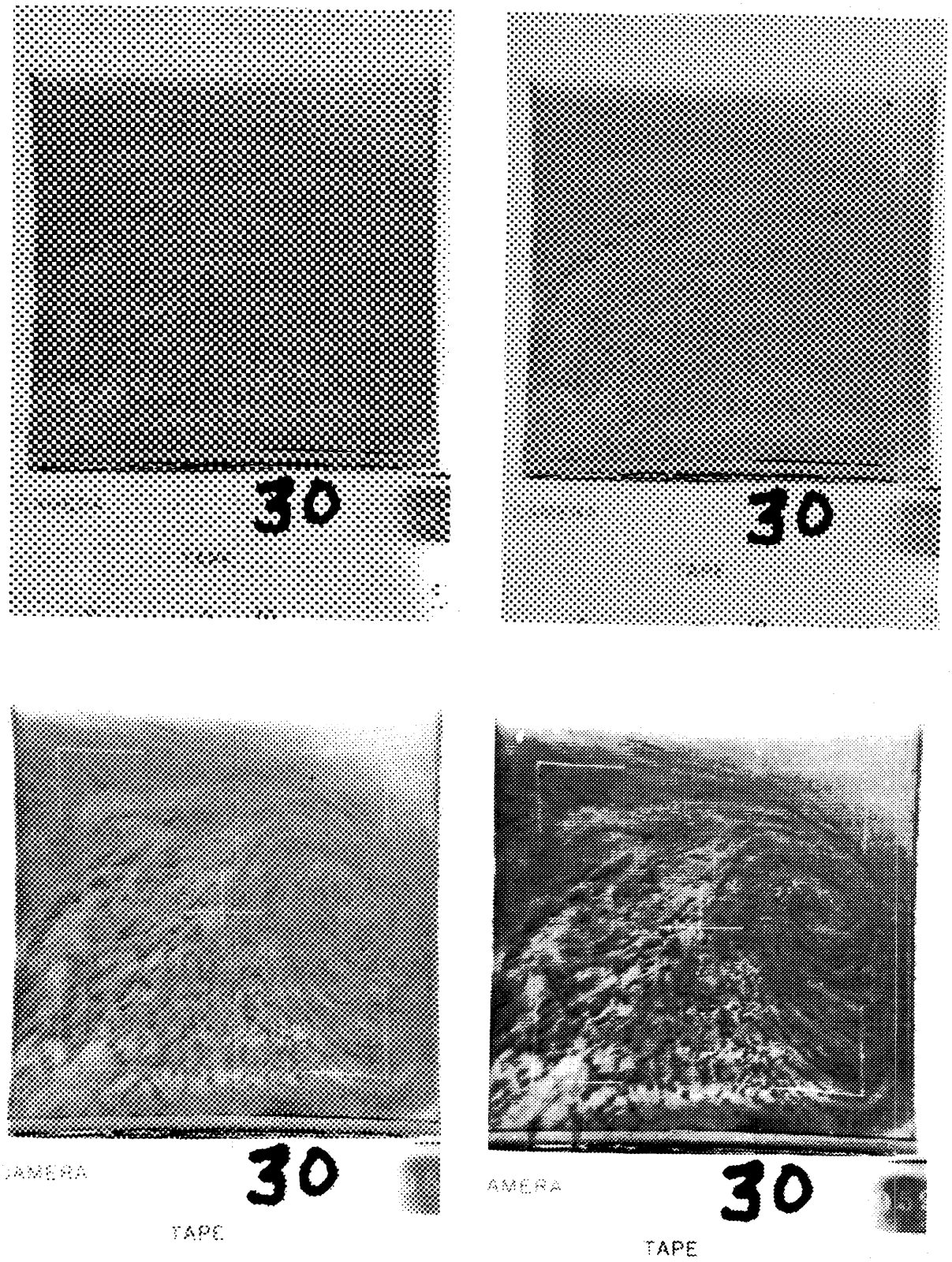


Figure 13. Pattern 30 at resolutions of 95, 111, 150 and 220.

imation the increase in the number of property filters to separate the sample patterns is not great, considerably reducing the overall system complexity. Implementation of the gray invariant version of this approximation in digital equipment is simple, but the analog form suffers from the same problems as the quadratic unit, but on a more limited scale. The gray invariant approximation was used in the TIROS frame experiments.

In the experiments on the alphabetic characters, eight methods for generating property filters were used. Four of these were designed to exploit differences in the means of the distributions, and four to exploit differences in the covariance matrices. Half of the techniques in each case produce property filters invariant to changes in the gray scale.

There was sufficient information in the sample patterns to permit each of these techniques to provide complete networks. The number of property filters required to separate the sample patterns generally ranged between 8 and 15, depending on the particular technique.

The generalization performance of these networks was reasonable. For the techniques based on differences in the distribution means, there is little difference in performance, each technique averaging about an 8% error rate. The best individual network had about a 6% error rate. The quadratic units gave a 10% error rate. The parallel hyperplane technique gave a 13% error rate and its gray invariant version yielded 15% errors. The best generalization performance generally occurred when the design was about 70% completed. Although the performance degrades only about 1% after the best level is reached, better levels might have been achieved if the coupling did not become as great as it does toward the end of the network design.

The utility of the gray invariance was demonstrated by modifying the gray scale of the sample patterns. A linear change has no effect on the performance of the gray invariant networks while having profound effects on the noninvariant ones. Even for some nonlinear changes in the scale, the effect on the noninvariant networks is much greater than for the invariant ones.

These performances were achieved on one binary decision. The design and generalization sets of sample patterns consisted of 20 examples, each containing 12 letters. These were sloppy handprinted characters. To put the results in perspective, several other design techniques were tried. The best of these, in which all of the sample patterns are stored in memory, and the unknown patterns correlated against all of these to find the highest correlation, gave a 5% error rate. In other tests perceptrons with both 50 randomly selected property filters, and 50 property filters designed by analysis of the distribution means, and forced learning and Bayes weights for decision function assignment gave 25 and 20% generalization errors respectively. Using 14 property filters taken from a network designed by the current technique (one which analyzes distribution means), the forced learning approach gave 9% generalization errors, just 1% more than the network from which the units were taken. The value of the selectivity and the coupling in the present technique is apparent.

Despite the shortcomings shown in the generalization data, and the hardware difficulties in implementation, the gray invariant version of the parallel hyperplane technique was used in the TIROS frame investigation. Unless a considerable amount of normalization can be achieved, methods based on differences in the distribution means will not be effective in photo interpretation. This leaves the quadratic units and the parallel hyperplane units. The parallel hyperplane units were selected, as they require only 22% as much memory for unit specification. The value of the gray invariance when the gray scale does change was deemed to be sufficient compensation for the slightly higher error rate on the black and white generalization patterns.

One thousand patterns were used as design patterns in the TIROS investigation. Five hundred of these contained vortices; 500 did not. An additional 200 patterns used as a generalization sample were also equally divided. This classification was accomplished by meteorologists with experience on TIROS photographs. Classification by Douglas personnel was less consistent.

Optical studies were performed on some TIROS pictures to determine the minimum resolution requirements compatible with good recognition performance by people. Sixty different frames were utilized. Prints of these frames were made at various resolutions, using a screen process to control the resolutions. These were examined by 15 to 20 people, in order of increasing resolution, to determine the coarsest resolution level at which their classification became consistent. A remarkable number of frames resulted in split decisions, nearly half of the people calling them vortex and half nonvortex, even at the highest resolution level. For the vortex patterns the decision point seems to split

evenly between the highest and lowest resolution levels (240 lines and 70 lines).

Seven networks were designed to separate the sample patterns. The first three of these were designed with 180-by-180 resolutions, with each property filter receiving 10 connections selected randomly from the entire input field. A different starting random number provides distinct networks. The first two networks required 250 property filters, the third required 220 to separate the sample patterns. Thus the networks averaged 2400 input connections, or 8% of the available picture points. Since the resolution study did not indicate that the smaller vortices could be identified at 70 lines (giving 8% as many elements as 240 lines), one network alone will not be particularly general. If the design sample is sufficiently representative, improved generalization should result if the networks are combined. Such combinations do not result in significant improvements in the generalization performance.

Each of the three networks can produce generalization error rates less than 40% at some stage in its design. Machine No. 1 is best in this respect, giving a minimum of 36.5% errors at one point, and maintaining 40% or less through most of the design. For machines 1 and 2 the error rate has a minimum at a point about half way through the design. Quite consistently, the error rate on the design patterns is 3–6% at this point. In all cases the error rate deteriorates toward the end of the design. Combining the networks overcomes this tendency for the error rate to rise. The conclusion from these data is that the increase in coupling in the design of the property filters is undesirable.

The poor generalization performance is due to the nonrepresentativeness of the sample patterns. That 500 patterns, unnormalized for size, position, horizon location, or fiducial mark location, do not adequately represent the vortex or nonvortex classes on a 32,400-point input field is not surprising. The network which achieved perfect separation on nonrepresentative samples may have accomplished some combinations of two things. The network may be a good design for a smaller class of vortex and nonvortex patterns of which the design sample is representative, or it may be a design which capitalizes on the nonrepresentativeness of the extraneous features in the samples, such as the location of the horizon or fiducial marks. The extent of the combination achieved cannot be determined without a very detailed study of the design and generalization patterns. Some indication is given in the resolution results to be discussed next. That at least some honest properties were found is evidenced by the fact that generalization performance is consistently better than chance.

Networks 4 and 5 were designed on lower resolution patterns. These maintained the 180-by-180 format, but the gray level changed only every second or third point, respectively. Generalization for these networks was tested against full resolution patterns. The networks required 240 and 250 property filters, respectively. Each network has a best generalization error rate of 41%, near the beginning of the design for Network 4 and near the end for Network 5. This significant deterioration is again indicative that some honest properties have been derived.

Networks 6 and 7 were designed with limitations on the input fields of the property filters. For Network 6 the connections for the filters were drawn from 90-by-90 subfields positioned randomly in the main field. For Network 7 45-by-45 subfields were used. These networks required 280 and 290 property filters for complete design. The rate of design is not noticeably slower than for the other networks until the error rate on the design patterns is less than 2%. The minimum generalization error rates are 41½% and 43½%, respectively, each minimum being achieved twice. Therefore, there does not appear to be any advantage, and indeed there is some disadvantage, in seeking more localized properties.

The following conclusions are drawn:

1. The design technique is capable of producing networks to separate the design sample of patterns. When this sample is representative, as with the alphabetic characters, good generalization results are achieved. In this case, the primary value of the process is in the set of property filters derived, rather than the discriminant function. When the sample is nonrepresentative, as with the TIROS frames, only limited generalization success is achieved.
2. The coupling between the design technique and the sample patterns is too tight. The increase in coupling in the property filter design should be removed or reduced, the parameter which controls the rate of design should be adjusted to give a lower rate, and perhaps the criteria for acceptance of property filters should be weakened.
3. The size of the design sample required depends upon the minimum number of picture points needed

for recognition and on the amount of normalization and "cleaning-up" of the patterns. For essentially unnormalized vortex patterns on a 32,400-point input field, 1000 sample patterns are inadequate to give a properly stratified (representative) sample.

4. If the designs achieved are to be practical, considerably more attention must be given to normalization of the patterns. A size normalization would permit the use of a resolution level suitable to all patterns rather than just the smallest ones. The resultant decrease in the number of picture points required would make the sample more representative by editing or averaging out noise, and by limiting the number of effective translations of the patterns. Local and general normalizations of the contrast in the frame could eliminate the need for gray invariant property filters, resulting in a much simpler hardware implementation of the parallel hyperplane technique, and somewhat better performance. Position normalization (i.e., centering) could make one of the techniques based on distribution means practical, or at least would simplify the recognition task.

ACKNOWLEDGMENTS

This program was sponsored by the NASA Goddard Space Flight Center. Mr. J. Silverman was Technical Officer for NASA. His assistance and suggestions during the course of the project were appreciated. The assistance of Mr. M. Maxwell of NASA/GSFC and Dr. Arking, Institute for Space Studies, Columbia University, in obtaining the digitized TIROS data is appreciated, as are their early suggestions regarding tape formatting and editing.

Numerous individuals at the Douglas Aircraft Company Computing Center in Huntington Beach and the Astropower Laboratory in Newport Beach assisted on the program. The principal contributors to the study were Messrs A. G. Mucci, A. G. Ostensoe, R. G. Runge, and M. Uemura.

REFERENCES

1. J. Daly, R. D. Joseph, and D. M. Ramsey, "An Iterative Design Technique for Pattern Classification Logic," Astropower Laboratory Paper EL-6320, presented at WESCON 1963 (July 9, 1963).

2. W. H. Highleyman, "Linear Decision Functions, with Applications to Pattern Recognition," *Proc. IRE,* vol. 50, no. 6 (June 1962).

3. "Design Study of a Cloud Pattern Recognition System," Astropower Electronics Department, MSSD, Douglas Aircraft Co. Final Report, Contract NAS 5-3866 (1965).

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*
1204 Milton Avenue
Pittsburgh 18, Pennsylvania

Secretary

MR. MAUGHAN S. MASON
Thiokol Chemical Corporation
Wasatch Division—M/S 850
Brigham City, Utah 84302

Chairman-Elect

DR. BRUCE GILCHRIST*
IBM Corporation
Data Processing Division
112 East Post Road
White Plains, New York

Treasurer

MR. FRANK E. HEART*
Lincoln Laboratory—MIT
P. O. Box 73
Lexington 73, Massachusetts

ACM Directors

MR. J. D. MADDEN
ACM Headquarters
211 East 43rd Street
New York, New York 10017

DR. GEORGE FORSYTHE
Computation Center
Stanford University
Stanford, California

MR. HOWARD BROMBERG*
CEIR, Inc.
1200 Jefferson Davis Highway
Arlington, Virginia 22202

DR. BRUCE GILCHRIST
IBM Corporation
Data Processing Division
112 East Post Road
White Plains, New York

IEEE Directors

MR. WALTER L. ANDERSON*
General Kinetics, Inc.
2611 Shirlington Road
Arlington 6, Virginia

DR. T. J. WILLIAMS
Control & Information Systems Lab.
Purdue University
Lafayette, Indiana 47907

MR. L. C. HOBBS
4701 Surrey Drive
Corona Del Mar, California

DR. R. I. TANAKA
3427 Janice Way
Palo Alto, California

Simulation Councils Director

MR. JOHN E. SHERMAN
Lockheed Missiles & Space Corp.
D-59-15, B-102
P. O. Box 504
Sunnyvale, California

American Documentation Institute Director

MR. HAROLD BORKO
System Development Corp.
2500 Colorado Avenue
Santa Monica, California

Association for Machine Translation and Computational Linguistics-Observer

DR. PAUL GARVIN
8433 Fallbrook Avenue
Canoga Park, California 91304

Executive Secretary

MR. H. G. ASMUS
AFIPS Headquarters
211 East 43rd Street
New York, New York 10017

*Executive Committee

Committee Chairmen

Abstracting

DR. DAVID G. HAYS
The RAND Corporation
1700 Main Street
Santa Monica, California

Admissions

MR. WALTER L. ANDERSON
General Kinetics, Inc.
2611 Shirlington Road
Arlington, Virginia

Award

MR. SAMUEL LEVINE
Bunker-Ramo Corporation
445 Fairfield Avenue
Stamford, Connecticut

Conference

DR. MORTON M. ASTRAHAN
IBM Corporation—ASDD
P. O. Box 66
Los Gatos, California

Constitution & By-Laws

MR. MAUGHAN S. MASON
Thiokol Chemical Corporation
Wasatch Division—M/S 850
Brigham City, Utah 84302

Education

DR. DONALD L. THOMSEN, JR.
BM Corporation
Old Orchard Road
Armonk, New York

Finance

MR. WILLIAM D. ROWE
Sylvania Electronics Systems
189 B. Street
Needham Heights, Massachusetts

Harry Goode Memorial Award

DR. ALSTON HOUSEHOLDER
Director, Math Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee

International Relations

PROFESSOR JOHN R. PASTA
Digital Computer Laboratory
University of Illinois
Urbana, Illinois

Planning

DR. JACK MOSHMAN
CEIR, Inc.
One Farragut Square, South
Washington, D.C.

Public Relations

MR. ISAAC SELIGSOHN
IBM Corporation
Old Orchard Road
Armonk, New York

Publications

MR. STANLEY ROGERS
P. O. Box 625
Del Mar, California

Social Implications of Information Processing Technology

MR. PAUL ARMER
The RAND Corporation
1700 Main Street
Santa Monica, California

Technical Program

MR. PAUL M. DAVIES
2703 Pine Avenue
Manhattan Beach, California

Newsletter

MR. DONALD B. HOUGHTON, 15-W
Westinghouse Electric Corporation
3 Gateway Center, Box 2278
Pittsburgh 30, Pennsylvania

1966 SJCC

MR. HARLAN E. ANDERSON
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts

1966 FJCC

MR. R. GEORGE GLASER
McKinsey & Company
100 California Street
San Francisco, California 94111

COSATI Liaison

MR. GERHARD L. HOLLANDER
P. O. Box 2276
Fullerton, California

Conferences 1 to 19 were sponsored by the National Joint Computer Conference, predecessor of AFIPS. Conferences 20 and up are sponsored by AFIPS. Copies of volumes 1–26, Part II may be purchased from SPARTAN BOOKS, scientific and technical division of Books, Inc., 432 Park Avenue South, New York, N.Y.

Volume	*Part*	*List Price*	*Member Price*
1–3		11.00	11.00
4–6		9.00	9.00
7–9		9.00	9.00
10, 11		7.00	7.00
12, 13		7.00	7.00
14, 15		8.00	8.00
16, 17		6.00	6.00
18		3.00	3.00
19		3.00	3.00
20		12.00	12.00
21		6.00	6.00
22		8.00	4.00
23		10.00	5.00
24		16.50	8.25
25		16.00	8.00
26	I	18.75	9.50
26	II	4.75	2.50
27	I	28.00	14.00

Cumulative Index to Vols. 1–16, Part II $3.00

Vol. 28. 1966 Spring Joint Computer Conference,
Boston, Massachusetts, April, 1966